Universidade do Minho
Escola de Engenharia

André Leite Ferreira

Methodological Approaches for Software Process Improvement in Multi-model Environments

Programa de Doutoramento em Informática das Universidades do Minho, de Aveiro e do Porto

universidade de aveiro

Universidade do Minho

U.PORTO

maio de 2016

Methodological Approaches for Software Process Improvement in Multi-model Environments

André Leite Ferreira

UMinho | 2016

Universidade do Minho
Escola de Engenharia

André Leite Ferreira

Methodological Approaches for Software Process Improvement in Multi-model Environments

Programa de Doutoramento em Informática (MAP-i)
das Universidades do Minho, de Aveiro e do Porto

Universidade do Minho

Trabalho realizado sob a orientação do
Professor Doutor Ricardo J. Machado
e do
Professor Doutor Mark C. Paulk

maio de 2016

STATEMENT OF INTEGRITY

I hereby declare having conducted my thesis with integrity. I confirm that I have not used plagiarism or any form of falsification of results in the process of the thesis elaboration.
I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

University of Minho, 30th May 2016

Full name: André Leite Ferreira

Signature:

Acknowledgements

To my family, specially my parents, my wife and my sons, for the extensive support and motivation they gave me during my research work. Without them certainly it would have been much, much harder. This work if for them.

I would like to thank my supervisor, Ricardo J. Machado. We met during the first year curricular classes of MAP-i at University of Minho. Ricardo has been a full-time supervisor and a also a friend. Many thanks for the opportunity to work with you and thanks for the final push.

I wish also to thank my external advisor, Mark C. Paulk for the inspiration and privilege to work closely with such an expert and reference in the field of Software Process Improvement. A special thanks also to CRITICAL Software S.A. and to Gonçalo Quadros for the inspiration and making this research work possible.

To all my friends at Azurém and Gualtar with whom I shared may daily work environment. It was a pleasure to work with all of you, for all a special thanks. To all of those of have directly or indirectly contributed to the work in this dissertation my sincere thanks. .

I would also like to acknowledge the following organizations for their financial support in the fulfillment of the research activities of this dissertation: Critical Software S.A. and Fundação para a Ciência e Tecnologia through the scholarship (SFRH/BDE /33302/2007).

To my parents

Abstract

Software has improved quality of life dramatically and has now a vital role in today's society, supporting simple services that range from simple electronic shopping to software responsible for flying planes or performing remote medical surgery. The demand on services is increasing and the result are systems of software that grow in size and complexity. For that reason these software systems are more prone to faults and software quality is an increasing concern for organisations developing software. The costs associated to lack of quality can simply put a software organisation out of business and worst, result in loss of human life.

The need to ensure high levels of software quality motivates organisations to adopt approaches to improve their software development process, also referred to software improvement models or simply improvement models. There are two paradigms to process improvement, the benchmark and the analytical based process improvement approaches. Benchmark based approaches are prescriptive in nature, defining requirements or prescribing a set of practices originating from top performing organisations, that are adopted by organisations aiming to improve their software process. Analytical approaches are based on strategies that aim first, to define business, process and product goals and then establish a clear understating of the impact of process performance in these goals.

A recent trend in software process improvement in the adoption of more than one improvement model into a single organisational environment, originating what are denominated multi-model environments. The goal is to attain the cumulative added benefit of adopted models. Several challenges arise in these multi-model environments that motivate the research work of this dissertation.

One challenge in multi-model environments is the comparison of improvement models for selection and integration purposes and existing approaches compare models in qualitative terms. We propose metrics of size and complexity to compare improvement models in quantitative terms. Additionally, in multi-model environments, ensuring compliance to model adopted is often expected and desired. We develop a model to manage compliance of organisational practices with multiple improvement models minimising the effort required for establishing compliance in these environ-

ments.

In cooperation with CRITICAL Software S.A. a process improvement process is proposed alighted with the analytical paradigm to process improvement and a set of CMMI-Dev level 5 specific goals. Finally we also addressed the issue of modelling complex system of processes that result from adopting multiple improvement models.

The main research method guiding this dissertation was *Design Research*. We followed the steps in the method in different extents. For validation purposes the method expects demonstration and experimental validation. We focussed mainly on demonstration and we lack the desired level of experimentation. Nonetheless we provide detailed demonstrations of proposed solutions. These were submitted and accepted in peer reviewed international conferences.

The main contribution of this dissertation is the demonstration, through practical scenarios, of a set to meteorological approaches to addresses challenges on conducting software process improvement in multi-model environments.

Resumo

O Software melhorou a qualidade de vida de uma forma considerável e assume agora um papel vital no suporte a simples serviços como pagamentos eletrónicos a software que é responsável pelo voo em aviões e por possibilitar cirurgia médica remota. A procura por novos serviços baseados em software está a aumentar e a diversificar-se. O resultado prático é que os sistemas de software estão a aumentar em tamanho e em complexidade. Por esta razão, estes sistemas incorrem num maior risco de exibir falhas e a qualidade do Software é uma procuração crescente nas organizações que desenvolvem software. Falta de qualidade pode simplesmente levar uma empresa à falência ou no pior cenário, resultar em perda de vidas humanas.

A necessidade de assegurar elevados níveis de qualidade no software motiva as organizações a adotar abordagens para melhoria do processo de desenvolvimento de software, também referidas como modelos de melhoria do software ou simplesmente modelos de qualidade. Existem dois paradigmas na melhoria do processo de desenvolvimento de software, uma primeira abordagem baseada em práticas de referência e uma segunda abordagem de base analítica. As abordagens baseadas em práticas de referência assumem um carácter prescritivo definindo um conjunto de requisitos ou práticas, originárias de organizações com processos de desenvolvimento com elevados níveis de desempenho. Os modelos resultantes são adotados pelas organizações que procuram idênticos níveis de desempenho. As abordagens analíticas são alicerçadas em estratégias que visam numa primeira fase definir objetivos de negócio, de processo e de produto e depois perceber, de uma forma clara o impacto das fraquezas da organização na capacidade de esta atingir os objetivos identificados.

Uma vaga recente no domínio da melhoria do processo de software é a adoção de mais do que um modelo de melhoria pela mesma organização, originando os denominados ambientes de melhoria multi-modelo. O objetivo é acumular os benefícios dos modelos adotados. Novos desafios emergem nestes ambientes que motivam o esforço de investigação desta dissertação.

Um desafio nestes ambientes é a comparação de modelos de qualidade para efeitos de seleção e integração. As abordagens existentes permitem comparar os modelos de uma forma qualitativa. No âmbito desta dissertação um dos contributos é uma pro-

posta de métricas de tamanho e de complexidade de forma a permitir uma comparação de base quantitativa. De igual forma, nos ambientes multi-modelo, assegurar o alinhamento das práticas organizacionais com os modelos adotados é na maior parte das vezes um requisito. No âmbito desta dissertação propomos um modelo que permite gerir a informação sobres as práticas organizacionais implementadas e os requisitos/práticas prescritas pelos modelos de melhoria adotados com o objetivo de minimizar o esforço necessário para assegurar o alinhamento de praticas organizacionais e os modelos adotados. Adicionalmente e em parceria com a empresa CRITICAL Software S.A. , é proposto um processo para a melhoria do processo de desenvolvimento de software alinhado com a abordagem analítica de melhoria de processos e com os objetivos específicos da área de processo de nível 5 do CMMI-Dev.

Finalmente, abordamos o problema de modelação de processos de software de elevada complexidade que resultam tipicamente da adoção de vários modelos de qualidade. É proposta uma abordagem de modelação de processos a níveis elevados de abstração que permite o seu refinamento para modelos de mais baixo nível, baseado num conjunto de regras de transição que permite a sua conversão de uma forma sistemática.

A metodologia de investigação adotada nesta dissertação foi *Design Research* e executámos os passos expectáveis da metodologia na extensão possível. No objetivo de validação, o método espera o uso de demonstrações e de experimentação. O nosso principal foco foi a demonstração, não tendo sido possível atingir o nível de experimentação desejável. Porém, o detalhe e extensão das descrições nas demonstrações é elevado e o trabalho foi submetido e aceite em conferências internacionais da área.

A contribuição desta dissertação é a demonstração, através de cenários práticos, de um conjunto de abordagens para endereçar desafios emergentes nas organizações que adotam múltiplos modelos de qualidade na melhoria do processo de software.

Contents

List of Figures

List of Tables

Acronyms

ANOVA Analysis of Variance (ANOVA)

ATAM Architecture Tradeoff Analysis Method

BPMN Business Process Modelling Notation

CAR Causal Analysis and Resolution

CMM Capability Maturity Model

CMMI-Dev Capability Maturity Model Integration for Development

CMMI Capability Maturity Model Integration

DMAIC Define Measure Analyse Improve Control

ESE Experimental Software Engineering

ETVX Entry Criteria, Tasks, Verification and Validation and Exit Criteria

GP Generic Practice

GQM Goal Question Metric

IDEAL Initiating, Diagnosing, Establishing,Acting, Leveraging

IEEE Institute of Electrical and Electronics Engineers

IS Information Systems

ISO International Organisation for Standardisation

OPM Organizational Performance Management

OPP Organizational Process Performance

OSSP Organisational Set of Standard Processes

PA Process Area

PDCA Plan-Do-Check-Act

PML Process Modelling Languages

QIP Quality Improvement Paradigm

QM Quality Management

QMS Quality Management System

QPM Quantitative Project Management

QPPO Quality and Process Performance Objectives

SME Small to Medium Enterprise

SP Specific Practice

SPEM Software and Systems Process Engineering Metamodel Specification

SPI Software Process Improvement

SPL Software Process Lines

UML Unified Modeling Language

Chapter 1

Introduction

Contents

This dissertation addresses a recent trend in the field of the software process improvement: software process improvement in multi-model environments. It contributes to the field of software process improvement by proposing solutions to challenges that arise when organizations adopt several improvement models as a strategy to drive their improvement efforts. This chapter presents the context of research, the challenges addressed, the research method and goals driving this research work.

1.1 Software Process Improvement

Services that rely on software are now pervasive in today's society, these can range from simple electronic shopping services to software responsible for flying air-planes and manage nuclear power plants. It's very difficult to picture our lives without the present of software. It has dramatically increased quality of life.

A fact is that software is becoming more complex as result of the ever growing demand of new functionality to improve effectiveness and efficiency of services. Layers of software systems are being combined and stacked originating complex software systems. An evidence is the increasing number of lines of code per application [Mil07]. However, software development is a challenging endeavour. The ability to deliver software with quality, in time and within budget it extremely difficult to most organizations developing software. Reports of unsuccessful projects and poor quality in delivered software are widely documented [Cha05].

As software system grow larger and complex, defects will be part of the increasing lines of code that bring these systems to work. Software quality, or the lack of it, is an increasing concern as systems are growing in complexity. Many examples of failures exist that resulted from software problems caused by defects that have been overlooked due to ineffective development practices and poor management decisions [Jon96].

Project cancellation and schedule and cost overruns form the greater percentage of how projects finish, leaving a small percentage of successfully completed projects. The Standish Group publishes on a yearly basis report on the investigation of the causes of software project failures. Evidences show that size correlates negatively with project success [Mil07] and software systems are getting bigger and more complex. Improving the way we develop software is thus a key challenge for society. Software is at the core of any modern service or product and ensuring its quality is vital. As a consequence, Software Process Improvement (SPI) is an increasing concern for any software company that wishes to survive [CF02].

SPI efforts are motivated by the quest of improving business performance where SPI initiatives are planned to achieve specific business or operational goals. These often include improvement in customer satisfaction, business profitability, market share, improve product and service quality and reduction in development costs and cycle time [Hal96]. Whatever the driver for improvement, SPI initiatives are challenging undertakings that require investment and time. Investment is related to the introduction of new resources, whether human or technological, required to implement the SPI initiatives. Time is required as improvement implies change in human behaviour.

A software process can be defined as the logical organization of people, materials, energy equipment, and procedures into work activities designed to produce a specified end result [Hum93] [FCB00]. The expected result involves satisfying cost estimates,

meet schedules and achieve the required quality attributes with some consistency. People and technology are integral parts in this process and perform a fundamental role in the resulting product quality. To sustain an acceptable level of consistency the software process must be effective most of the times, meaning there is an end product resulting from the process, this is not always the case with an *ad-hoc* approach, characterised by being highly dependable on external factors like competence and extraordinary efforts by people in the organisation. When this external conjecture fails the product frequently is not delivered. To be effective a process must be planned and executed with accordance to a policy, comprehending a fist level of achievable control [Hum89] . Software organisations need assurance they have the capability and consistency to deliver products that meet customer needs. Risk and surprises with the software development deliveries and maintenance problems are the most recurrent problems referred by unsatisfied customers. These involve reschedule and late deliveries, unconformity with the requirements, delivery with defects and late maintenance response times[PCCW93].

An accepted organizational strategy to achieve some level of control in product quality is to use appropriate processes and methods in software development[Hum93]. Processes and methods are viewed as control levers in product quality. To improve product quality the process must be constantly improved. Organizations begin their process improvement when they find product issues in the field and start investigating how to prevent similar issues to occur in the future.

In this dissertation we will adopt the term *improvement models* os simply *models* to refer to any SPI approach, whether benchmark based or analytical based. In the literature the following terms are also used: quality model, improvement technologies, improvement frameworks or simply quality standards. Preferably, the term improvement models will be used but related terminology can also be found to ensure alignment with referenced material or authors.

There are two paradigms that organizations can adopt to drive their SPI efforts [Car91]. The *benchmark based* approach, or also called *top-down* approach, considers that organizations should adopt a set of best practices to define their software development process. These practices are defined by normative bodies that collect evidence of high performing organizations and thus set the current state of the art in software engineering. Two of the best practices models in the domain of Software Engineering are Capability Maturity Model Integration for Development (CMMI-Dev) [CKS11] and ISO12207 [II08].

The *analytical* approach, or *bottom-up*, implies that before improvement initiatives are initiated a quantitative evidence is used to determine where improvements are needed and also to validate if improvements were successful. Examples of these approaches are Shewhart *Plan-Do-Check-Act* improvement cycle [SD86] and Basili's

Quality Improvement Paradigm [Bas89].

The benchmark based approach success was recognized by Card [Car04] and Conradi [CF02]. Organizations adopt prescribed practices to their own set of organizational practices to drive their improvement efforts. Analytical approaches are considerably less present as a strategy for SPI efforts. The paradigm requires a strong basis on software process measurement which is recognized to be a challenge in Software Engineering [Mun95]. The applicability of the analytical approach motivates one of the research goals for this dissertation.

A recent trend in SPI is the adoption of multiple improvement models originating multi-model environments [SKMM08c]. Market pressure, competitiveness, regulatory compliance or the need to solve a particular issue are general business drivers for organizations to adopt multiple improvement models. A common combination of improvement models is a pair of benchmarked based models, namely, CMMI-Dev and the ISO9001 standard. The goal is to obtain the cumulative added value of each model into a single environment. Several challenges arise in these environments. These will be described next and provide the motivation for the research work for this dissertation.

The need to compare improvement models

Multi-model environments are characterized by the possibility of having several improvement model being implemented concurrently at different hierarchical levels and/or across different organizational functions. Usually, adoption decision lies at different levels of authority in the organisation and is motivated by different needs and perspectives of distinct business units. Also, models are likely to accumulate over the years, being adopted one after the other. These efforts, if not supported and coordinated appropriately carry significant risk of failure [SKMM08c].

Additionally, the integration effort may easily end up in misalignment of improvement models chosen for implementation creating additional reconciliation costs in the final solution. This may result from failing to identify existing relationships between models that can easily lead to operational problems. As a consequence, productivity is impacted negatively resulting in excessive costs and erosion of expected benefits. In multi-model environments the ability to align are compared to SPI solutions .

Finally, the overall picture of capability and cost of quality for each model may be difficult to attain when these are combined into a single environment. Organizations adopting more than one improvement model are challenged to be effective and efficient in using multiple quality models[FS98, Pau08].

To mitigate the risks of failure and inefficiency, organizational process improvement groups need the ability to compare models effectively before these are adopted for implementation. A fact is that comparing models is not straightforward. Firstly, the

number of models is considerably high across multiple domains and disciplines covering a diverse set of subjects. Secondly, comprehensiveness of descriptions and structural differences need to be considered for effective comparison. Several approaches allow to compare improvement model in qualitative terms, providing subjective level of evaluation of models differences and similarities. A quantitative approach is desirable to deliver a more objective evaluation of models differences and similarities. The goal is to improve the understanding of models and support adoption decisions.

The challenge of comparison of improvement models in quantitative terms motivates a research goal addressed in this dissertation.

The need for compliance in multi-model environments

Ensuring compliance with practices adopted is generally a requirement when improvement models are adopted. Compliance involves collecting objective evidence resulting from performed practices that satisfy requirements or best practices defined by improvement models. Specifically, benchmark based improvement models often require that organizations demonstrate compliance, through audits and appraisals, as formal recognition of compliance.

When organizations adopt several improvement models the requirement of compliance also applies. In multi-model environments it may be required to demonstrate compliance of organizational practices with the set of models adopted. A fact is that improvement models often share their scope, meaning they prescribe best practices for the same knowledge domain. The result is that improvement models may overlap and properly managing the information resulting from identifying where models share their scope is an opportunity to minimize efforts to ensure compliance in multi-model environments.

The problem is that improvement models use different structures and terminology when describing their content, raising the challenge of managing such information to properly support compliance with multiple improvement models effectively. To our knowledge research has not addressed this issue of compliance in multi-models environments and our objective is to propose a solution for an information system that can support joint audits or assessments in a single effort.

The need for modelling complex systems of processes

A by-product of adopting multiple improvement models is the considerable size of the resulting system of processes. Benchmark based models are extensive in the number of practices they prescribe. CMMI-Dev v1.2 lists 175 Specific Practices and ISO12207 lists a total of 124 Activities. When adopted by an organization, practices originate complex systems. Complexity results from the high number of processes and their

activities along with interdependencies between these activities. If activities and their dependencies are not documented properly it becomes difficult to grasp the overall work-flow of such systems.

A representation of these systems is needed to ensure they are accurate and useful to an organization. Modelling languages are used to describe process and activity details. The benefits of having a process model are several. Aside providing a representation of activities to be performed in the organization, a process model facilitates process guidance and may support enforcement and/or partially automate the software development process.

Research in process modelling has focussed in developing modelling languages capable of describing low level process descriptions and several successful approaches exist. Describing complex systems with low level languages can easily become infeasible if one wishes to analyse the overall work-flow of the system. A form of abstraction is required for modelling complex systems. Modelling large process systems is a fairly recent research subject and existing approaches do not provide methods for having high level process specifications linked or translated to low level process specifications. This issue of derivation of a process model at higher levels of abstraction and having explicit alignment with low level process models motivates our research work in this topic.

1.2 Synopsis of state of the art and research objectives

Comparison of improvement models is relevant for selection and composition of models when adoption of multiple improvement models is considered. Selection and composition are two steps from an harmonization framework proposed by Siviy to manage multi-model environments [SKMM08c]. Selection occurs prior to the composition effort and represents one important step. At this stage, the most suited model or set of models is expected to be identified. A selection decision may be formed from simply following common industry patterns of models adoption or by having to meet specific regulatory requirements.

Approaches to guide selection and adoption of models may include affinity groups, taxonomies, mappings, selection and implementation patterns and formal decision methods [SKMM08b]. These approaches provide different levels of comprehensiveness for model comparison. Taxonomies enable a high level comparison, generally to compare a considerable set of models. Conversely, mappings are often used to compare two models in a level of detail considerably higher when compared with taxonomies. These approaches provide a comparative and qualitative analysis of the models considered for comparison. In a taxonomy for improvement models proposed by Paulk

[Pau08] a challenge is left open on how to compare improvement models by considering, in the scope of an attribute related to architecture, a quantitative understanding of improvement models content. This challenge motivates our first research goal.

Research Goal 1: How can improvement models be compared in quantitative terms in the presence of architectural structural differences and different levels of detail?

Ensuring compliance of organizational practices with improvement models is a requirement for organizations adopting multiple improvement models. To our best knowledge research as not addressed this issue. We define our second research goals to addressed the issue of managing compliance in environments characterized by having adopted multiple improvement models.

Research Goal 2: How to manage structural and terminology differences of multiple models to support the evaluation of compliance with multiple improvement models in a single organizational environment and in a single iteration?

Maturity Level 5 (Optimizing) is defined by CMMI-Dev as: *'..an organization continually improves its processes based on a quantitative understanding of its business objectives and performance needs.'* At this level there is a clear intersection of the two existing paradigms for software process improvement, benchmarked vs. analytical approaches. It is also argued that at this level the real benefits of the benchmarked based approaches are attained providing the adequate return on investment on SPI efforts. CRITICAL Software is a software house certified with CMMI-Dev Maturity Level 5. The research work described in this dissertation was carried out in a context where CRITICAL Software was defining it's organizational practices in line with CMMI-Dev Level 5 requirements and also preparing and appraisal of Level 5 for the first time. Considering the interest by the research community on how organizations can operate in an optimizing level we defined our third research goal as:

Research Goal 3: To what extent the analytical paradigm to software process improvement can support organizations in achieving CMMI-Dev high maturity requirements?

As stated previously, adopting multiple improvement models may lead to complex systems of processes mainly as result of the considerable number of practices that are expected to be adopted by organizations. Modelling of complex systems of processes is a recent research subject and solutions focus on providing modelling constructs that are used to develop processes at higher levels of abstraction, also called the *design*

phase of process modelling [HK89]. Research in the field of process modelling has focused on the development modelling languages for designing processes at lower levels of abstraction, also called the *implementation phase* of process modelling [HK89].

Developing processes using different levels of abstractions is beneficial when one needs to deal with complexity of large systems but on the other hand raises the challenge of ensuring the adequate transition of the information captured at higher levels and the specifications at lower levels.

Concepts of *Process Landscape*, *Software Process Lines* and *Process Line Architecture* were developed to provide means for abstracting low level process details and focus explicitly on defining logical dependencies of high level entities. At higher levels of abstraction the focus in more on modelling behaviour or orchestration of the system of processes rather than elaborate on the specific details of each process or activity being modelled.

However, existing approaches developed for designing processes do not explicitly describe how high level modelling constructs relate to low level modelling constructs. This link between high and low level process modelling constructs is possible by mapping high level modelling languages and low level modelling languages to ensure adequate design traceability between process specifications. This shortcoming motives our last research goal.

Research Goal 4: Can modelling language constructs used to develop high level process specifications be used to inform the development of low level process descriptions and thus maintain traceability between process modelling specifications?

1.3 Research approach

Research is defined as an activity that contributes to the understanding of a phenomena, typically a natural phenomena [VK07] and research methods are a set of research activities that a research community considers appropriate for the production of understanding. It is generally accepted the existence of two main research methodologies: quantitative and qualitative and these form the majority of research work in the field of Information Systems (IS) [Rec12]. Two additional approaches are emerging in the IS field, the *Mixed Methods* approach and the *Design Science* approach. The *Mixed Methods* approach combines quantitative and qualitative methods for data collection and analysis, which are orchestrated either in sequence or concurrently.

Design Science research is a prominent research method that as emerged as a response to criticisms aimed at either quantitative or qualitative methods [Rec12]. The natural science research methods, in which quantitative and qualitative approaches are

placed, are suited to study naturally existing or emergent phenomena but are alleged insufficient to study the core of the IS field that faces *'wicked organisational problems and challenges'*. Design Science has emerged has a paradigm shift to support research when developing novel, innovative solutions to problems faced by organizations [HC10]. The design science paradigm as its roots in engineering and the *science of the artificial* and has emerged as a problem-solving approach to IS research. Practitioners adopting design research seek to develop new ideas, practices, technical capabilities, and products through which the analysis, design, implementation, and use of information systems can be effectively and efficiently accomplished.

Our research considered Design Research as the main research method. Design science research is defined by Havner [Rec12] as:

"... a research paradigm in which a designer answers questions relevant to human problems via the creation of innovative artefacts, thereby contributing new knowledge to the body of scientific evidence. The designed artefacts are both useful and fundamental in understanding that problem.".

The methodology of a design cycle is defined by [VK07] and depicted in Figure 1.1. The methodology defines a sequence of steps, a set of expectable outputs and the flow of knowledge between the steps.

The Design Cycle begins with *awareness of the problem* where the problem is identified and defined. The next step a *suggestion* for a problem solution is expected to be inferred from existing knowledge or theories, or a tentative solution can be developed using a research methodology. The actual *development* of an artefact follows the suggestion step and is performed iteratively using as starting point a first tentative design for the solution. As the implementation of the artefact is completed, the *evaluation* step is performed according to functional specification implicit or explicit in the suggestion phase. Evaluation is important to determine how well the artefacts works and often requires the use of empirical methods that relate to theory testing (for confirmatory purposes) methods and techniques, namely action research, controlled experiments, simulation, or scenarios.

A Design Science Research methodology for IS research is proposed by [PTRC07] that incorporates principles, practices, and procedures required to carry design research. The goal of the methodology is to have a common acceptable framework to carry out design research in IS field and a mental model for its presentation.

The design research process includes six activities, elaborated in [HC10] and summarized next.

- Activity 1. *Problem identification and motivation*. The research effort begins

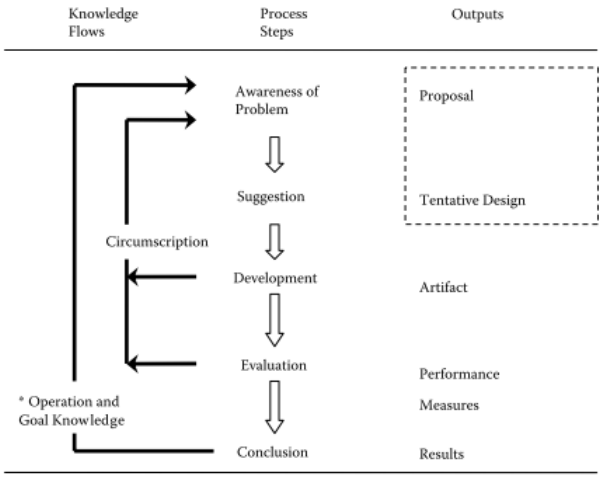

Figure 1.1: The general methodology of design science research (in [Rec12]).

with defining the problem, preferably in its canonical form. The goal is to have the problem described in a common and unique representation to ensure a broad understanding of the elements that are part of the problem domain. Adequate justification for a solution is necessary both to motivate the researcher and the research audience in pursuing the solution and to make clear the researcher reasoning about the research effort. Resources relevant for this activity include knowledge about the state of the problem and the relevance of the solution.

- Activity 2. *Define the objectives for a solution*. In this step the goal is to infer the objective of a solution from the problem definition. Objectives can be quantitative, describing how the desired solution can be better than current ones and/or qualitative, describing how the new artefact can support solutions to problem not addressed before. The resources required for this step are the knowledge of the state of the problem and existing solutions and their efficacy if available.

- Activity 3. *Design and development*. The third step involves creating a potential solution to the problem, an artefact. Examples of artefacts are potentially constructs, models, methods, or instantiations [HMPR04]. The design and development activity aims to define the artefact desired functionality and its architecture and to create the actual artefact. Resources required to perform this activity include knowledge of theory that can lead to the an actual solution to the problem.

- Activity 4. *Demonstration*. The fourth step involves demonstration of using the artefact to solve one or more instances of the problem. Demonstration can be achieved experimentation, simulation, case study, proof, or other support activ-

ity. For the demonstration step knowledge is required of how to use the artefact to solve the problem.

- Activity 5. *Evaluation*. In the fifth step the objective is to observe and measure how well the artefact supports a solution to the problem. Evaluation can be performed by comparing the objectives for the solution with the actual results observed from the use of the artefact in the demonstration. Relevant metrics and analysis techniques are relevant to this evaluation step, where quantitative functional and non-functional measures are desired for an effective comparison. The evaluation can be sustained by empirical evidence or logical proof and at the end of this step researchers may choose to execute a new design and development iteration going back to step three to improve the solution result or may choose to proceed o the next step of communication of results.

- Activity 6. *Communication*. The final step of the research process involves communication. The research is not complete if the results are not made available to the research community and other relevant audiences like practitioners. The communication must be complete by describing the problem and its importance, the details of the artefact developed, its utility and novelty and its effectiveness.

Validation of design research results can focus on the technical aspect of the artefact developed or an evaluation of the social-technical aspect including usefulness and organizational impact[HC10]. When considering the technical performance aspects of the artefact, analytical modelling, simulation, or measurements provide a good base to conduct evaluation. When considering the organizational impact of the artefact one can opt to perform studies using quantitative surveys or qualitative interviews.

Whether a more technical or social-technical evaluation is performed the approach taken to perform such evaluation can be based on *observational case studies* where the artefact is evaluated in a specific business environment or *experimental* methods where the quality attributes of the artefact are analysed in a controlled environment.

When not possible to evaluate the artefact through observation in the environment or in controlled experiments one can opt to follow a *descriptive evaluation* of the artefact. A descriptive evaluation may reside on the informed argument method where claims about the ability of the artefact in providing a solution to the problem are stated and are logically sound in the basis of the knowledge base of the problem domain.

The contributions of these dissertation are supported by instances of the design cycle research methodology. The primary source of validation of our research work considered the following:

- Demonstration and descriptive evaluation - informed argument based on the

knowledge base are used along with the scenarios defined to develop a convincing argument of the utility of the solutions proposed.

- Communication: results were peer reviewed in the context of international scientific conferences that provided an external assessment of the soundness of our proposals. The content of chapter 3, 4 and 5 and 6 was published in peer reviewed international conferences.

1.4 Dissertation outline

This dissertation is structured in 8 chapters. In Chapter 1 the context and motivation for the research work performed in this dissertation is detailed. A set of challenges that organizations face when adopting multiple improvement models is described. In line with these challenges, research goals are defined and the research methodology adopted to guide research efforts is detailed.

In Chapter 2 the general context of software process improvement research is given. An analysis of the existing paradigms applicable to software process improvement is described. A more detailed analysis on the state of the art in software process improvement in multi-model environments is provided.

Chapter 3 provides the first contribution of this dissertation, addressing research goal 1. We start by detailing the specific challenge of comparison of improvement models and then propose an approach to solve the issue of delivering a quantitative understanding of improvement models. We demonstrate the use of the method in delivering a quantitative understanding of several ISO standards when compared with CMMI-Dev.

In Chapter 4 research goal 2 is addressed and a model to manage compliance in multi-model environments is proposed. We start by providing the context of what means establishing compliance between organizational practices and adopted improvement models and describe the specific challenge of ensuring compliance in multi-model environments. We then propose a solution to organize the information relevant to establishing compliance in a multi-model environment scenario. We demonstrate the use of the proposed model in support of a process assessment of an environment using three improvement models.

In Chapter 5 a comparison between analytical approaches to SPI and CMMI-Dev high maturity process areas is detailed and a SPI process is proposed. In Chapter 6 an experiment of improving the software process in an industrial setting is described. The improvement initiative is in line with high maturity requirements set by CMMI-Dev Optimizing staged level. We detail a experiment performed to understand the Inspection process performance and derive an improvement based on sound statistical

analysis of the experiment performed. The case study is described in a perspective of delivering insights on the steps required to have an organization operating at an optimizing level where different paradigms in process improvement are combined to drive process improvement.

In Chapter 7 research goal 4 is addressed and a meta-model to support the definition of process architectures with the goal of modelling complex systems of processes is proposed. Additionally, a method based on a set of transitions rules that allows to refine the architecture specification into low level process descriptions/specification is detailed. The meta-model and the transition rules are applied in a demonstration case of defining a process architecture and a subset of architectural components are refined into low level process specifications.

In Chapter 8 concluding remarks on the research work performed are provided and possible future research work IS presented.

Chapter 2

Software Process Improvement in Multi-Model Environments

Contents

This chapter presents the state-of-the-art in software process improvement. Software process improvement is put into context with the underlining premises of the process paradigm. We then describe and compare the two existing paradigms on software process improvement and describe the most relevant approaches in both paradigms. Recent methodological approaches for software process improvement in multi-model environments are also described.

2.1 The process approach

Software Process Improvement (SPI) is grounded on the concept of process discipline applied to software development. A generally accepted benefit of having a process approach is that it streamlines a groups effort and activities towards achieving a common goal [Zah99]. The expected result is disciplined work that fosters capability by decreasing the uncertainty of process results. The opposite scenario is one that relies on functional areas to operate the business. The major risk to organizational performance is that functional areas can easily become isolated if local goals are not carefully aligned with overall business goals, often resulting in organisational inefficiencies.

The process approach addresses the risk of isolation resulting from vertical functional areas by creating an horizontal, cross-cutting process that gathers functional areas in a single streamlined end-to-end process. A process oriented multi-disciplinary team assumes responsibility of process results and performance goals are more easily aligned with final business objectives.

The shift becomes visible when isolated functional area goals give place to process goals that depend on each of the functional areas participating in the overall process outcome in an integrated approach. Establishing an organisational process happens first by process internalization then institutionalisation. Internalization is when an individual naturally on painlessly follows the process, institutionalisation happens when everyone in the organisation follows the process and mechanism to ensure process adherence are in place.

SPI assumes that establishing a process is a tactical approach to deal with the complexity of managing software development. SPI movement initiated in the 1970s with the introduction of structures methods that focussed on formalization of definition of requirements and traceability of requirements through design and build of software systems, also called the first wave of the process movement [Zah99]. The quality movement initiated during the 1930s was highly influential to the software process movement and most of the concepts introduced by Walter Shewart and Edwards Deeming [Dem00b] and Philip Crosby [Cro79] developed for manufacturing were adopted and firstly introduced in the software industry by Basili [Bas85] and Humphrey [Hum89].

However, there are fundamental differences between developing software and manufacturing. Software development is not production in the sense that software development does not produce the same things over and over again but rather each software product is different from the last. Additionally, technologies are human based, creativity and human ability plays a relevant role in process variation that is not comparable to manufacturing. A corollary is that the process to develop software is necessarily different, as differences in the scope and most likely the product goals will require

different processes to develop each software product.

Aside these differences, the fundamental premise that supports SPI as an approach to improve software development is that product quality depends highly on the quality of the processes used to develop it [PCCW93]. Arguments supporting a process focus to improve software development problems are:

- The quality of a software product is largely determined by the quality of the process used to develop and maintain it, and a good software development process does not always deliver a good product.
- A disciplined process should be able to ensure a quality of both process and product.
- An undisciplined environment means that quality is unpredictable and elusive and when achieved it is unlikely that the reasons that lead to success are clearly identifiable and thus impossible to make sure they are repeatable.
- Manufacturing as shown that discipline on the production line is essential to ensure product quality.

Developing software results from people performing specific tasks with help from development tools [Zah98]. Three entities are involved in this description: *people*, *work instructions* and *development tools* have direct impact on the outcome of software development projects. In this perspective, improving software development implies improving the contribution of people, better work instructions and new and better tools. The focus on ensuring competent people or use new tools seems to be insufficient in eliminating most of software development problems.

Focussing on people with desirable competences may be insufficient as software development most often requires working in a team. Even the most competent people if poorly orchestrated become limited in what they can achieve as a team. The complexity of software development requires that people work efficiently and effective and that goes beyond competences and knowledge of the individuals developing software.

Secondly, tools by themselves seem to provide null or limited benefits when deployed in poorly organized development environments. The corollary is that new technology is not the only factor impacting performance. When not properly deployed it is even likely to increase the chaos - *a fool with a toll is still a foll*. The third key factor is the development process. Sequencing and orchestration of work instructions have a considerable impact on the quality of the software process and is one of the topics of research in software process improvement.

The following section describe how SPI initiatives are implemented, which factors impact SPI initiatives success and how SPI success is evaluated.

2.2 Software process improvement

According to Card [Car91] there are two paradigms to SPI, the analytical or principle based approach (also called bottom-up) and the benchmarking or model based approach (also called top-down). The analytical approach requires that quantitative evidence is used to determine where improvements are needed and also to validate if improvements were successful. Examples of these approaches are Shewhart Plan-Do-Check-Act improvment cycle [SD86] and Basili's Quality Improvement Paradigm [Bas89]. The benchmarking involves identifying an *excellent* organization in a field and documenting its practices and tools. In this approach a less-proficient organization adopts a set of practices and tools of a benchmark organization to achieve excellent performance. Examples of model or benchmark based approaches in the scope of Software Engineering are the Capability Maturity Model (CMM) [PCCW93] and more recently CMMI [CK06] from Software engineering Institute and from ISO, ISO15504 [ISO04a] and ISO 12207 [II08]. These best practices frameworks describe *what* to do without prescribing *how* to do it.

The underlining argument in the top-down approach is the belief that software development, as an engineering discipline, has a set of engineering best practices that are applicable to any organization developing software [TM94]. A argument is analogous to the example of ISO 9001 that defines a set of quality requirements based on a agreed set of engineering and management best practices applicable to a wide set of engineering fields. Typically, organizations adopting a top-down approach have their initial set of practices assessed, by performing a gap analysis to a reference model and begin their improvement efforts by bridging the gaps identified between their organizational practices and practices from the reference model. The top-down proponents acknowledge that every product is different and the process to develop it requires the necessary adjustment to the product specific objectives. For each project the process is defined from an existing common set of organizational process, typically by performing tailoring [Xu07].

The argument of the bottom-up approach is that all software is different and as such, improving the process requires a prior understanding of the software business context before an improvement change is defined. The proponents of the bottom-up approach acknowledge existence of generalized process concepts applicable to software engineering but the improvement initiative shall be driven to address specific needs/problems identified and not to comply to reference best practices that may not be necessarily aligned with organizational improvement needs, thus creating additional inefficiencies [TM94].

Both approaches share the common understanding that no software product is equal and the process to develop it needs to be adjusted to product specific needs. Organiza-

tions adopting the top-down approach address this fact by tailoring the organizational standard set of processes to develop project specific process models. The underline improvement strategy relies on meeting benchmark practices and as result achieve better performance. Organizations adopting the bottom-up approach rely on understanding the local process, products, characteristics and business goals and only then implement a change/improvement program. The strategy is first to develop a detailed insight on the local context and only then define the strategy to initiate an improvement program.

A significant difference between SPI paradigms is that with the benchmark based approach one organization can decide to adopt a best practices model without a detailed understanding of organizational performance shortfalls. A example is when an organization needs to comply with regulatory requirements or to comply with client or market imposed standards. The driver is to meet minimal requirements to be able to bid for contracts. An organization decision to adopt a benchmarked based approach does not require a thorough quantitative understanding of the problems of the organizations but rather an acknowledgement of overall inefficiencies or performance shortfall on developing software.

On the other hand, the analytical approach requires that an organization, that identifies a business performance shortfall or a desire to improve performance goes further into characterizing the environment at a more tactical level. Only then it is expectable that a strategy to solve or improve performance is defined and an improvement project or initiative is initiated. Analytical approaches are somehow related to goals set by CMMI-Dev at Level 4 and level 5 maturity levels. At level 4 a quantitative understanding of process performance is built and continual improvement initiatives are expected at level 5.

The benchmark based approach became popular in the software industry, as revealed by several reports and case studies on the adoption of best practices models [DS97, FO99, Hal96, Hol95, SB97]. Typically these studies report the achievement of improvements on product quality, life cycle time reduction and increased productivity. SPI initiatives based on CMMI and/or CMM, which are based in the concept of process maturity, demonstrate that advancing the levels of maturity reduces the likelihood of high severity defects and therefore decreases the probability of catastrophic or major functional failure. Few studies evaluate the success of benchmark SPI initiatives directly on the the economic benefit of SPI evaluating the amount of money saved by not having to fix defects that are caught as result of SPI efforts [HKS12] [DS97].

Success of benchmarked based approaches was recognized by Card [Car04] and Conradi *et al.* [CF02] however several concerns were subject of discussion. Firstly, benchmarked based approaches, and particularly CMM, were defined based on inferences rather than actual practices from excellent organizations that at the time did not exist. This argument relates specifically to optimizing organizations, or organizations

operating at CMM level 5. Also, maturity is self fulfilling in the sense that an organization may be recognized as mature if it adopts a set of predefined practices and no other evidence that the practices actual helped the organization is required. This issue is addressed in [FO99] where in addition to CMM a strategy based on Critical Success Factor is used to provide the driver for process improvement and CMM is used as an enabler to meet identified success factor goals and is not the end goal itself. Additionally, implementing good practices does not mean that bad practices are eliminated, e.g., good practices may be performed poorly resulting in organizational inefficiencies. SPI based on best practices model approach requires investment, mostly to define and implement new practices, meaning that productivity may decrease if these are not implemented appropriately e.g., with clear goals for cycle time reduction and product quality improvement [DS97]. Organizations adopting benchmarked approaches believe that implementing all practices is required to achieve improved levels of performance.

Another concern related to existing benchmarked based approaches is that they may be inadequate to small to medium organizations also referred as small settings [DD98]. Arguments are that recognized benchmark based approaches are limited in the guidance they provide on how to implements quality models and the benefits expected seem to be tangible only when organizations are of considerable size. Also, high investments required to implement these initiatives, whether by demanding extensive resources and/or knowledge, which typically are scarce in small settings. Case studies on implementing SPI initiatives based on benchmarked based approaches in small settings focus on the need to adapt benchmark based approaches by tailoring and adapting the set of practices described to a minimal set adequate to small settings.

Another perspective on this issue is that benchmarked based approaches and specifically CMMI/CMM are for all sizes but their implementation requires interpretation, tailoring and also good judgement in it's use as a tool for software process improvement. Small organizations may fail to address properly the use of quality models by setting them as goals of improvement rather than measures of improvement. This results when implementations of the model are driven to satisfy the practices/ requirements defined rather than to improve business goals [Pau98].

Evaluation of SPI Initiatives Success

Besides the arguments on limitations of benchmark based approaches the number of reports in achieving success in implementing SPI initiatives are vast in the literature. On the other hand, concerns exist about validity of how SPI initiatives success is studied and reported. A extensive study was conducted by Unterkalmsteiner et.al [UGI+11] to understand how SPI initiatives success is determined. The study, based on a systematic

literature review, identified and characterized different approaches used to evaluate the success of SPI initiatives.

The study highlights that documented SPI initiatives fail to provide a thorough and complete description of the context where the SPI initiative is performed. This fact makes it difficult to replicate the studies to achieve similar results. It is of major importance that description of the process change and it's environment is complete in order to provide the confidence that the study can be replicated. The rule is that the studies fail to provide such information.

Validity of the findings is also questionable by several reasons. The most common validation strategy identified is the *Pre-Post Comparison* when confounding factors are not properly considered or evaluated. Once again the findings are not supported by a detailed description of the context making the interpretation of the results questionable in the sense that factors not discussed may be linked to the result. Less common validation include *Statistical Analysis* and *Statistical Process Control*, *Surveys and Cost-Benefit Analysis*. Evaluation of SPI initiatives is also limited by the presence of measurement definition inconsistencies that limits the communication and comparison do SPI initiatives results. Software measurement discipline does not yet provide the adequate support in definition, selection and validation of measures and thus how an organization chooses to measure specific process or product attributes may not be subject of consensus. Measurement scope is also an issues in the sense that SPI studies are focused on process and product quality, meaning that SPI studies emphasis is at project level rather than organizational level. Perspectives at the organizational level are desirable to validate the long term effects of SPI initiatives.

Key Factors of SPI Initiatives Success

SPI is a challenging undertaking and acknowledging the key factors to SPI success is valuable information for organizations pursuing SPI initiatives. A study performed by Dyba [Dyb05] empirically tested a model that expresses the factors for SPI success. Using a survey sent to a random sample of 154 software and quality managers in the Norwegian IT industry, the study evaluated the hypothesis that each factor was predictive of SPI success. The model included six facilitating factors of SPI: business orientation, involved leadership, employee participation, concern for measurement, exploitation of existing knowledge, and exploration of new knowledge. Success of SPI was measured using the feedback from managers according the level of perceived SPI success and the performance of their organization with respect to cost reduction, cycle time reduction, and customer satisfaction. The study conclusion is that that each of the factors is positively and strongly correlated with the success of SPI. Specifically about measurement, the study concludes that *"it is important that measurement systems are*

designed by software developers for learning, rather than by management for control".

2.2.1 Benchmark based SPI

"Maturity levels should be measures of improvement, not goals of improvement. That is why we emphasize the need to tie improvement to business objectives" [Pau98].

Organisations have at their disposal a long list of quality models, standards, best practices, regulatory policies and other types of practices that aim to provide the necessary guidelines to help organizations guide these efforts. Each model provides unique features to address specific problems and models vary in scope and domain of applicability. These models define specific sets of practices or requirements that when implemented provide a degree of confidence to the organization adopting the model that quality of developed software and performance will improve.

National and international bodies propose and support frameworks and process models to support SPI initiatives, examples are ISO9001 [IEC00], CMMI-Dev [CKS11] and ISO/IEC 15504 [ISO04a] among others. Motivations that drive organisations to adopt the standards vary and can range from business contractors demanding compliance with certain standards, often required to address global markets and self initiative to increase process capability / performance usually driven by organisational goals of customer satisfaction and productivity.

This section presents an overview of popular standards and models used in SPI initiatives. In an improvement program different types of models may be involved. An improvement initiative typically requires managerial efforts to guide process improvement, a model to guide the process assessment and the a use of a process reference model targeted for improvement. This section provides details on process reference models which define best practices for software processes. The best practices approach focus on defining what an organizations should do and does not provide any details on how practices are to be implemented..

ISO15504

ISO/IEC 15504 [ISO04a] also known as SPICE (Software Process Improvement and Capability dEtermination) may be used as a framework for the assessment of processes and as a process reference model. The two contexts set for process assessment are process improvement and process capability determination. The purpose is to provide a structured approach for the assessment of processes within the organizations or for a third party organization with the objective to understand the state and suitability of the processes under assessment.

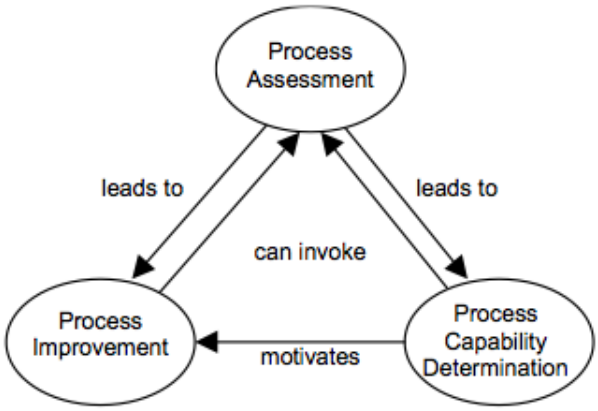

Figure 2.1: Process assessment relationships in SPICE (in [ISO04a])

The standard aims to provide a method for the organizations to improve product quality through the use of proven, consistent and reliable methods for assessing the state of his processes and use the results as input for a process improvement program. Organizations expect to become a capable organization, improving responsiveness to customer and market requirements, minimizing life-cycle costs of their products and maximize end-user satisfaction.

ISO 12207

The standard ISO/IEC 12207 / IEEE 12207 [II08] establishes a common framework for software life cycle processes. Additionally, provides definition of base terminology to be used by the software industry. The purpose is to provide a defined set of processes and concepts that facilitate communication among contractors, suppliers and other stakeholders in the life cycle of a software product.

It can be used in one or more of the following scenarios: by an organization to help establish an environment of desirable processes; by a project to help select, structure and employ the elements of an established set of life cycle processes to provide products and services; by an contractor or a supplier to help develop an agreement concerning processes and activities and by organizations and assessors to perform assessments that may be used to support organizational process improvement. The standard does not detail the life cycle processes in terms of methods or procedures required to meet the requirements and outcomes of a process. This standard supplies a process reference model that supports process capability assessment in accordance with ISO/IEC 15504-2 - Process assessment.

CMMI for Development

CMMI proposes a model that integrates various disciplines that characterize the business process, consisting in best practices for development and maintenance activities applied to product and services. The premise for the CMMI model is that *quality of a system or product is highly influenced by the quality of the process used to develop and maintain it*, and the focus is on improvement the processes in the organization. The latest version is CMMI for Development [CKS11] and the scope of the model is to cover development and maintenance activities in industries including aerospace, banking, computer hardware, software, defence, among others.

CMMI uses levels to describe the evolutionary path that an organization can achieve in the processes used to develop and maintain products and services. The notion of improvement paths allows an organization to choose to improve specific process areas or, in an integrated approach, improving across multiple predefined process areas. The former is called the *continuous* representation and is used in organizations where a set of chosen process are relevant for the organizational business goals, the later is called the *staged* representation and is concerned with the overall maturity of the organization.

Representations define the approach for process improvement allowing organizations the flexibility to plan the evolution from *ad-hoc* and immature to disciplined and mature processes. Depending on the improvement path chosen, levels can be named capability or maturity. Capability applies to a specific process area chosen for improvement. Maturity level relates to a set of predefined process areas that must be implemented to achieve a specific level of maturity.

Level	*Continuous Representation Capability Levels*	*Staged Representation Maturity Levels*
Level 0	Incomplete	
Level 1	Performed	Initial
Level 2	Managed	Managed
Level 3	Defined	Defined
Level 4		Quantitatively Managed
Level 5		Optimizing

Figure 2.2: Comparison of capability and maturity levels (in [CKS11])

CMMI is structured by Process Areas (PA), each describes goal and practices to be achieved and implemented. PAs are arranged in a staged representation in levels, for each level (1 to 5) a predefined set of PAs must be implemented to achieve the respec-

tive level certification. In the continuous representation PAs area arranged according to four thematic categories: Process Management, Project Management, Engineering and Support.

CMMI certifications result from appraisals activities, three class types of assessment exist comprehending different levels of formality. Class A is the most formal appraisal and is conducted by an official lead assessor, known as the Standard CMMI Assessment Method for Process Improvement (SCAMPI). The class B and C are less formal and are used for assessing process capabilities and do not produce capability or maturity rating.

ISO 9001

ISO 9001 promotes the adoption of a process approach to the design and implementation of an organization's quality management system. It defines the requirements to be meet by an overall system driven to manage quality. ISO9001 is not a product driven quality standard, quality control standard or a quality assurance standard. It's purpose is to provide the ability to consistently provide products that meets customer needs and applicable regulatory requirements and enhance costumer satisfaction through the effective application of the resulting system. It is considered to be a generic in the sense that is applicable to all organizations regardless of type, size and product provided and is not a standard specific for software engineering best practices, it focus mainly in improving processes continuously.

One of the requirements for being compliant with ISO9001 is the implementation of a process for measurement, analysis and improvement, to establish conformity of the product and continuously improve the performance the effectiveness of the quality management system.

BOOTSTRAP

Kuvaja [KPB99] introduced an assessment approach denominated *TAPESTRY* that consists of a two day workshop where a self-assessment model named *BootCheck* is applied under the guidance of software improvement experts. This self-assessment consists of a computer-based tool that contains a classification schema, an assessment model, and an assessment method. The schema is used for classifying the assessment results for benchmarking purposes. The assessment model contains a process model (Figure2.3), capability levels, scoring, rating and results presentation principles. The process model is a downscaled version of the BOOTSTRAP [SKK94] process model which is aligned with the ISO 15504 requirements. This assessment model is composed by 19 out of 35 processes from BOOTSTRAP considered to be the most important processes for any Small to Medium Enterprise (SME).

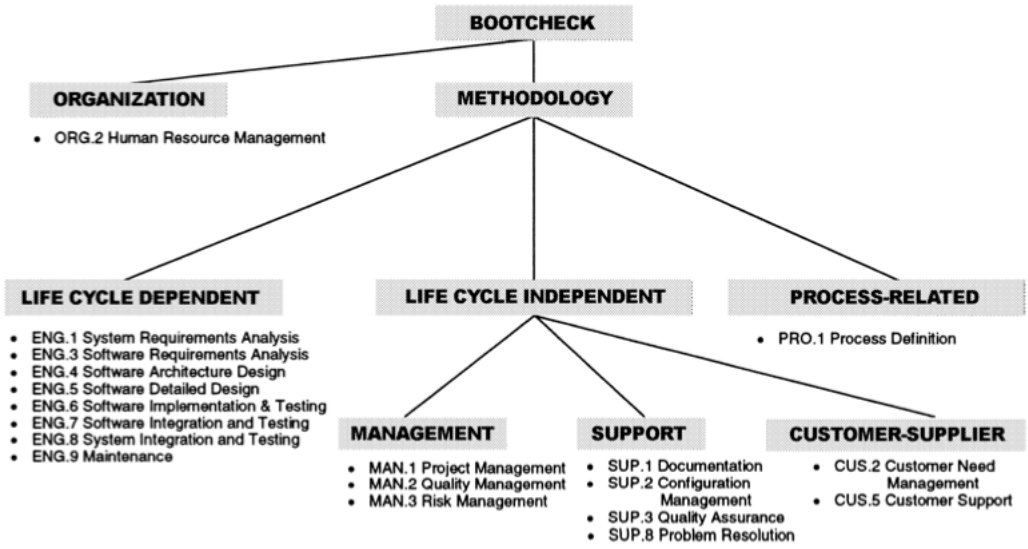

Figure 2.3: BootCheck process model (in [KPB99])

2.2.2 Analytical based SPI

Analytical approaches are based on establishing a improvement cycle based on an understanding of the organizational weakness with respect to improvement or business goals [Bas85].The fundamental difference to benchmarked based approaches is that the driver for improvement are weaknesses identified on process performance and product quality as in the benchmarked based approach the main driver is closing gaps to external process models. Analytical approaches are based on strategies that, first define business, process and product goals and then establish a clear understating of the impact of current process performance on these goals.

Analytical approaches relate to the scientific method in the sense they seek to have an understating / model of the phenomena under study, e.g, establishing the relation between the software process and the process outcome relying on quantitative information of a controlled execution of the process. Analytical approaches are based on the belief that software development is inherently an evolutionary and experimental discipline, mainly as result of underlining characteristics of software development, namely: software development is not mass production and every product is different, technologies used are human based and depend greatly on human creativity thus a great source of variability is present in software development. As result of this variability it is hard to develop models that allow reason about the relation between process and product [Bas93].

In the remaining of this section a set of analytical approaches used for SPI initiatives will be described and later compared in terms of their purpose and applicability.

PDCA

The Plan-Do-Act-Check cycle is a continuous improvement strategy developed by Shewhart in 1939 [SD86] and popularized by Demming [Dem00b] in the 80s. The approach originated in manufacturing with the objective of improving a production line/process model over subsequent PDCA cycles, where improvement effects on product are cumulative over time. The PDCA cycle is part of the 14 principles for transformation of the western management that were defined by Demming with the goal to halt the decline of western industry. The principles were based on Japan's top management since the fifties.The approach begins with the step *Plan* where the current state of the production line is studied, potential improvements are identified and new methods are defined for implementation. Along with the new methods, quantitative quality criteria is set as target for improvement. In *Do*, the improvement plan is performed, ideally in small scale to validate the improvement. In *Check* the effects of the changes are observed and compared with the quality criteria defined in the plan step to check if the improvement was successful. The last step, *Act*, the results of the improvement initiative are studied to learn on the changes and to decide whether the changes are to be implemented/institutionalised in the production line or in case of failure, the root cause is identified and corrective are feed into a new improvement cycle.

In 1993 Deming [Dem00a] modified the Shewhart cycle and called it the *Shewhart cycle for learning and improvement*- the **PDSA** cycle. He described it as a flow diagram for learning, and for improvement of a product or of a process. The PDSA is illustrated in Figure 2.4.

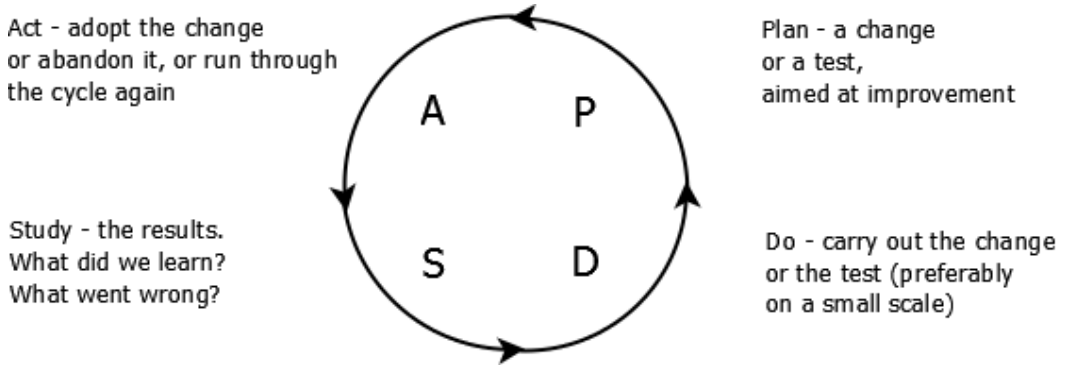

Figure 2.4: PDSA improvement cycle (in [Dem00a])

Quality Improvement Paradigm and Experience Factory

The Quality Improvement Paradigm (QIP) is a six step approach for an organization to evaluate and improve the methodology or process for developing software [Bas85].

QIP is based on the scientific method and considers that software engineering is a an evolutionary and experiential discipline that requires controlled experimentation of techniques used in software development to better understand their limitations and to improve them.

The QIP approach includes the steps outlined in Figure 2.5 [BR88] and is applicable to software projects that develop software. The first step involves *Characterise* the project and the environment with respect to applicable models and metrics. In *Set Quantifiable Goals*, goals are identified that define and drive project and organizational success. In *Choose* the appropriate methods and tools for the project that maximize the chance of meeting the specific project and organizational goals are chosen, often from a set of organizational set of processes or methods. *Execute* the process, develop the product and collect and validate the collected data and analyse it to provide real time feedback to the project. *Analyse* the data to evaluate the current practices, determine problems and make recommendations for future project improvements. Finally *Package*, involves summarizing the experience in the form structured knowledge gained from the current project and store it in an experience base to be reused ion future projects.

In QIP, each software development project is used to experiment the applicability of methods and techniques for software development. A combination of project characteristics and environment along with the project results are used as experimental knowledge to improve future projects. Citing Basili et.al. on QIP [Bas85]: *The Quality Improvement Paradigm is based upon the notion that improving the software process and product requires the continual accumulation of evaluated experiences (learning) in a form that can be effectively understood and modified (experience models) into a repository of integrated experience models (experience base) that can be accessed and modified to meet the needs of the current project (reuse).*.

To fully have a SPI initiative based on the QIP approach logical and/or physical separation of the project development organization from the systematic learning and packaging of reusable experiences is required. This separation gives origin to the concept of *Experience Factory* [BCR02b, BCR02a] that is responsible for providing adequate support to project development by analysing and synthesizing all kinds of experience, acting as a repository for such experience and supplying that experience to various projects on demand business needs. Figure 2.6 details how the *Project Organization* relates to the *Experience Factory*.

IDEAL

IDEAL [McF96] is a software process improvement program model with the purpose of describing the generic sequence of steps recommended for organizations initiating

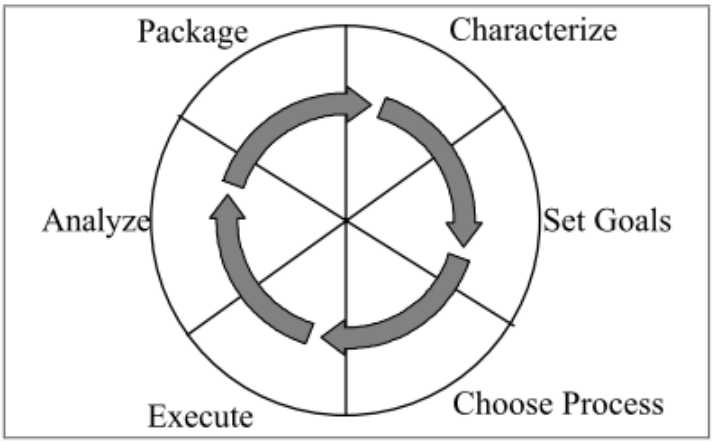

Figure 2.5: Quality Improvement Paradigm improvement cycle (in [BCR02a])

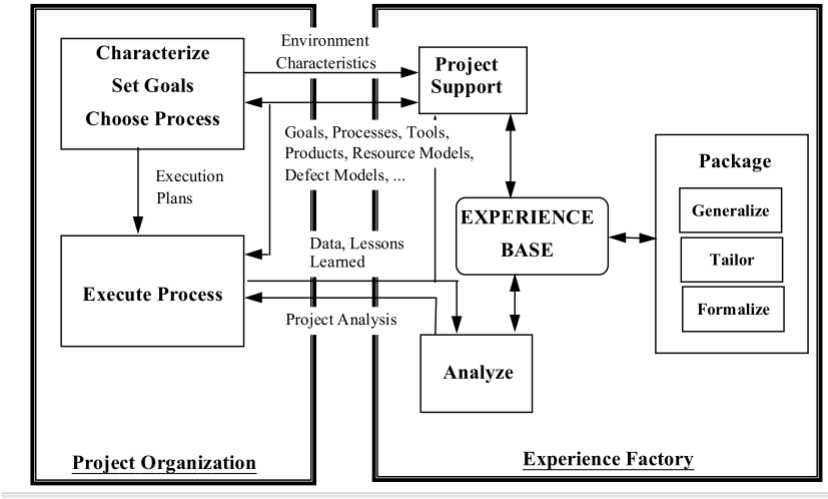

Figure 2.6: Experience Factory (in [BCR02a])

a SPI program. It is composed by five phases: *Initiating*, *Diagnosing*, *Establishing*, *Acting* and *Learning*. The Initiating phase focus on establish of an improvement infrastructure, roles and responsibilities are defined and a plan is created to guide the organizational effort. The generic goals for the SPI initiative are defined. Two key groups are established, a management steering group and a software engineering process group.

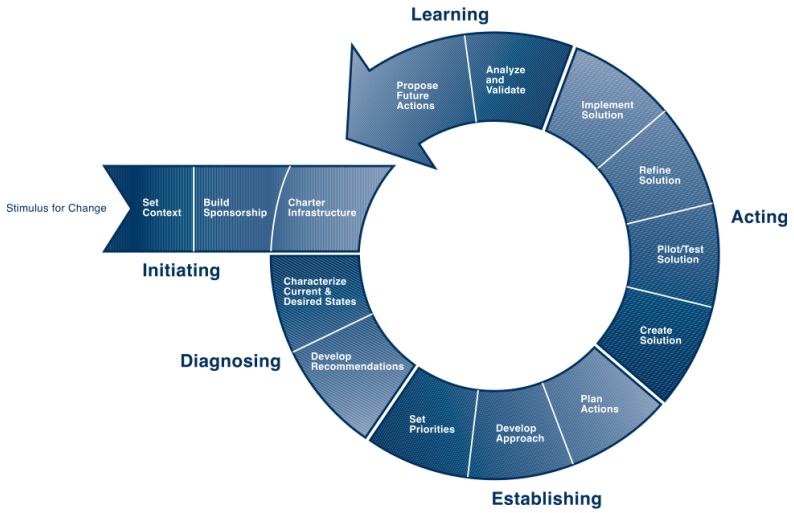

Figure 2.7: IDEAL process improvement program model (in [McF96])

In the *Diagnosing* phase appraisal activities are performed to establish a baseline of the organization current state. The results and recommendations from appraisals and any other baselining activities will be reconciled with existing and/or planned improvement efforts for inclusion into the SPI action plan. In the *Establishing* phase an action plan is drafted based on prioritized improvement activities that will address organizational issues previously identified. Measurable goals are derived from the general goals defined in the *Initiating* phase providing action specific . In the *Acting* phase solutions are developed to address process problems. Successful process improvements are expanded to entire organization. In the *Learning* phase the next cycle is prepared through the IDEAL model, lessons learned are applied to refine the SPI process.

When applying the IDEAL model two components for a process improvement activity should be considered. A strategic component and a tactical component. The strategic component, based on the organizations business needs and drivers, will provide guidance and prioritization of the tactical activities.

DMAIC - Six Sigma

Six Sigma is a continuous process improvement approach originated in Motorola [Pyz03]. The term six sigma is based on the measurement standard that variation in a product

attribute showing a three sigma deviation from the mean is the point when the process requires correction and/or improvement. This standard measure was set empirically in the 20's at Bell Labs and is the base concept for today's Statistical Process Control. In the late 70s at Motorola a 50% design margin was added on top of the three sigma for all the key processes specifications, which lead to a six sigma level of capability in their key processes. Sigma stands for standard deviation and is used as a measure of variation, controlling the amount of variation on the output of a process is essential to a reduction of the cost and improvement in quality which directly impacts customer satisfaction. As with the PDCA approach, Six Sigma was developed in the context of manufacturing industries where a production line or process models is executed repetitively to produce the same product over and over again.

Six sigma is based on four principle components, the first one is the framework Define Measure Analyse Improve and Control (DMAIC), a fact based problem solving process which consists of steps Define, Measure, Analyse, Improve and Control with tool-gate reviews between all phases. Several variations on DMAIC exist, the Design for Six Sigma (DFSS) variance Design Measure Analyse Design and Verify (DMADV) with the phases Define, Measure, Analyse followed by Design and Verify is used when one wishes to create a process instead of improving an existing one.

The second component is a suite of tools that are used in the DMAIC process, these include cause and effect diagrams, check-sheets, stratifying graphs, histograms, Pareto charts, scatter diagrams, statistical process controls charts and tools with strong statistical basis and strong customer focus. The third component is the methodology for measuring process capability, with processes operating at six sigma the level of defects expected are 3.4 defects per million defect opportunities. This measure is used as a comparing measure within process that address the same measured defect. The fourth component is a top down organizational infrastructure that includes training and certifying the problem solving team practitioners as green-belts, black-belts and master black-belts.

Studies of applicability of six sigma concepts to software engineering is increasing in software engineering literature [Bie04]. Six sigma use in software focus on monitoring and controlling with the objective of reducing process variation.

Total Quality Management

Total Quality Management (TQM) is probably the aggregating system perspective on quality management to which all presented analytical approaches relate to. TQM encapsulates and delivers a philosophical view along with a set of principles for organizations to achieve quality products [Flo93].

Flood summarized the definition of quality based on the various perspectives of

quality by Demming [Dem00b], Juran [JG99], Crosby [Cro79], Feigenbaun [Fei83], and the British Standard Definition as: *Quality means meeting customer's agreed requirements, formal and informal, at lowest cost, first time every time*.

TQM is a philosophical or systemic approach to quality (*Total*) in the sense that all persons within the organizations must be aware, involved and acting in quality improvement. The concept of (*Quality*) is inherently linked to customer satisfaction and is achieved by meeting agreed customer requirements, whether these are formal or informal at the lowest cost possible, with no loss or waste in resources. Additionally, according to agreed requirements, the organizations will not accept products bellow customer expectations, it must strive to produce product meeting product specifications at first delivery, every delivery. Along with this view of quality, TQM also includes the following principles:

- A focus on preventions of problems rather than an acceptance to cope with then in a fire-fighting manner.
- Quality improvement must involve management.
- Everybody must be involved, from all levels across all functions.
- Quality improvement must involve management.
- Every job must add value.
- There must be an emphasis on measurement to help to assess and to meet requirements and objectives.
- A culture of continuous improvement must be established (steady improvements and/or dramatic leaps forward)
- An emphasis in promoting creativity.

2.3 Multi-model environments

A multi-model environment is created when an organization chooses to adopt several improvement models as a strategy to guide their SPI efforts [SMM08]. The decision to adopt several improvement models is typically motivated by:

- The need to address challenges of different nature when organisations aim to improve performance.

- The need to comply with quality requirements set by clients or markets. Suppliers for the Aerospace or Aviation segment typically are required to comply with standards applicable to the space segment e.g, European Cooperation for Space Standardization (ECSS) standards or in the Aviation market may requires compliance with Software Considerations in Airborne Systems and Equipment Certification, DO-178B standard.

- Standards imposed by stature or regulations [Pau08].

The expected benefit of adopting more than one improvement model is the cumulative added value of the models adopted. When organisations adopt a multiple model strategy to process improvement their goal is to leverage the best practices made available by different models to better address improvement challenges. The value proposition of using multiple standards is the incorporation of their best practices in a single solution, that otherwise would not be possible to obtain by a single standard approach [SMM08].

Improvement frameworks vary considerably in their scope of applicability. Some are disciple oriented and other are market /vertical oriented and a great number of standards and best practices models exist in the software domain, being maintained by a considerable number of organizations [Moo99][She01]. With an increase of the number of standards, their scope of applicability often overlaps and different levels of detail are used to define models goals. This context raises two challenges for organizations adopting multiple standards: the first is selection of the most adequate standards and the second is how to integrate or combine standards into a single environment [SK08b].

Selection and integration relates to finding an approach on how to accommodate multiple standards into a single environment, managing their differences and similarities to obtain the cumulative added value of adopted standards.

A possible example of scope overlapping and differences in the level of detail is CMMI-Dev and 828-2012 - IEEE Standard for Configuration Management in Systems and Software Engineering [IEE12] standard. In CMMI-Dev a process area of Configuration Management is defined with a set goals to be met by an organization. Institute of Electrical and Electronics Engineers (IEEE) also defines a set of requirements for Configuration Management practices for systems and software engineering. CMMI-Dev defines 3 goals and 9 specific practices in about 11 pages detailing what is expected to be performed with respect to Configuration Management practices and IEEE devotes 58 pages with information that includes definitions, activities, and examples for Configuration Management practices implementation. An initial analysis just driven by the scope of the standard may suggest that 828 provides more detail on how to imple-

ment the Configuration Management process are thus can be used with CMMI-Dev as guidance to implement configuration management practices in the organization.

Siviy [SK08b] argues that one of the issues with multi model environments is the competition between standards when several initiatives are concurrently implemented at difference hierarchical levels in the organization. This leads to competition over resources by the different SPI initiatives to meet their specific needs. The overlapping efforts become costly and the benefices of each technological approach are undermined by the conflicting approaches. Additionally, standards are *stove-piped* along the years of improvement efforts motivated by different improvement needs. Lack of coordination in the integration of existing and new approaches leads to an unplanned multi-model environments. This results in lack of effectiveness of the overall SPI landscape solution, where redundancies and unrealized synergies between standards become apparent.

The current state of the art in multi-model environments is best described using an conceptual framework that defines the steps required to guide improvement efforts in multi-model environments. The framework defined by [SK08b] provides a comprehensive description of the steps that guide the development and implementations of improvement initiatives in multi-model environments. The reasoning framework guides the development of an appropriate solution to meet organizational objectives, by understanding and leverage the use of quality standards of interest, as well as composing these standards in a process architecture to create an harmonized solution.

The reasoning framework is composed by several elements that include a work flow of four steps in sequence, a set of questions to be addressed and principles to use in the process of alignment of the layers and levels of an integrated business (see Figure 2.8). The first step involves the alignment of organizational and improvement objectives and identification of candidate standards. The relevance of this step is based in the argument that business drivers should govern the selection of each standard which should be select based upon their ability to provide the necessary features and capabilities to achieve organization mission and operational objectives.

The second step goal is a strategic selection of standards to meet organizational objectives. In this step a categorization of standards is developed to facilitate and guide the selection of standards, which are best suited to accomplish the objectives defined in the first step. The categorization involves the identification and definition of high level relationships between standards to identify possible gaps, overlaps and synergies.

The design of the improvement multi-model solution is carried out in the third step that splits in two sub-steps, namely, standards composition and development of a process architecture to accommodate adopted standards. The composition of multiple standards requires a understanding of the standards involved and how they connect

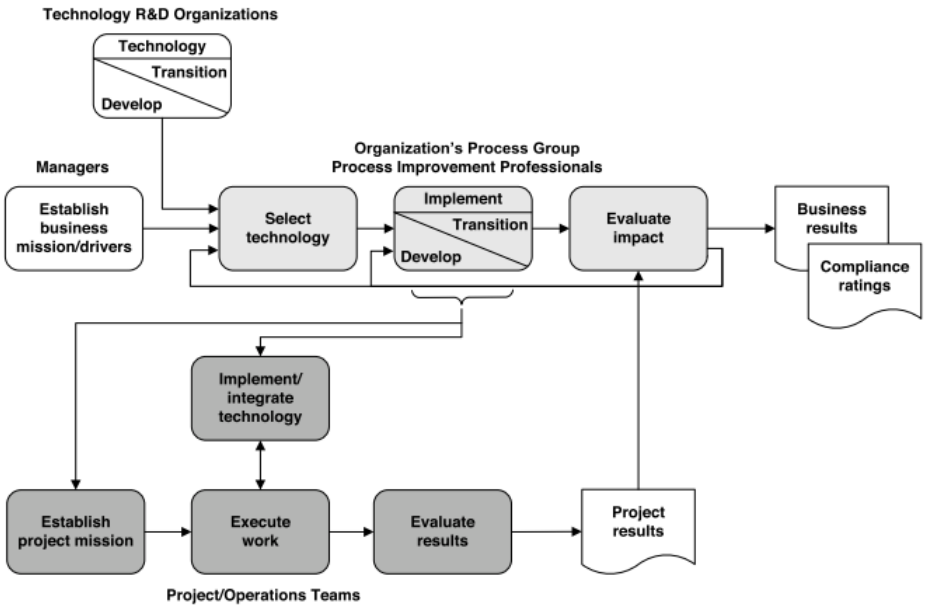

Figure 2.8: High-level process harmonization (in [SK08b])

to each other. This often involves identifying similarities and differences or enabling features between them. When identified, these connections inform the development or update of a process architecture. Developing a process architecture refers to the activity of developing the actual organizational processes (solution) from the standards chosen in the previous steps.

The last step focus on implementing the multi-model process improvement solution and measuring results. In this step the elements of the organizational change management assume a centre role in the design of the deployment activities being carried out. The activities involve the establishment of a process improvement infrastructure that coordinates roles and responsibilities of the champions for different standards.

In summary, the reasoning framework defines four steps. The first step focus in mission translation, the second step deals with standards selection by developing a categorization of standards. The third step focus on standards composition and multi-model solution implementation and the last with deployment and evaluation of impact of improvement solution.

From the steps defined in the reasoning framework, the ones that involve alignment of organizational and improvement objectives, solution deployment and measurement of results are not unique in multi-model scenarios and apply also to single model environments. On the other hand, standards selection and composition and multi-model solution implementation are tightly connected to multi-model SPI and are considered as new research topics [SPS08]. The following sections detail existing approaches to

the specific challenges of multi-model environments.

Standards Categorisation

In the topic of standards categorization (step two of the harmonization framework) the objective is to identify a set of attributes of standards that can serve as base for standards comparison. The decision to adopt multiple standards is not straightforward. Several issues are raised when multiple standards are considered for adoption, namely:

- Differences in terminology - standards use different terms to refer to similar concepts, specifically from different normative bodies [HC01].
- Differences in structure and level of detail - standards use different structure to organize their content and can also differ in the level of detail used to define their best practices or quality requirements [Tin96].
- Standards address similar organizational needs - different improvement models address similar organizational needs and the scope of their best practices is also addressed by other improvement models.
- Improvement models adoption sequence - the order by which the standards are adopted / implemented may influence the decision to adopt a standard [SMM08].

A multi-model environment requires considerable knowledge of the improvement models available and even deeper knowledge of the models chosen for a simultaneous implementation. Categorization implies that models are compared using some strategy to describe their differences and similarities. Approaches that support the adoption decision process by providing some level of categorization are: *taxonomies*, *affinity grouping of models and standards*, *selection and implementation patterns of standards*.

Four classes of comparison methods are identified by [HC01] for comparing SPI improvement models. These methods are applicable in terms of the level of detail desired for the comparison that can range from high to low level comparisons. Low level comparisons require a more detailed analysis of models contents.

- **Comparative characteristics** - relevant characteristics are used to develop a high level understanding of standards by describing key attributes of the standards. The typical use of characteristics is to have several standards compared by providing information for each characteristic. Typically, the characterization is not detailed as often requires to provide yes/no or a relative short descriptions about the characteristic.

- **Model mapping by comparing kernel concepts** - mappings focus on creating a map from statements or concepts of one standard to statements on another standard. The mapping is typically weighed using a mapping function that classifies the mapping association, e.g., strong, week or none. The standards mapping is useful when organizations adopt two or more standards to identify association between standards adopted. Examples of models mappings are available in [PBPV09, MS03] and will be discussed in detail in Chapter 3

- **Bilateral comparison using textual descriptions**: Bilateral comparisons can be created having two standards and taking a point of view of a standard and describe the second standard in terms of the first. Typically this is created using a textual description.

- **Mapping based in needs *versus* standards properties** - this approach considers organizations and environmental needs that must be meet by standards. This approach does not compare standards directly and requirements set by organization needs can greatly limit the choice of standards.

Halvorsen and Conradi [HC01] address the problem of comparison and selection of improvement models by proposing a taxonomy to characterize improvement models (note: the author uses framework instated of quality model). A taxonomy is defined to objectively compare SPI standards and to facilitate information gathering. The taxonomy aims to used as a tool by organizations in the decision process about which standard to use. The motivation for developing a taxonomy is based in the argument that SPI standards are adopted without objective evidence of appropriateness and that standards are difficult to compare due to of their comprehensive nature and diverse scope. Another argument is that terms about software quality and software process are not consistent across standards requiring an individual interpretation regarding these concepts.

The taxonomy developed uses the comparative characteristics method and comprises a 25 characteristics grouped in five categories. It is influenced by previous work on the topic and tries to capture generally aspects regarded as relevant when comparing standards.

The taxonomy does not aim to provide enough information to enable a sound decision in standards selection. It aims to facilitate a high level comparison required to deal with the diversity and comprehensiveness of existing standards. A quick overview to choose the relevant frameworks to be further investigated is the value proposition of the taxonomy. Strong points regarding the framework attributes refer to it's appropriateness by considering commonly used characteristics to compare SPI standards and

being compact in proving a quick analysis to knowledge limiting analysis to a high level.

Paulk [Pau08] elaborates on the work of [HC01] and proposes a taxonomy with 3 major categories that comprise a total of 10 attributes which elaborate on specific topics. The taxonomy goals is to support understanding and comparing diverse SPI frameworks and drive a discussion for better understanding the underlying concepts and architecture of improvement frameworks. The categories proposed and the attributes for each category are:

- **The User Community for the Framework**: Audience, Work Orientation, Support Infrastructure, Drivers for adoption.
- **Framework Architecture and Structure**: includes the attributes of Rating Scale, Architecture and Content.
- **Infrastructure**: includes Diagnostic Method, Improvement Method, Certification Scheme.

The taxonomy is open to evolution and provides the motivation for the work described in Chapter 3.

Similar work on taxonomies is proposed by Heston and Phifer [HP09] introducing the the concept of *process DNA* and *Quality Genes* that summarize concepts or quality characteristics from several key industry standards. Classification is based in an analysis of each model contents to gauge its depth and coverage of each quality gene. Each model is classified for each quality gene as *high correlation*, *some correlation* or *no correlation*. Organisations may interpret quality genes as needs and assess which models are more suitable to meet their process improvement objectives.

Moore [Moo99] proposes a conceptual model in the form of an architecture to help the integration of SPI frameworks in the specific context of Software Engineering frameworks from the IEEE Computer Society. The motivation was to provide the software engineering community an integrated collection of standards that could be applied in unison from which users could select appropriate standards. The architecture comprises three elements that define the organisation criteria, namely:

- Normative levels organizes the standards in a sequence of six layers that range from more general non-prescriptive to more specific prescriptive standards.
- Object of software engineering organizes the relevant entities in software engineering: project, process, customer, resource and product in an abstract model and suggest the categorisation of existing standards according these entities.

- Relationships to other disciplines - exists to ensure that relevant disciplines to software engineering are identified. Examples are standards that address general disciplines of systems engineering, quality management and project management and cross-cutting disciplines such as security, dependability, safety.

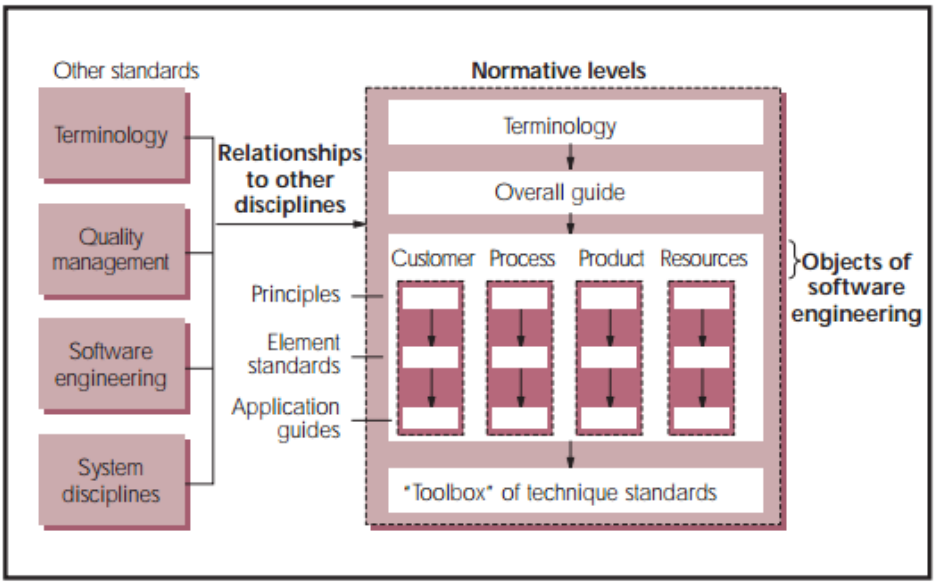

Figure 2.9: Architecture for standards integration (in [Moo99])

A different approach in the topic of standards selection is presented by Siviy *et al.* [SF04] by using an affinity matrix that groups models by their strategic value and application focus (see Figure 3.12). Additionally, it indicates typical decision authority and can be used for pattern selection analysis. The matrix may be used in different goals in mind, it can be used in to link mission translation to its strategy or when combined with descriptions of model relationships can be used to develop the multi-model strategy.

The matrix presents two axis that characterize models according a strategic classification taxonomy, it shows the three types of standards: governance, infrastructure and tactical. Also, it divides the categories into discipline-specific and non-discipline-specific elements. Additionally the axis are annotated with directional arrows indicating decision authority of engineering improvement groups. These groups have increasing decision authority towards discipline specific and tactical standards. The matrix is an enabler of logical analysis, by allowing the identification of under or over populated areas, e.g. by populating the matrix an organization may find that lacks operational structures that are enterprise specific. Also, when used to characterize models chosen by different organization it can be used to study model selection patterns, thus enabling standards selection and strategy definition.

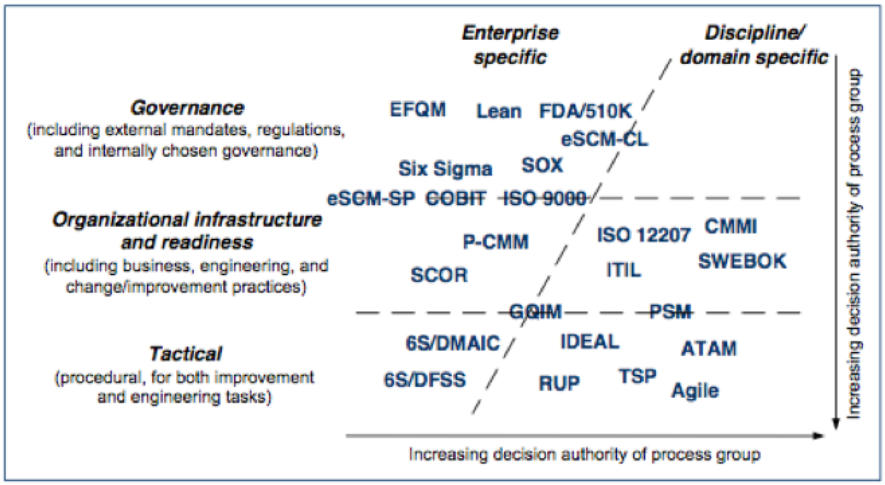

Figure 2.10: Strategic classification taxonomy (in [SF04])

Approaches presented evidence operational differences when selection of models is to be performed. Characteristics based comparisons are useful in providing a high-level overview of models content. Characteristics are seen as properties or attributes that are useful to understand and compare a diverse set of standards. Conversely, bilateral or model mappings are applicable to pairs of models and imply a deeper analysis of models content to find differences and similarities. Characteristics and/or needs mapping approaches are helpful for model selection, allowing to compare a diverse set of models. Models or bilateral mappings are useful in the composition step. Composition may benefit from a lower-level comparison resulting of a deeper model analysis.

Lower level comparisons require consideration of model structural differences and level of detail of of descriptions. Often, gaps exist as a direct result of different elaboration levels and structural differences may also difficult comparisons.

Multimodel Composition and Solution Implementation

Following to the topic of solution implementation Srivastava and Murthy, [SM06] working for Tata Consulting Services, developed an integrated approach using CMMI and Six Sigma. The initial objective was to develop a Quality Framework to address specific strategic challenges of the organization. The choice was to integrate CMMI and Six Sigma, where CMMI provided the what to do and the Six Sigma provided the how to do it. The integration examples provided show a mapping process between CMMI process areas and applicable Six Sigma tools and the integration of Define Measure Analyse Design and Verify (DMDAV) framework with the development phases

defined in the organizational Quality Framework.

The mapping process between Six Sigma and CMMI process areas involved the selection and use of Six Sigma tools to address the specific goals of each CMMI Process Area (PA), e.g. the Quantitative Project Management PA defines the specific goal of statistically manage process performance, Six Sigma provides control charts to help identify special causes of variation. The relevance of the synergy is that by identifying the relevant causes of variations in project management sub-processes is possible to manage the process performance. Other example is given with Causal Analysis and Resolution (CAR) PA that defines a specific goal of determining the causes of defects. The Ishikawa Diagram is a visual tool used to brainstorm and logically organize possible causes to address a specific problem or effect. The use of this type of diagram within the context of CAR helps to identify possible root causes of defects.

Also, an integration example of DMADV framework with the organizational Quality Framework is given. The integration consisted in mapping the development phases defined in the Quality Framework with the framework phases of DMDAV. The organizational framework presented comprehends six phases, namely, Project Start-up, Requirements analysis, System Design, Detailed Design, Build & Test and Verification and Validation. The DMDAV defines five phases Define, Measure, Analyse Design and Verify. The phases map one to one in the order they are given, the exception is the Design phase that map to the Detailed Design and Build and Test in the Quality Framework.

The mapping process incorporates the relevant six sigma tools to be used in the process of elaborating the deliverables for each development phase, e.g the Requirements Analysis uses the Affinity Diagram, the Quality Function Deployment and Critical To Quality Drill Down Tree to help translate the voice of customer to elaborate the System Requirement Specification. The Detailed Design, Build and Test phases uses a wide set o Six Sigma tools, namely: Control Impact Matrix, Cost Benefit Analysis, Design of Experiments Simulation and Modeling, Failure Modes and Effects Analysis and Validation Plans to help elaboration of low level design, unit test plans, unit test specifications and code. The authors emphasize the rigour that Six Sigma provides by the use of DMADV framework and Six Sigma tools. These had a direct impact in defect reduction and process improvements that increased considerably cost savings.

Another work in the topic of solution implementation is presented by [Mut06] with a description of integration of CMMI and Software Engineering Body of Knowledge (SWEBOK) to define a set of organizational standard processes. The objective of integration of CMMI with SWEBOK his to leverage the two approaches, in that CMMI provides *what* to do and the SWEBOK provides *how* to do it. CMMI outlines the process steps and the SWEBOK provides process details.

The integration effort, where CMMI requires the definition of process elements

and SWEBOK provides the implementation details, serves as motivation to develop the base process elements that can be used as building blocks for a process architecture. Insights are given on characteristics that can be considered when defining process elements, where process elements are defined as the fundamental unit of a process and each process element covers a closely related set of activities [SKMM08a]. Within these basic building blocks are task descriptions (functional), when and how tasks are performed (behavioural), who performs the tasks (organizational) and the guiding principles and strategies (meta architecture). Complementing the process characteristics are the process elements components, namely: entry and exit criteria, inputs and outputs, activities, roles and responsibilities, stakeholders, measurements, controls (verification, configuration management), related processes and tools, standards and training.

The need of a process architecture is one of a structural conceptual framework to be used in the definitions of an organizational set of standard processes is common in all approaches.

A process architecture serves as the building block for the integration of multiple standards and models in the process definition exercise, by elaborating on the basic components of process elements. The process elements provide a comprehensive structure for processes definition. The different views suggested aim to characterize a process and identification of components to be specified for each process, constitute the process architecture.

The examples provided, focus on process mappings between CMMI and SWEBOK to develop the organizational standard processes, but do not describe an example of a complete process element and no architectural standard components are presented.

A different approach is presented by Siviy *et al.* [SF04] by introducing a synergistic relationship of the CMMI and Six Sigma at a different level. It uses CMMI process areas and Six Sigma frameworks, Define Measure Analyse Improve and Control (DMAIC),

Practitioners reports of combining Improvement Models

Siviy and Forester [SF04] carried out a significant study to understand a specific context of multiple model adoption. The research focussed in understanding how Software & IT Practices and Six Sigma are being used by organizations. Several leading companies participated in the research, Lockheed Martin IS&S, Motorola and Boeing. Using grounded theory as research method they collected data using case study interviews, surveys and supported this data with literature review.

The findings revealed that Six Sigma is an enabler of SPI initiatives. The evidences gathered show that it helps to integrate multiple improvement approaches in the cre-

ation of a single solution and facilitates roll-outs of process improvement initiatives. Additionally Six Sigma is frequently used as a mechanism to help sustain and improve process performance with Six Sigma adopters having high comfort levels with a variety of measurement and analysis methods.

Also, benefits of integrated process solutions with Six Sigma spread across several domains of IT performers using practices specific to their domain, e.g. best practice for IT Service Management Information Technology Infrastructure Library (ITIL)[20005] Control Objectives for Information and related Technology (COBIT), and CMMI.

Specifically when combined with CMMI, Six Sigma accelerates the transition between maturity levels, moving from maturity level 3 to 5 in 9 months and maturity level 1 to 5 in 5 years. This is possible by the strategic and tactical methodologies and tools made available by Six Sigma. Six Sigma is also used for enabling, accelerating, or integrating capacity of improvement standards. Adopters report quantitative performance benefits, using measures they know are meaningful for their organizations and clients. The findings also show the joint use of Six Sigma, CMMI, and Architecture Tradeoff Analysis Method (ATAM) [KKC00], is useful in the process of developing processes by leverage connections among Design For Six Sigma (DFSS), ATAM, and the engineering process areas of CMMI.

2.4 Conclusion

Both Paulk and Halvosen define a set of characteristics to which frameworks are characterized. The characterization is open, thus a standardized set of responses for each characteristic is not enforced. Heston's proposal, on the other hand, proposes a set of attributes to which frameworks are characterized using a pre-defined ordinal categorical rating scale. The former are more flexible when characterization is to take place but the absence of an unambiguous relational concept may diminish the ability to compare models when compared with the later approach. Heston's includes a scale that embeds automatically a logical relationship, which may facilitate comparisons but on the other hand is less flexible in for characterization. Siviy's affinity matrix is the simplest approach. Models are placed in a matrix quadrant according to their content but the level of information it delivers is more limited. Kirwan's proposal requires a more detailed analysis of models content to apply the proposed classification scheme. For that reason it is more suited for model composition, but it may also be helpful for selection purposes.

In the context of the harmonization framework defined in by [SK08b], the comparison methods mentioned may be applicable in the selection and composition challenges of multiple model integration. The characteristics and needs mappings might be more

aligned with an initial selection stage, where the number of models may be considerable and a high-level characterization may present itself more useful. Composition may benefit from a lower-level comparison by implying a deeper model analysis, allowing the identification of overlapping and gaps among models. For this reason, the framework mappings and bilateral comparison might be more suited to the composition stage. Nevertheless, it may be valuable for selection purposes, if a detailed analysis is required to choose among a reduced set of models. Lower level comparison requires consideration of elaboration of descriptions and structural differences of models architectural components. Often, gaps exist as a direct result of different elaboration levels and structural differences may also difficult the comparison. For instance, ISO9001 uses shall statements to describe model requirements and CMMI-Dev uses the concept of practice to describe what needs to be carried out. Mappings require finding a common ground for reconciliation of these differences.

This chapter provides a background for the research documented in the following chapters of the dissertation. Improving software quality is the major goal of software process improvement approaches. Benchmark based approach is the most adopted SPI approach by organizations investing in SPI. Different types of quality standards were presented and a summary of their detailed objectives was provided.

Additionally, the specific context of software process improvement initiatives in multi-model environment was described. The use of multiple models is an subject of interest for both industry and academia. The motivation is twofold, organizations aim to achieve the cumulative added value of adopting multiple models in a single environments as a strategic approach to software process improvement and secondly to satisfy compliance to industry requirements for standards compliance. In the multi-model context the harmonization framework defined by Siviy [SMM08] provides a summary of the challenges of multi-model environments.

Chapter 3

Size and Complexity Metrics for a Multi-model Improvement Taxonomy

Contents

When organizations adopt multiple improvement models the challenge of selection and integration of models needs to be addressed. Several methods exist to compare models for informing both selection and integration decisions by delivering a qualitative analysis of models considered. In this chapter a quantitative method to characterize the structure of improvement models in terms of size and complexity is proposed. The method is applied to compare ISO standards and CMMI-Dev in quantitative terms.

3.1 Comparison of improvement models

An increasing number of organ3izations are adopting multiple improvement models to improve overall effectiveness and efficiency [SMM08]. Market pressure, competitiveness, regulatory compliance or the need to solve a particular issue are typical business drivers for organizations to adopt multiple improvement models. The goal is to obtain the cumulative added value of each model into one single environment.

The implementation of improvement models can be performed concurrently when several improvement initiatives are implemented at the same time or in sequence, one after the other. Adoption decision may reside at different levels of authority and motivated by different needs and perspectives of distinct business units. One potential risk of these environments is lack of coordination of individual improvement initiatives across different organizational functions [SK08a]. Lack of support and poor coordination increases significantly the risk of failure in implementing multi model environments resulting in misalignment of adopted models originating reconciliation costs [Pau08]. Additionally, not properly understood or unclear relationships between models leads to operational inefficiencies and redundancies leading to productivity losses. The practical result is excessive cost and erosion of benefits of adopting multiple models.

Improvement models often share scope of concern, prescribing practices or defining requirements for a shared knowledge domain. The ability to understand and reconcile similarities and differences of improvement models is a relevant pre-condition for the development of an effective multi-model solution [HP09] [FS98] [Pau08].

One form of analysis of improvement models when these are considered for adoption is the exercise of comparing improvement models. The motivation is that by comparing models one can reconcile differences and similarities for planning a joint implementation.

3.2 Related work

As described in Chapter 2, Halvosen and Conradi [HC01] identified several approaches for comparing improvement models, namely: comparison based on *characteristics*, comparison by *model mappings* based on kernel concepts, comparison based in *needs mappings* and comparison based on *bilateral mappings*. [Tin96] identified *point of view* and *level of detail* as two perspectives of analysis of improvement models. Level of detail is relevant depending on related SPI knowledge of the organization adopting an SPI model. The argument is that less knowledgeable or experienced organizations in SPI will benefit from comparisons at higher level considering few details. When or-

ganizations have relevant knowledge about a model and want to understand less-know models, using the point of view of well known models could be beneficial.

Comparison based on characteristics

Comparison based approaches are generally proposed as taxonomies of characteristics or attributes. Each improvement models subject for comparison is characterized according to a set of attributes considered in the taxonomy. The comparison is based on the resulting classification of each attribute for each improvement model considered. Preferably characteristics should be objective, measurable and comparable [HC01]. Examples of taxonomies based on characteristics are proposed by Paulk [Pau08], Halvosen and Conradi [HC01]. A less elaborated approach is proposed by Siviy [SKMM08b] that proposes an affinity matrix composed by a pair of attributes. Heston and Phifer [HP09] introduce the concept of *process DNA* and *Quality Genes* that summarize concepts or quality characteristics from several key industry standards. Classification is based on an analysis of each model contents to gauge its depth and coverage of each quality gene. Each model is classified for each quality gene as *high correlation*, *some correlation* or *no correlation*. Organizations may interpret quality genes as needs and assess which models are more suitable to meet their process improvement objectives.

Model mappings by comparing kernel concepts

Improvement models are more and less structured in the form they deliver their content. Textual statements or requirements are organized most of the times in an hierarchical approach where levels are used to provide additional detail on the content and focus of the model. The *model mapping* is the process of creating a map from statements or concepts of one model to those of another by means of a mapping function [HC01]. The purpose of mapping is to identify overlaps and correlations between models. An example of a mapping function characterizes correlations as *strong*, *week*, *no correlation* [Tin96]. Examples of models mappings are available in [BPPV09a] and [MSS09a, MS09, MSS09b]. These comparisons require a deeper analysis of improvement models content and provides a detailed understanding of improvement models similarities and differences.

Bilateral mappings

In bilateral mappings two models are compared textually [HC01]. The bilateral mapping often results from a description of the findings of performing characteristics

based comparison or model mappings. It can be useful when knowledge about one model exists and its desirable to view new models in the perspective of the model where most knowledge exists.

Needs mappings

In needs mappings, organizational needs drive the analysis of the improvement models considered for adoption. Models are analysed and described in terms of identified needs.

The four methods for comparison presented differ in terms of operational differences when selection of models is to be carried out. Characteristics based comparisons are useful in providing a high-level overview of models content. Characteristics are seen as properties or attributes that are useful to understand and compare a diverse set of models. Conversely, bilateral or model mappings are applicable to pairs of models and imply a deeper analysis of models content to find differences and similarities.

Characteristics and/or needs mapping approaches are helpful for model selection, allowing to compare a diverse set of models. On the other hand models mappings are useful in the composition step. Composition may benefit from a lower-level comparison, resulting of a deeper model analysis. Lower level comparisons need to consider structural differences of the models and the level of elaboration or detail of descriptions. Often, gaps exist as a direct result of different elaboration levels and structural differences also difficult comparisons.

These approaches have in common the fact that information derived from the comparison is qualitative in nature. None of the approaches focus on developing a quantitative understanding of comparing improvement models. This shortcoming motivates the research work on the subject of comparing improvement models.

Motivation and research goals

Our search for a quantitative approach to compare improvement models is framed in an evolution of the taxonomy proposed by Paulk [Pau08]. The taxonomy goal is to help potential users understand the intent of different improvement models more consistently and thus improve the quality of the information derived from comparing improvement models. Paulk´s taxonomy is composed of a set of attributes divided in tree categories, namely: 1) the *expected user community and its improvement drivers and strategies*, 2) the *framework architecture and structure* and 3) the *diagnostic methods* associated with the framework.

The expected user community and its improvement drivers and strategies category includes four attributes: *audience, work orientation, support infrastructure, and*

drivers for adoption. The framework architecture category has three attributes: *the rating scale, architecture and content*. The last category, diagnostic methods has three attributes: *diagnostic methods, improvement method and certification schemes*.

Paulk encouraged the evolution of the taxonomy and left open how to characterize proposed attributes for each category. In this chapter a method to characterize the attribute *architecture* in terms of size and complexity is proposed and its applicability demonstrated.

As described in chapter 2, when improvement models are considered for a joint implementation several issues arise, namely: identifying similarities and differences in the scope of the models, differences in structure and level of detailed used to describe their content, differences in terminology used and lastly the adoption sequence. A quantitative understanding of the structure and content of an improvement model will improve the quality of the information derived from a comparison. A quantitative understanding will allow a better interpretation of the results of comparing improvement models e.g., it is possible to translate in quantitative terms the differences between comparing CMMI-Dev to ISO9001 and CMMI-Dev to ISO12207?

To develop such method, differences in structure and level of detail of models description are relevant for a quantitative understanding of a comparison. Our research goal is to develop a solution that provides an answer to the following question:

How can improvement models be compared in quantitative terms in the presence of architectural structural differences and different levels of detail?

The following sections describe an approach to derive a quantitative understanding of improvement models and a demonstration case of it's applicability is documented.

3.3 Attributes of size and complexity for improvement models

Can we be compliant with model X and model Y? How much they overlap? Are there any significant differences between them? The answers are hard to attain objectively with methods available. This research effort aims to develop an approach to perform a quantitative evaluation that may be useful to answer questions above. Our approach to deliver a quantitative understanding on improvement models focuses on architectural attributes of size and complexity.

Size

The characterization of size aims to translate the perspective that scope of a model can be measured if one considers a reference scope for comparison and within a shared scope the amount of information present may vary, introducing the concept of elaboration of description. This notion of size may be useful to quantify objectively gaps and redundancies between models. It may provide useful information, for instance, to be used as criteria in estimating effort for model implementation or to evaluate the level of information present in model descriptions.

One of the challenges is the difficulty to find a reference dimension for scope to effectively compare scope size. An *include-all* reference dimension for scope may be infeasible to define properly due to the magnitude of disciplines and domains covered in improvement models.

A second challenge is how to evaluate detail of descriptions. Information is embedded in models structural components in the form of textual descriptions. Different type of components and types of relations are established defining the models architecture. An overview performed on the models considered in our study (detailed in the next section), reveals the use of a hierarchical relation between major architectural elements.

The notion of hierarchy is used to encapsulate descriptions that are further elaborated as the level in the hierarchy increases, providing additional levels of detail in a bounded scope. For instance, CMMI-Dev uses three architectural component types (among others) to organize its content, namely: process areas, specific goals and specific practices. A hierarchical relation of inclusion is present: process areas are described using specific goals which are detailed using specific practices. Supported in this inclusion property, an elaboration hierarchy may be derived to group model descriptions at a desired level of detail.

The inclusion of a component as part of the definition of other component indicates a new level of elaboration or detail, e.g., for the mapped model LP, level $LP_{(n)}$) is further detailed by components at level $LP_{(n-1)}$ and stops at the leaf components, corresponding to $LP_{(0)}$. A similar scenario is applicable for the reference model LR. An elaboration hierarchy comprises an ordering of detail levels (see Figure 3.1) level zero is at the lower level of elaboration. Lower levels form part of components at higher hierarchical levels.

One can consider the number of components each model has at specific level of elaboration but this number does not provide meaningful information on elaboration or detail of a model descriptions. Often, a direct logical comparison between components of different models is easy to establish. The number of components provides a more meaningful notion of detail when is the value is considered in the scope shared resulting from a comparison. Scope might be seen as a shared characteristic in component descriptions. The number of components used to describe the shared scope is

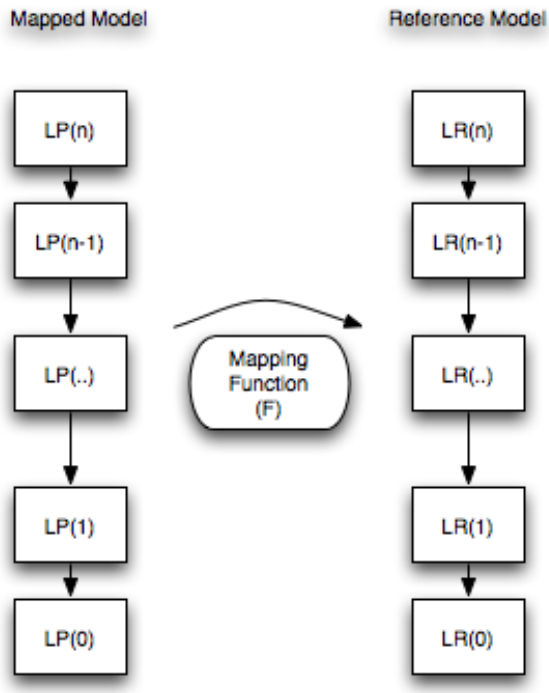

Figure 3.1: Elaboration hierarchy

an indicator of different levels of information present. The argument is that size of a model, to be an objective measure, requires a reference. This reference may be used as an objective assessment of magnitude or breadth for size. However in a shared scope or size there can be different levels of information present and this introduces the notion of depth in size of a model.

Therefore, two dimensions may be considered when conceptualizing size of a model: **scope shared** when compared to a reference model and the **level of elaboration** present within a shared scope by considering the number of architectural components present in each model within the shared scope.

Complexity

Complexity or the perception of complexity appears to be generated by three factors working in combination: variety, connectedness and disorder [Hit03]. Structural connectedness can be used to assess architectural complexity and deliver an objective measure of architectural components linkage of a model. Structural connectedness aims not to provide an indicator of the complexity of implementation but rather understand which models have a *systems* view for their requirements definition. By linking

components, models are explicitly integrating components aside just listing requirements. Structural connectedness may be used to evaluate the systemic view of a model and compare it objectively with existing or future implemented solutions.

The models mapping approach described earlier provides an approach to identify shared scope between two models. The method for quantifying size and complexity of a model is developed by considering the outcome of a models mapping exercise and is detailed in the next sub sections.

3.3.1 Mapping models

The goal of mapping a model to another is to identify similarities and differences of models content. Figure 3.2 represents two models being mapped, a mapped model M_P and a reference model M_R. Area 2 represents the similarities or shared scope and area 1 the differences taking as reference M_P resulting from the mapping exercise. A typical mapping focuses on finding the shared scope of M_R and M_P (shaded area in Figure 3.2) no special attention is given to area 1 and area 3. By extending the analysis to areas 1 and 3 it is possible to extract additional information by considering the dimension of elaboration of descriptions on the shared scope (shaded area). A quantitative analysis of models size is possible by relating areas 1, 2 and 3 by considering differences in the level of information present in shared scope.

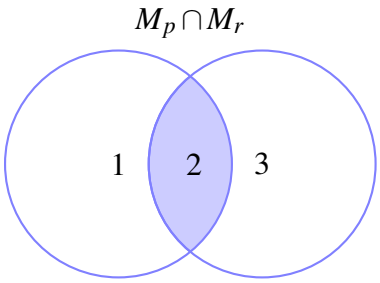

Figure 3.2: Model mapping

A mapping exercise requires two models and a mapping function. A model is chosen to be the reference model M_R and the other to be the mapped model M_P. In a mapping process the scope of the mapped model M_P is analysed for coverage by establishing a mapping between architectural components of the mapped model and architectural components of the reference model M_R.

An example may consider the reference model CMMI-Dev and the mapped model ISO9001. The mapping result identifies the ISO9001 requirements coverage that is attained if an existing CMMI-Dev model is in place. The mapping function F establishes

a relation between architectural components of M_R and M_P. It is important to notice that no assessment for coverage of the reference model is being considered. In fact, the total scope of the reference model is not evaluated in this setting, it provides only the reference base to which overlapping and gaps are identified against the mapped model.

As mentioned earlier, models use components to define the scope of their content descriptions and a hierarchical relation is established to provide a progressive level of detail in descriptions. The mapping function F is defined to compare architectural components scope at specific levels of detail. Figure 3.1 depicts this concept for a mapping function.

In a mapping process the mapping function characterizes the level of shared scope between model components. Coverage is established at $L_P x$ of M_P to a level $L_R y$ of M_R, where $x \in [0, NX]$ and $y \in [0, N_Y]$ and N_X and N_Y are the number of elaboration levels present in M_P and M_R, respectively.

3.3.2 Size

Models used in the Software Engineering domain vary in the scope they cover and the level of elaboration they use in their descriptions. One may argue that ISO9001 is broader and less detailed than CMMI-Dev or that CMMI-Dev is broader than ISO15939 [707] in what relates to measurement and analysis and the latter is more detailed than CMMI-Dev within the measurement scope.

Narrowing the scope often results in increased detail of descriptions. Increased detail is achieved by using a greater number of components at a similar level of elaboration or further elaborating by creating new levels of elaboration. An objective measurement of this perception would help to clarify the subjective nature of these assertions. Defining an elaboration hierarchy of model descriptions (relating model architectural components) provides the scope boundary crucial for performing an evaluation of shared scope and level of detail in descriptions.

As stated previously, comparing models scope size requires the use of a normalizing size reference. Effective comparison implies that a shared characteristic is found to derive a relative measure of size. Scope shared will provide the link to objectively compare scope´s size and the scope of the reference model will be used for this purpose. The number of architectural components, at specific levels of elaboration, covering a shared portion of scope, will be used to measure detail of descriptions. The following concepts are relevant to characterize the attribute of size of a model.

Shared Scope

Shared Scope is a measure of the degree of coverage of the mapped model M_P relative to a reference model M_R. It can be expressed using the following notation:

$$(M_P, M_R, \varphi), \forall \varphi \in \Re \tag{3.1}$$

The coverage factor φ is the result of a mapping performed at a chosen level of granularity, e.g. ISO9001 is covered by a factor of φ by CMMI-Dev when shall statements are compared to CMMI-Dev practices or, considering as reference a full implementation of CMMI-Dev, one can attain φ coverage of ISO9001 requirements. The following concepts are enunciated regarding shared scope.

1. **Mapping function**

   A coverage function F establishes the degree of coverage between a pair of model components. It receives as input a component element C_R of M_R at a desired level of elaboration L_Ry and a component element C_P of the mapped model M_P at the desired level of elaboration L_Px.

   $$\varphi = F(C_P, C_R) \tag{3.2}$$

   The mapping function often assumes the use of a categorical ordinal scale that characterizes the level of coverage that one component has over the mapped component. The scale can be recoded to assume values that translate the meaning of each category into a discrete numeric scale, where the recoded φ assumes values between [0,1] where 0 represent no mapping and 1 a total coverage.

2. **Component mapping**

   A component C_i of a model M_P at a desired level of granularity L_Px is compared using a mapping function F to every component C_j of M_R at the desired level of detail L_Ry.

   $$(C_i, C_j, \varphi_j), i \in [0..n_x] \, and \, j \in [0..n_y] \tag{3.3}$$

   where n_x is the number of components of M_P at L_Px, n_y is the number of components of M_R at L_Ry and φ_j is the coverage factor resulting from applying F. The value of φ_j is the coverage obtained for C_i of M_P by each C_j of M_R.

3. **Component coverage**

A component is covered if exists at least one component in M_R that shares some scope with a component of M_P otherwise is not covered. The highest value obtained in a mapping is the maximum coverage obtained for that component of M_P considering all components of M_R.

Then, a component C_i of a model M_P is said to be covered by M_R by a factor of φ_c if, $\in C_j \; \exists \, M_R$ that:

$$(C_i, C_j, \varphi_j), \varphi_j > 0 \tag{3.4}$$

and $\varphi_c = max\left[\varphi_j\right] \;\; j \in [1..n]$ where n is the number of elements that satisfy equation 3.4 for each C_i.

4. **Scope coverage S_c of a mapped model M_P**

   The shared portion of the mapped model is obtained by identifying architectural components with shared scope. These include components that satisfy $\varphi_c > 0$ in equation 3.4.

$$S_c = \frac{\sum_{i=1}^{n} \varphi_c(i)}{n * \varphi_{max}} \tag{3.5}$$

   In equation 3.5, n is the total number of component maps considered in the mapping and φ_{max} is the maximum possible coverage for each component.

5. **Scope coverage of a reference model M_R**

   Concerning the reference model, it is only feasible to say that a portion of its scope is shared with the mapped model. A component of a mapped model may be covered totally by a component of the reference model, but the opposite may not occur. The reflexive property does not apply in this setting. Still, an approximate measure of scope coverage can be derived considering the cardinality of the set of components from the reference model identified as result of the mapping, where:

$$\Theta = \frac{N}{N_T} \tag{3.6}$$

   where, N is the number of components of M_R at L_Ry referenced in the mapping process, N_t is the total number of components of M_R at L_Ry and Θ is the reference factor.

Elaboration of Descriptions

Elaboration of descriptions uses a measure of the number of architectural components of M_R that are referenced in the mapping process for a scope bounded by each architectural component of M_P e.g., a requirement from ISO9001 maps to x practices of CMMI-Dev. The level of elaboration of a component C_i from M_P regarding M_R is given by the cardinality β_i of the set of components S_R from M_R that satisfies the condition in equation 3.4, where, n is the number of components of M_P at L_Px.

$$(C_i, \beta_i, S_R), i \in [0..n] \tag{3.7}$$

Elaboration of descriptions of M_P, (M_{Pd}) is given by the central tendency mean of frequencies of β_i (n is the number of components of M_P at L_Px, f_i is the number of occurrences of each different group of β_i and f_t the total number of occurrences of f_i's.

$$M_{Pd} = \frac{\sum_{i=1}^{n} \beta_i * f_i}{f_t} \tag{3.8}$$

Values of β in equation 3.7 vary from 0, indicating that a mapped component with any degree of coverage is absent of M_R. The opposite scenario occurs when a component maps to all components of M_R. Large values of β indicate significant differences in the elaboration of descriptions for a shared scope. The inverse relation is also considered for the mapped components of M_R. It measures the cardinality of the set of components C_i of M_P that reference a component C_j of M_R. The relation is expressed as follows: the level of elaboration of a component C_j of M_R regarding M_P is given by the cardinality γ_j of the set of components C_i from M_P that satisfies condition set in equation 3.4.

$$(C_i, \gamma_j), j \in [0..n] \tag{3.9}$$

In equation 3.9, n is the number of components of M_R at L_Ry. Elaboration of descriptions of M_R, (M_{Rd}) is given by the central tendency mean of frequencies of γ_i, where n is the number of components of M_R at L_Ry, f_i is the number of occurrences of each different group of γ_i and f_t is the number of total occurrences of f_i's.

$$M_{Rd} = \frac{\sum_{i=1}^{n} \gamma_i * f_i}{f_t} \tag{3.10}$$

The difference of elaboration in models descriptions is given by the elaboration factor E_f:

$$E_f = \frac{M_{Pd}}{M_{Rd}} \tag{3.11}$$

When E_f approximates 1 it indicates the overall shared scope is described on average by the same number of architectural components at the chosen level of granularity. It is expectable that, in the assumption of similar levels of detail not every component map translates to a one to one relation, some variation may occur. But, if the level of elaboration is similar between models both measures are expected to have the same order of magnitude on an average case. Values greater than one indicate that M_P is less elaborated then M_R for the shared scope at specific levels of elaboration. Values inferior to one indicate more elaboration in M_P.

The notion of balance in elaboration of descriptions for a shared scope is given by the value of standard deviation for equation 3.8 and equation 3.10. High values indicate considerable differences in elaboration of descriptions within a shared scope. Standard deviation of E_f aims to translate an evaluation of homogeneity between models descriptions.

3.3.3 Complexity

In order to establish a measure of model complexity we considered structural connectedness. Every model has its component structural variety, with explicit connections between defined components. Structural connectedness considers any reference that an architectural component has, by definition, to any other architectural component of the same type. Connectedness as a complexity characterization aims to measure the number of internal links that models have in their descriptions. These links are associated with information flow between architectural components and provide a measure of structural intra-dependence. Structural connectedness S_trC is defined as:

$$S_{tr}C = \frac{n}{N*(N-1)} \tag{3.12}$$

where, N is the number of architectural components in the model and n is the number of unidirectional references between architectural components. In this perspective complexity will measure architectural component interconnectedness of a model. If models are used to define a system of processes as result of tactical composition of models as enunciated by [PK08], an analysis of the level of interconnectedness in the model is relevant to inform composition decisions and integration effort or to assess component intra-dependence of an existing system of processes.

3.3.4 Computing Models Size and Complexity

The following steps are required to compare size and complexity of improvement model. The first four steps refer to the mapping exercise, the remaining refer to the

quantitative analysis steps.

1. The process of computing size of a model begins by identifying the models to be considered for comparison. One of the models is chosen to be the reference model M_R and the other the mapped model M_P

2. For each of the models an elaboration hierarchy is identified.

3. A coverage function F is then defined using a coverage scale to map models components at specific levels of elaboration.

4. The mapping is carried out using a mapping function (eq. 3.2).

5. Compute models **Size**.

   5.1 - With the mappings available, the **Shared scope** (eq. 3.1) is obtained by computing the individual component coverages (eq. 3.4 and 3.5).

   5.2 - **Scope coverage** indicator of the reference model is obtained using (eq.3.6).

6. **Elaboration factor** is calculated using eq. 3.7, 3.8, 3.9, 3.10 and 3.11

7. **Complexity** is computed using eq. 3.12

3.4 Measuring model comparisons

CRITICAL Software S.A is a Portuguese software house that achieved a CMMI-Dev (version 1.2) maturity level 5 rating in 2009. CRITICAL had a multi-model Quality Management System (QMS) compliant in different extents with standards like ISO9001, Allied Quality Assurance Publications (AQAP), ISO12207, ISO15504 and CMMI-Dev.

As a former CMMI-Dev level 3 organization (since 2006) CRITICAL had a set of practices in place and a gap analysis between level 3 to level 5 was performed to identify the necessary changes to meet level 5 requirements. Bridging this gap required an evolution of the existing QMS to include new practices. By having a multi-model process solution there was also the need to identify already existing QMS practices that could relate to required practices of CMMI-Dev level 5. Figure 3.3 depicts the exercise carried out to identify gaps between the QMS (CMMI-Dev level 3 compliant) and CMMI-Dev level 5.

Two mappings were performed, the first mapping considered QMS and ISO9001 and the second QMS and CMMI-Dev level 5. In these mappings the QMS was used as

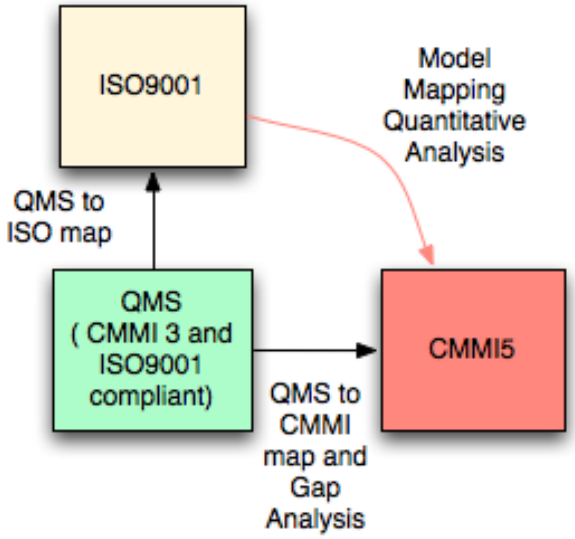

Figure 3.3: QMS gap analysis exercise

the reference model and ISO9001 and CMMI-Dev level 5 as the mapped models. The purpose was to understand the level of coverage obtained for ISO9001 requirements and CMMI-Dev level specific goals considering existing QMS practices. ISO9001 and CMMI-Dev were chosen for this detailed exercise as they represented the most relevant models to CRITICAL business strategy. In order to design and implement new practices, it was relevant to understand how ISO9001 related to CMMI-Dev level 5. The information was important to inform the exercise of defining new practices to be part of the QMS, namely the possibility to adjust or reuse existing practices to fulfil quality requirements. The need, to understand models overlapping and the interest to understand how much of the improvement models were covered by current QMS practices, motivated the development of the method for mapping models at a quantitative level described in this chapter. As a detailed analysis is of the QMS is not possible to document due to CSW confidentiality agreement a demonstration case of it's applicability is documented to demonstrate the level of information derived from the method by considering the following scenario:

1. ISO models will be mapped to CMMI-Dev.
2. CMMI-DEV will be used as the reference model
3. ISO9001, ISO12207 and ISO15288 will be the mapped models.

As stated in sub-section 3.2, the method for comparing prescriptive improvement models is supported by the models mapping technique. In a mapping technique,

three factors may influence the result of the final model coverage, namely: firstly, the mapping function chosen to carry out the mapping, secondly, the subjectivity of the author evaluation and, lastly when considering a pair of models, the level of elaboration chosen for both models to apply the mapping function also influences the result of the coverage. In order to control these variables and have a meaningful comparative scenario, three mappings publicly available by Mutafelija and Stronberg [MSS09b, MS09, MSS09a] were used as a starting point to perform the quantitative analysis. The mappings were authored by the same authors and applied the same mapping function. The last factor does not apply for this setting (this implied having mappings of two models with maps at different elaboration levels). An architectural analysis of models considered for the exercise is described next.

CMMI-Dev

The central architectural component in CMMI-Dev is the process area. It groups related practices that together help achieve a set of goals that are important to implement the respective process area (see Figure 3.4). Process areas are grouped using maturity levels or process categories. Each process area is elaborated using goals which can be specific or generic. Each specific goal is detailed using specific practices. For each practice, further detail is given in the form of typical work products and sub-practices. Generic goals and associated practices follow the same configuration, but apply to all process areas and are denominated institutionalizing practices.

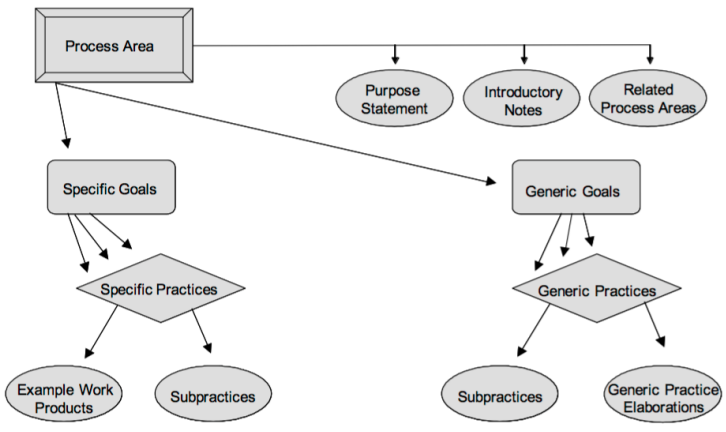

Figure 3.4: CMMI-Dev Model components (in [Tea10])

ISO12207

ISO12207 central component is the process, grouped into two major divisions, System Context Processes and Software Specific Processes. These two divisions are further divided into seven sub-divisions of groups of processes. For each sub-division process descriptions are proposed. Each process description defines the scope, purpose, outcomes and activities. Each activity is detailed using tasks.

ISO15288

ISO15288 uses the same structure of ISO12207 but one division is suppressed (division related to software).

ISO9001

ISO9001 content is based on requirements that are expected to be confirmed during appraisals. These are to be met by organizations when conformance with the standard is desired. The standard uses what could be characterized as a decomposed requirement structure with five levels of elaboration. Requirements are grouped in categories and these categories define the scope of requirements that are detailed using requirement items. Each item is elaborated using textual descriptions in the form of shall statements. These can assume the form of shall clauses. Shall statements are the core architectural component of ISO9001 standard.

To demonstrate the applicability of the quantitative method developed, a description of how the steps described in sub-section 3.3.4 are performed is described next:

1. Step 1: CMMI-Dev is the reference model (M_R) and ISO12207, ISO9001 and ISO15288 are the mapped models (M_P).

2. Step 2: considering the analysis performed of the mapped models performed, the mappings were performed between *shall statements* in ISO9001 and *tasks* in ISO 15288 and ISO12207 and *specific goals* and *generic goals* in CMMI-Dev (version 1.2).

3. Step 3: the coverage function used considered a ordered categorical scale containing values 0, 30, 60, 100 to represent the level of coverage between model components.

4. Step 4: The mapping step resulted in three mappings publicly available by Mutafelija and Stronber [MSS09b, MS09, MSS09a].

The final steps, 5 to 7, related to the quantitative computation of *Size* and *Complexity* are detailed in the next sub-sections.

3.4.1 Size: shared scope

Shared scope between ISO9001, ISO1207 and ISO15288 were computed and as example, the shared scope calculation between ISO9001 and CMMI-Dev is detailed.

The component mapping value, resulting from step 4, for each ISO9001 requirement (shall statement) to CMMI-Dev specific goal was used. Figure 3.5 shows a partial result of the coverage value given available in [MS09] and Table 3.1 shows the results of computing component coverage using (eq. 3.4). For each *Requirements Keyword* column in Table 3.5 the maximum coverage is determined considering each map established, e.g., *Establish QMS* requirement maps to 29 CMMI-Dev practices, 22 practices correspond to one generic practice (GP 2.1) of all 22 process areas and an additional 8 practices from 2 specific process areas (OPF and OPD) with a maximum coverage of 100 for each CMMI-Dev practice, fully covering the ISO requirement. The result of this calculation is depicted in Table 3.1 for each requirement map in Table 3.5.

Sect.	ISO 9001:2000 Requirement Keywords	CMMI PAs	CMMI Practices	Conf
4	**Quality management system**			
4.1	**General requirements**			
	Establish QMS	All PAs OPD OPD OPD OPF OPF OPF OPF	GP 2.1 SP 1.1 SP 1.2 SP 1.3 SP 3.1 SP 3.2 SP 3.3 SP 3.4	100 100 100 100 100 100 100 100
	Identify processes	OPD	SP 1.1	100
	Determine sequence	OPD	SP 1.1	100
	Effective operation	OID OPF	SP 2.1 SP 3.3	30 30
	Resources	All PAs	GP 2.3	100
	Monitor processes	All PAs All PAs	GP 2.8 GP 2.9	100 100
	Implement actions	OPF OPF	SP 2.1 SP 2.2	100 100
	Manage using ISO standard	All PAs	GP 2.1	60
	Control outsourced processes	SAM SAM SAM	SP 1.3 SP 2.1 SP 2.2	100 100 100
	Outsourced process control in QMS	SAM SAM SAM SAM	GP 2.2 SP 1.3 SP 2.2 SP 2.3	100 100 100 100

Figure 3.5: Partial table of ISO9001 to CMMI-Dev mapping (in [MS09])

With all values computed for individual component coverage, the shared scope is obtained computing (eq. 3.5). Each value in the coverage column of Table 3.1 is

added and then divided by the number of architectural components multiplied by the maximum coverage for each component. Using notation in equation 3.1, shared scope for ISO mappings and CMMI-Dev is, (see also Figure 3.6):

$$(ISO9001, CMMI-Dev, 0.83) \tag{3.13}$$

$$(ISO15288, CMMI-Dev, 0.77) \tag{3.14}$$

$$(ISO12207, CMMI-Dev, 0.74) \tag{3.15}$$

ISO9001		Coverage	Number of associated practices
4.1 General Requirements	Establish QMS	100	29
	Identify processes	100	1
	Determine sequence	100	1
	Effective operation	30	2
	Resources	100	22
	Monitor processes	100	44
	Implement actions	100	2
	Manage ISO standard	60	22
	Control outsourced processes	100	3
	Outsourced process control in QMS	100	14

Table 3.1: ISO9001 component coverage

The results shows that an organization compliant with CMMI-Dev can be confident that the ISO9001 requirements are address in a measure of *.83* of the overall set of requirements. The similar conclusion is possible related to ISO12207 and ISO15288. The value for the shared scope of 12207 and 15288 when having a CMMI-Dev implementation is *.77* and *.74* respectively.

Shared scope indicator of the reference model

An additional calculation was performed to have the value for the shared scope of the reference model, in this case CMMI-Dev. The indicator translates a measure of the percentage of CMMI-Dev scope addressed by each of the ISO standards.

Two groups of CMMI-Dev practices were identified, the first with practices referenced in the ISO's mappings with a shared scope ($\varphi_j > 0$ in equation 3.3) and a second

group of practices with no reference ($\varphi_j = 0$ in equation 3.3). Tables 3.2 and 3.3 show an example of this exercise for the CMMI-Dev Requirements Management Process Area. In the *Reference* column a value of 1 indicates the practice is referenced by a shall statement of ISO9001 and a value of 0 indicates that no reference exists. This exercise was replicated for all 22 PAs of CMMI-Dev. Equation 3.6 was used to obtain an indicator of scope shared with CMMI-Dev, using the number of elements of the first group as N in equation 3.6 and the total number of practices as N_t in equation 3.6 by adding elements of the first and second groups. This calculation was replicated for ISO12207 and ISO15288 mappings.

PA	Goal	Practice	Reference
REQM	Specific	1.1	1
		1.2	1
		1.3	1
		1.4	1
		1.5	1

Table 3.2: Requirements Management PA scope coverage by ISO9001 - Specific Practices

Values of the shared scope of the reference model show that in ISO9001 to CMMI-Dev map 68% of overall CMMI-Dev practices are referenced in the mapping. A total of 88% relate to maturity level 3 specific goals (see Figure 3.7). ISO15288 map references 53% of total CMMI-Dev practices and a 66% of practices for a maturity level 3 scope. ISO12207 map references 56% of all CMMI-Dev practices and 72% of practices for a maturity level 3 scope.

It was also considered how ISO models are referencing generic practices (GP) and specific practices SP practices, separately. ISO9001 map establishes references to 88% and 82% to GPs and SPs, respectively (see Figure 3.8). In ISO15288 map a total of 53% and 80% practices are referenced, respectively. For ISO12207 a total of 61% and 77% practices are referenced, respectively.

The scope of GPs considered is bounded for a Capability Level 3 as no references for level 4 and 5 GPs occurs in any of the maps considered.

3.4.2 Size: elaboration of descriptions

To understand the difference in elaboration or detail of descriptions we computed equation 3.7 for each shall statement in ISO9001 to CMMI-Dev-Dev mapping and the opposite relation for each CMMI-Dev practice using equation 3.9.

PA	Goal	Practice	Reference
REQM	Generic	1.1	0
		2.1	1
		2.2	1
		2.3	1
		2.4	1
		2.5	1
		2.6	1
		2.7	1
		2.8	1
		2.9	1
		3.1	1
		3.2	1
		4.1	0
		4.2	0
		5.1	0
		5.2	0

Table 3.3: Requirements Management PA scope coverage by ISO9001 - Generic Practices

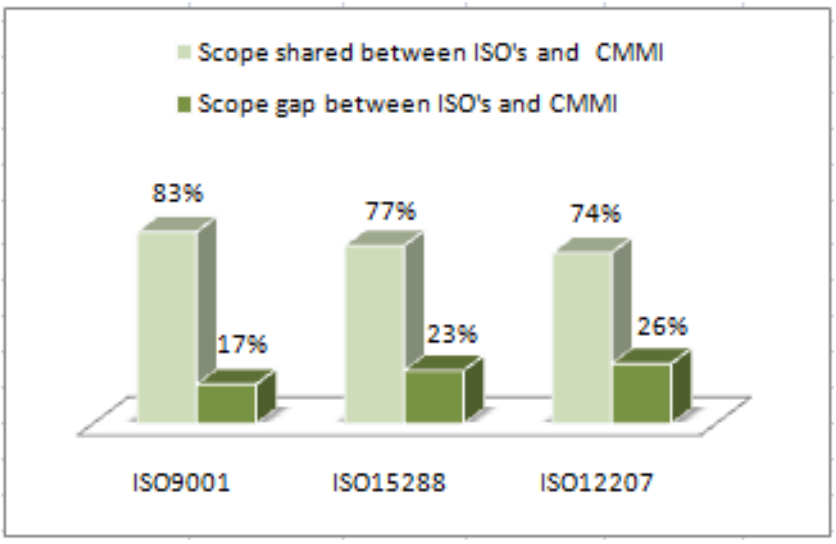

Figure 3.6: Coverage of ISO standards by CMMI-Dev

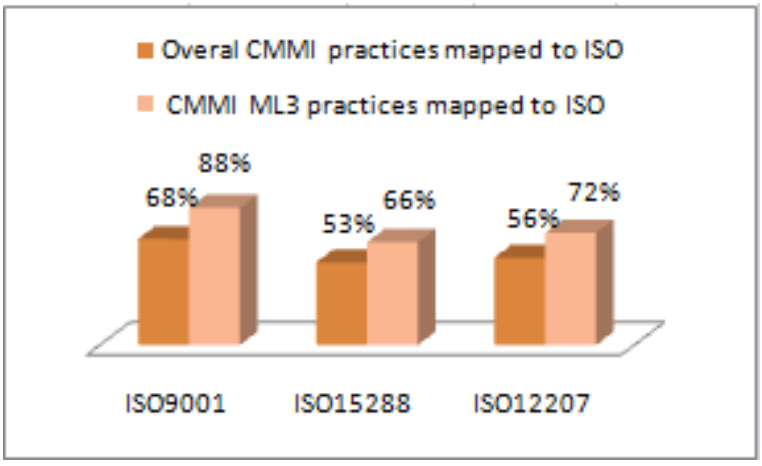

Figure 3.7: CMMI-Dev Practices Referenced in the Mapping Process for each ISO standard

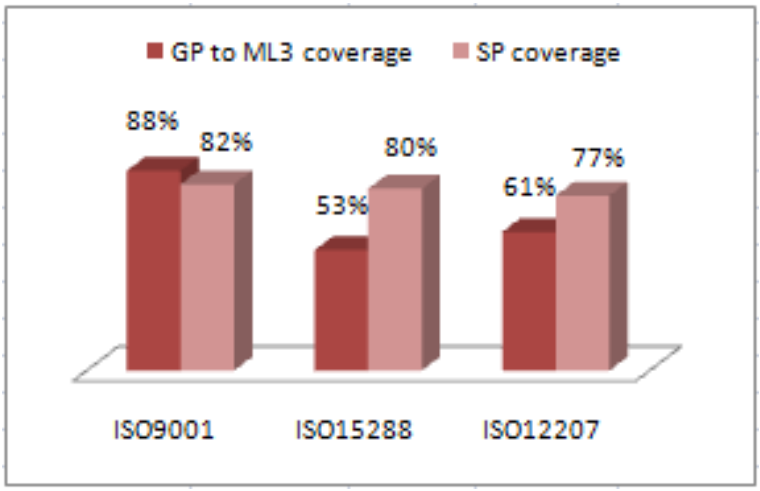

Figure 3.8: CMMI-Dev GP and SP Referenced by ISO Standards

Figure 3.9 shows a single occurrence (y-axis) of a shall statement making reference to a group of 67 CMMI-Dev practices (x-axis) and that exists 57 occurrences of a shall statement making reference to a single CMMI-Dev practice. Figure 3.10 shows the opposite relation. Two occurrences of a CMMI-Dev practice referenced by a group of 12 shall statements and above 80 occurrences of a CMMI-Dev practice referenced by a group of single shall statements.

Computing equation 3.8 and equation 3.10 based on the frequency analysis we obtain the central tendency of architectural components referenced for each model. The values are summarized in Figure 3.11 with values of E_f applying equation 3.11.

An average value of 7 CMMI-Dev practices are referenced by each shall statement with 10.2 units of standard deviation. Conversely, each CMMI-Dev a practice is refer-

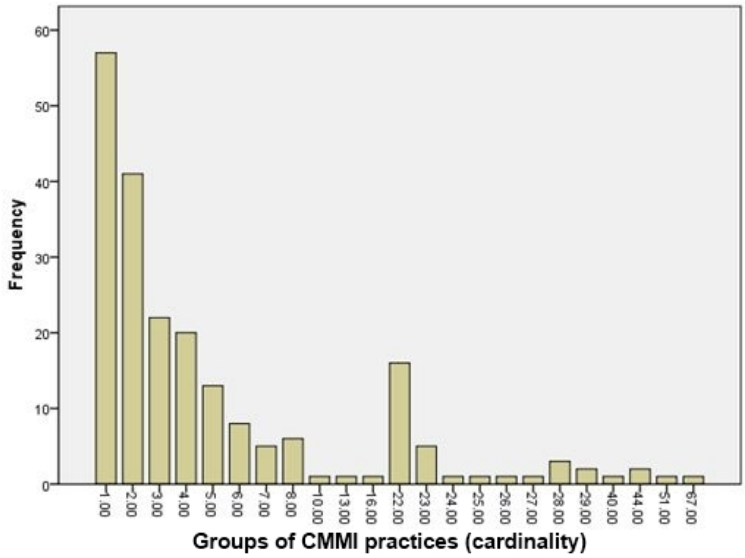

Figure 3.9: Frequency Analysis of Shall Statements Referencing to CMMI-Dev Practices

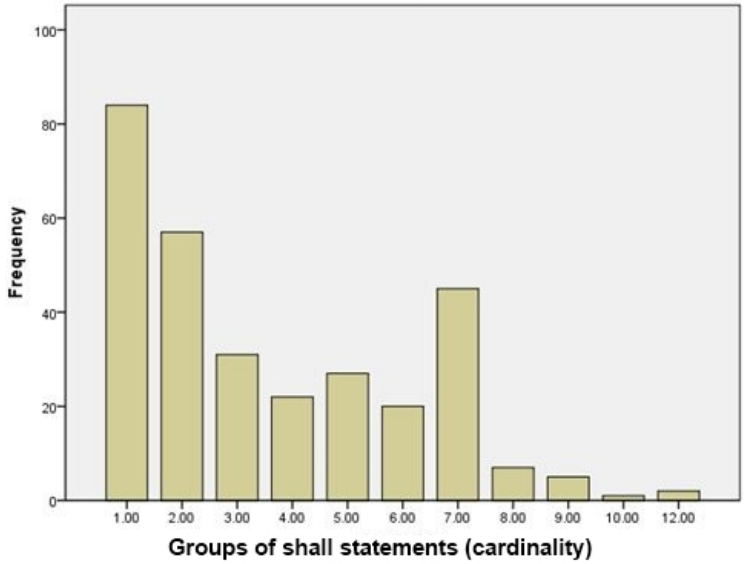

Figure 3.10: Frequency Analysis of CMMI-Dev Practices Referenced by Shall Statements

enced by an average of 3.6 shall statements with 2.5 units of standard deviation, with an E_f of 1.95. ISO15288 tasks make reference on average, to 3 CMMI-Dev practices with 5.4 units of standard deviation and each CMMI-Dev practice is referenced, on average, by 2.1 tasks with 1.5 units of standard deviation, resulting in an E_f of 1.38.

Thirdly, ISO12207 shares, on average, each task scope with 3.2 CMMI-Dev practices with 6 units of standard deviation. Each CMMI-Dev practice is referenced on average by 2.7 tasks with 2 units of standard deviation. This leads to a 1.16 elaboration factor between ISO12207 and CMMI-Dev.

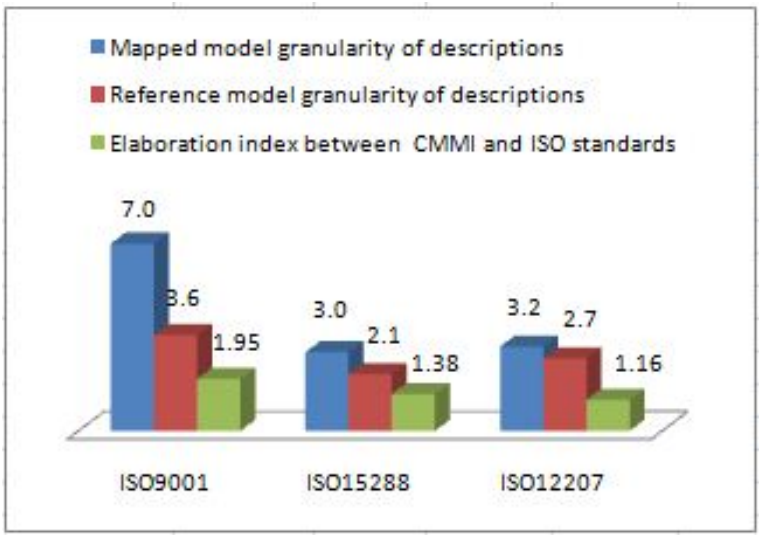

Figure 3.11: Elaboration Factor (E_f) between ISO Standards and CMMI-Dev

3.4.3 Complexity: structural connectedness

Both ISO12207 and CMMI-Dev define explicit links at process level. ISO15288 has no inter-process or any other type of internal references. For ISO12207, internal references origin at the task level but the ending reference is the process architectural component. For simplicity and not jeopardizing the semantics of the connection defined, when links are established from task to process it will be considered the process to which task belongs, resulting in a process to process link.

A similar scenario occurs in ISO9001, it defines internal links that originate at shall statement level but reference an ending requirement item. It will be considered a link between the originating item requirement that has a shall statement referencing another item requirement. Table 3.13 shows a reference matrix elaborated to identify ans compute the number of links between CMMI-Dev process areas. Figure 3.12 depicts the values obtained using equation 3.12 for each model considered (each value is multiplied by a factor of 100 for readability).

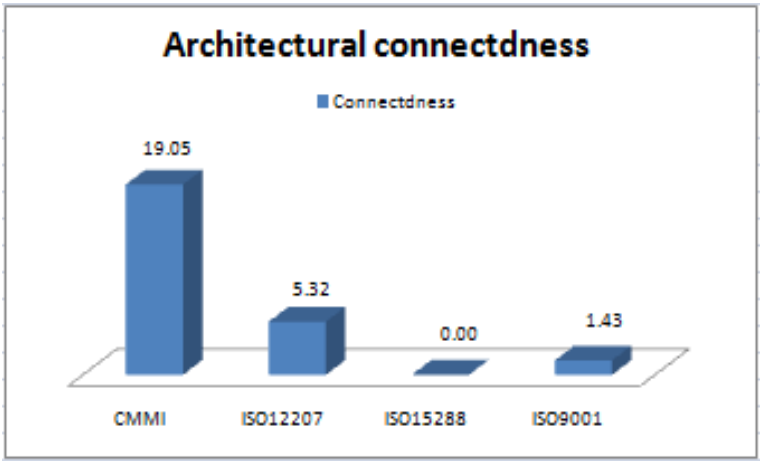

Figure 3.12: ISO and CMMI-Dev Structural Connectedness

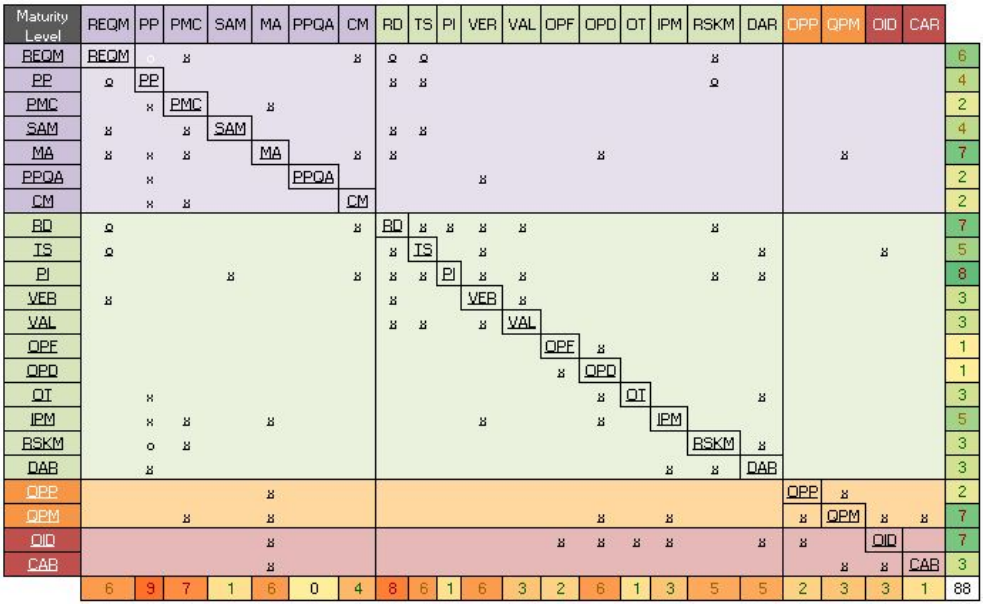

Figure 3.13: CMMI-Dev Process Area Reference Map

3.5 Analysis and discussion of results

The method to compare in quantitative terms several models helps to understand the significant (above 74%) shared scope between the ISO mapped models and CMMI-Dev (Figure 3.6). The majority of ISO9001 requirements out of scope of CMMI-Dev are relative to the ISO clause *Control and Monitoring of Measuring devices*, that justifies 50% of shall statements not in the scope of CMMI-Dev (relates to area 1 in Figure 3.2). The majority of tasks of ISO15288 out of scope of CMMI-Dev are relative to the *Operation Process tasks* fulfilling 33% of total task not addressed by CMMI-

Dev. The remaining are associated to single tasks spread across other processes. In ISO12207 about 50% of scoped out tasks relate to the Software Operation Process and Software Reuse Process areas.

The scope coverage analysis of the reference model helped to understand that high maturity practices of CMMI-Dev are poorly addressed by ISO standards (Figure 3.7). Generic Practices of high maturity levels (level 4 and 5) of CMMI-Dev are out of scope of ISO standards. Also, four Process Areas of CMMI-Dev level 4 and 5 are significantly out of scope of ISO12207, scoring below 25% on referenced practices, with Causal Analysis and Resolution process area totally absent of ISO12207. A similar scenario occurs in ISO15288 with three of the highest maturity Process Areas, with less than 40% referenced tasks and with Organizational Innovation and Deployment Process Area, referenced only at 7%.

Additionally, the information resulting from applying the method provides a better understanding of shared scope between ISO9001 and CMMI-Dev when compared to the other ISO standards considered. ISO9001 shall statements make reference to 88% of practices of CMMI-Dev maturity level 3, scoring higher than ISO15288 and ISO12207. It also seems to be more balanced when considering GP´s and SP´s. ISO9001 covers both SP´s and GP´s in a similar degree with 82% and 88% of practices referenced, respectively. The other ISO standards are significantly less oriented to cover GP´s. Two process areas of CMMI-Dev are completely absent of ISO9001, namely Causal Analysis and Resolution and Decision Analysis and Resolution.

Concerning elaboration of descriptions, based on E_f values, we argue that CMMI-Dev, within the shared scope, is more detailed than any of the model considered in the mappings (Figure 3.11). Comparatively, ISO9001 is less elaborated followed by ISO15288 and thirdly ISO12207. The measures of E_f seem to translate the intuition that ISO9001 being a general purpose standard may be less elaborated when compared to a model that targets product and services development. Also, ISO12207 could be considered the most similar model to CMMI-Dev with a smaller E_f value, translating the notion that the level of information describing the shared scope is somehow similar. Finally, within the shared scope, ISO9001 is more heterogeneous in addressing elaboration of requirements comparatively to ISO12227 and ISO15288 tasks. The values of standard deviation indicate that subsets of requirements are considerable less elaborated than others. This variation is less significant in the other ISO standards.

Links between architectural components in model descriptions provide information to guide the development and implementation of process architectures. On this subject the differences between models are considerable. CMMI-Dev is the only model that includes in its structure an architectural component to define links between process areas. ISO standards establish links when these are considered convenient, but the approach is not systematic as in CMMI-Dev. The result is that CMMI-Dev is more

informative on this matter, delivering relevant information to guide the development of a system of processes.

3.6 Conclusion

Organizations adopting multiple improvement models into a single environment face the challenge of integration of adopted models. Typically integration requires some sort of comparison of models content. Several methods exist to compare models in qualitative terms but none of the approaches delivers a quantitative perspective of the analysis.

In this chapter a method to characterize improvement models is described, by proposing measures of size and complexity as a characterization of improvement models architecture. This is proposed as an evolution of a taxonomy defined by Paulk [Pau08].

The goal was to improve the quality of information that can be derived from a comparison by delivering an objective analysis of improvement models architecture. The method is based on the model mapping technique to compare models and acknowledges that models often share scope and use different levels of detail when defining their content. Also, to characterize models complexity we considered structural connectedness to provide an objective measure of the level of information present in the model in terms of content inter-dependency.

The analysis revealed that ISO standards share a high percentage of scope with CMMI-Dev and that CMMI-Dev is ahead of ISO standards in terms of prescribing relations between architectural components.

A quantitative understanding provides a more objective analysis of improvement models differences and similarities. A demonstration of the applicability of the method in delivering a quantitative analysis of comparing ISO models and CMMI-Dev was documented in detail.

Size in terms of *Shared Scope* metrics allows to have an overview of the similarities and differences between two mapped models. This information may be relevant to elaborate to higher levels of information, namely construction or comparative maps between improvement models e.g., quantitative affinity matrices.

Elaboration of description information can be used to identify how similar one model is when compared to a reference model in their shared scope. This allows to develop a similarity map between models, considering the extent of information they evidence for their shared scope.

The method can be used by improvement groups for developing a systemic quantitative analysis on model comparison and evaluate or justify their adoption with quan-

titative information. Additionally, for normative bodies and organizations defining improvement models, it allows building comparative quantitative charts of their models. It may also be used to evolve or reconcile content with other models.

Chapter 4

Compliance in Multi-model Environments

Contents

Software development organizations are adopting multiple improvement models to guide their improvement efforts. A recent trend is the simultaneous adoption of CMMI-Dev and ISO models into a single environment, originating multi-model process solutions. In this chapter, a model to manage compliance of organizational practices in environments where multiple improvement models are adopted is proposed. A demonstration case of how it can be used to manage an assessment for compliance of a process area that is shared by CMMI-Dev, ISO12207 and ISO9001 is documented.

4.1 Compliance in multi-model environments

Organizations who choose to adopt more than one improvement technology to improve their overall performance are motivated by several reasons. One of which is normative compliance required by specific markets, *e.g.* Aviation and Space market segments require that suppliers comply with specific standards. A second reason is the need to comply with regulatory requirements, e.g., standards mandated by regulations or laws and lastly as a strategic approach for overall performance improvement.

A recurrent combination of improvement models is the simultaneous adoption of CMMI-Dev and ISO9001 into a single environment. This combination is probably the most analysed combination of improvement model in the software sector and translate to a good example of how multiple improvement models can be used in a single organizational environment [MS03][YYL+04] [BCPP12]. Other example is the combination of CMMI-DEV and ISO12207 [BPPV09a].

As described in Chapter 3 benchmarked based approaches often share a portion of their scope. A high degree of shared scope can exist between improvement models targeting the software domain, namely between CMMI-Dev, ISO12207 and ISO1528. This shared scope often translates to practices prescribed that have similar or comparable end goals. A practical example is the CMMI-Dev Process Area of Configuration Management that prescribes 3 specific goals composed of 7 specific practices and ISO12207 prescribes 2 Activities and 4 Tasks addressing the same knowledge domain of Configuration Management for Software Engineering.

Another form of combining improvement models is given by Siviy *et al.* that identifies tactical combinations between Six Sigma [Pyz03] and CMMI-Dev [SKMM08b]. Elements of each considered improvement approach are compared and mapped to identify possible tactical combinations to drive process improvement. The combination of both approaches is centred in identifying synergies between elements both CMMI-Dev and Six Sigma. CMMI-Dev is a benchmark based approach and Six Sigma is a principle based or analytical approach and this combination is sometimes denominated *what/how* combinations. A *what/how* combination refers to the applicability of prescriptive or benchmarked based improvement models and principle or analytical based improvement model into the same organizational environment where one, the how, is enabling the other, the what, rather than comparing the scope of the practices prescribed.

Benchmarked based approaches often require a formal recognition of their adoption. This is performed by an external organization, also know as a registrar or certification body. The registration process verifies that an organization is compliant with the improvement model adopted by performing a set of audits or assessments that upon success result in a compliance certificate. This is the case for ISO9001 that has a three

year during which the certificate is valid (includes yearly surveillance audits).

CMMI-Dev has a similar model with an appraisal process that assesses and certifies that an organization is operating at a specific maturity level. Typically improvement models also require that adopting organisations perform internal verifications of compliance, by performing internal audits or assessments. On the other hand, the analytical based paradigm typically does not involve a formal registration as it relates more to *know-how* rather than a prescription for what needs to be done.

When adopting benchmarked based improvement models the need to ensure compliance is twofold. First, to make sure the adopted practices are implemented in the organization and thus the improvement strategy is in place and effective. Second, make sure that the external audit performed by the registrar is successful and formal recognition of compliance is achieved. In summary, compliance requires identifying an implemented practice and an actual outcome of that practice that satisfies the prescribed practice adopted.

Organizations adopting several benchmark based improvement models face the challenge of ensuring compliance with multiple improvement models. Based on the premise that improvement models can share scope of concern by prescribing practices that are similar in their final intent, identifying similarities and differences in model adopted becomes relevant, not only to define an joint implementation plan but also to verify compliance of adopted practices.

A related concern in managing compliance with improvement models is that models change and evolve over time, by releasing improved and revised versions. This happens both with CMMI-Dev and improvement models by ISO and IEEE. As result, organizations inevitably need to change their practices by adopting new practices or dropping deprecated practices.

Problem definition and research objectives

In a managerial prescriptive, assuring traceability between implemented practices and prescribed practices is a good investment. Traceability supports the evolution of implemented practices by allowing to check if a change in an organizational practice impacts the compliance with an adopted practice from an improvement model. This is straightforward in single environments but in multi-model environments a strategy is required to consider possible similarities and differences of adopted improvement models. An *ad-hoc* approach is prone to inefficiencies that can jeopardize effectiveness and goals of assuring compliance.

Our research goals is defined as:

Research Goal: How to manage structural and terminology differences of multiple models to support the evaluation of compliance with multiple improvement models in a single organizational environment and in a single iteration.

The scope of our problem is bounded by the following considerations:

- Multi-model environments result from the combination of more than one benchmark based improvement model (this excludes combination with analytical approaches, where the analytical approach is evaluated in some other form of compliance assessment).
- Improvement models used to form the multi-model environment may or may not share scope of concern.

As stated earlier, an improvement model requires some level of internal checks for compliance, most of the times in the form of audits and assessments. In multi model environments when shared scope exists between improvement model, an opportunity exists to maximize the efforts spent in verification of compliance by considering the possibility of joint audits. This means performing a single audit and verify compliance to more than one improvement model.

Audits and assessments in multi-model environments will benefit from identifying similarities concerning purpose and expected outcomes of considered improvement models. Specifically, one of the opportunities identified in harmonizing multiple improvement models is to optimize costs in audits and assessments for operational units and projects [SKMM08c].

To our knowledge, previous research has not considered how information on identified shared scope between improvement models can be managed to support multi-model audits and assessments in a single effort.

4.2 Related work

Siviy *et al.* introduced the concept of harmonization as an approach to deal with multi-model environments challenges [SKMM08c].The harmonization framework described in Chapter 2 identifies the steps needed to choose, compose and implement multiple improvement solutions. Harmonization is introduced as a general framework to align different improvement models into a single environment. in summary, the following steps are defined:

1. Alignment of organizational and improvement objectives and identification improvement technologies

2. Categorization of improvement technologies, strategically

3. Design the improvement solution and

4. Implement the multi-model solution and measure results.

A technique applied in multi-model scenarios is the model mapping technique. It is used to compare improvement models with the purpose of finding associations between improvement models by using a mapping function. Model mappings can be used in the harmonization context to support of selection and composition of improvement models. The semantic associated to the mapping function determines the type of model comparison and composition.

In recent literature, the most recurrent type of comparison, involves comparing model in terms of purpose and expected outcomes, also denominated what/what comparisons, named as *degree of relation or support* in [BCP+10] or mapping confidence in [MS03] and characterizes the amount of shared scope between models. Examples of these mapping exercises are provided by Pino *et al.* in [PBP+10, PBPV09, BPPV09b].

When organizations adopt several improvement models, the sequence of model adoption becomes is also a relevant consideration. One may choose a first model for adoption, carry out an implementation of the chosen model and then choose a second model for implementation, following the implementation of the first model. Other possibility is to choose more than one model and plan a joint implementation.

When adopting more than one improvement model, harmonizing is beneficial to improve efficiency of joint implementations [SKMM08c]. Harmonizing focuses on finding possible similarities and synergies of chosen models to facilitate and improve efficiency of joint implementations.

When adopting an improvement model, practices and/or requirements are interpreted according to organizational specific context and needs. Organizational specific practices are defined and implemented aligned with adopted improvement models.

If a more formal approach is used to define organizational practices, e.g., to meet CMMI-Dev maturity level 3 goals, practices definition may be formalized using process models and/or process modelling languages, e.g., SPEM (Software & Systems Process Engineering Meta-model Specification version 2.0) provides relevant process concepts for process definition. The result is an OSSP (Organizational Set of Standard Processes) that provides a collection of process and practices definitions to be enacted by the organization. From this set, project or organizational units define specific processes considering tailoring guidelines if applicable.

The design of multi-model environments will benefit from an explicit step for harmonizing models before practices are incorporated in the organizational environment.

Figure 4.1 depicts a high level interpretation (not exhaustive on concerns related to harmonization) of the harmonization framework introduced by Siviy *et.al.* in [SMM08].

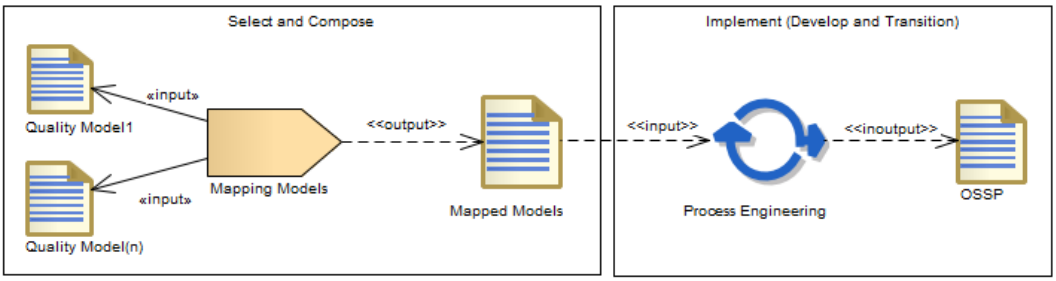

Figure 4.1: High level Harmonization Framework

In designing our solution to support multi-model audits and assessments we considered the scenario where a *model mapping* technique is applied and the organizational environment has an OSSP (Organisational Set of Standard Processes) defined. An OSSP provides detailed definitions on how the organization processes should be performed. This means that a least a cycle in the in Figure 4.1 was completed.

The motivation to consider such scenario is twofold. First, although model mappings are subjective in nature, they provide an objective evaluation of models similarities and differences. This is relevant to inform the scenario of a joint audit where improvement models overlapping needs to be identified.

Second, this research was carried out in the context of a Portuguese software house with a multi-model process solution. CRITICAL Software S.A. that has achieved a CMMI-Dev maturity level 5 rating and complies with standards like ISO9001, Aerospace Standards 9100 and 9006 and ISO12207 and with a well documented QSSP.

When devising our solution the following assumptions was considered:

- A mapping exercise between improvement models that aims to identify shared scope as the semantic associated to the mapping function provides a first level guidance on identifying possible reuse points for joint audits and assessments.
- Further, if an OSSP provides the necessary detail on how practices should be performed and these practices are aligned with one or more improvement models, OSSP elements can be reused to plan data collection tasks facilitating the implementation of audits and assessments on a single effort.

4.3 Managing compliance in multi-model environments

The purpose of a model mapping exercise is to find similarities and differences between a pair of improvement models. A mapping involves pairs of models and a

mapping function that relates entities of both models to deliver a mapping result. A mapping result is a set of relations categorized by the mapping function between all descriptive structural element of one model to all descriptive structural elements of the second model. When mapping models, differences in structure and terminology need be analysed to execute the mapping e.g., CMMI-Dev defines *Specific Practices* within *Process Areas*, ISO12207 uses *Activities* and *Tasks* and ISO9001 uses *shall statements*.

The differences in structure result mainly from differences in terminology when the different normative bodies organized the information available in models. However, most often it's possible to map semantically similar terms, *e.g.* and *Activity* in ISO12207 is semantically similar to a *Specific Practice* in CMMI-Dev. A mapping most often is performed comparing semantically similar elements

When executing a mapping between improvement models with objective of providing support to joint audits and assessments, the following considerations assume central relevance:

1. A descriptive element from a improvement model can share a **scope of concern** with one or more descriptive elements from other models. The degree of the sharing or similarity can be characterized quantitatively or qualitative, e.g., a CMMI-Dev practice can share, with different degrees of similarity, scope with several ISO9001 shall statements. A descriptive element in this context can be seen as a *quality requirement* for the fulfilment of implementing an improvement model. In practice, the mapping defines how much of one quality requirement when implemented can be re-used to support the implementation of a mapped quality requirement.

2. The degree of similarity of scope between descriptive elements of different models is not **reflexive** (*à priori*) – *e.g.*, stating that a CMMI-Dev *specific practice* is related in a certain degree to an ISO9001 *shall statement* is not the same as stating the mentioned ISO9001 *shall statement* is related to the CMMI-Dev *specific practice* in the same degree. This fact has been also identified by [PBP+10].

3. The first consideration assumes that a relation can be established between descriptive elements, or quality requirements, to characterize the degree of shared scope. Whatever the scale used for characterizing the degree of relationship, the semantics associated should be how related are intended **purpose** and **expected outcomes** of compared quality requirements whether these relate to a product outcome or service outcome), e.g., the contents of the output can be used as evidence to demonstrate, partially or totally, the fulfilment of the mode.

One relevant point in developing a generic solution to manage compliance in multi-model environments is one that can be agnostic of the structural differences of im-

provement models. Based on the aforementioned considerations two constructs are proposed to model the information relevant in managing compliance in multi-model environments, namely:

- **quality requirement** - refers to any descriptive element of an improvement model, *e.g. specific goal* in CMMI-Dev of *activity* in ISO12207 or *shall statement* in ISO9001.
- **coverage** - is an association between quality requirements that results from applying a mapping function that characterizes how similar are *purpose* and *expected outcomes* of a pair of quality requirements.

Figure 4.2 depicts this relation where a **quality requirement** can be related to multiple quality requirements by a **coverage** value that translates how related are intended **purpose** and **expected outcomes** of compared quality requirements. The *coverage* construct allows to define dependencies between quality requirements allowing the identification of possible reuse points for evidence collection.

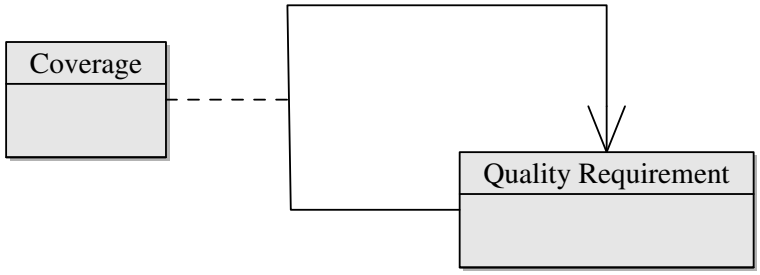

Figure 4.2: Coverage between Quality Requirements

The second consideration states the self-association - **coverage** - in Figure 4.2 is not reflexive. This is a limitation resulting from how the mapping is performed. When relating a quality requirement to a second quality requirement from a different model and analysing the degree of similarity of the intended output, one needs to consider that one quality requirement is fully implemented and compare how it relates to the mapped quality requirement, *e.g.*, when comparing a CMMI-Dev specific goal purpose and expected outputs, one may assert that, if fully implemented, it can be used as evidence to satisfy an ISO9001 shall statement. In this case, CMMI-Dev assumes the role of reference model and ISO9001 as the mapped model. The degree of similarity is characterized as the amount of reuse of the output of the CMMI-Dev implemented practice is expected to provide to satisfy the compared ISO9001 shall statement.

When comparing quality requirements to identify shared scope, the possible scenarios are identified: in Figure 4.1, the upper left Venn diagram shows how a mapped requirement (transparent circle) can be partially (70 out of 100) covered by using a

subset of the outcome of a reference quality requirement (shaded circle). The second from the upper left represents an example where full coverage is attained but the reference requirement can be said to be more extensive in the scope it defines. The upper third diagram represents an example where the comparison can be considered reflexive; the scopes are similar and the outcomes are similar. Therefore, *à priori* association between two requirements cannot be considered bi-directional. The last Venn diagram identifies the scenario where no scope is shared between quality requirements under comparison.

When performing mappings to support a joint audit and/or assessments, choosing a quality model that provides the most detailed and most alighted quality requirements with organizational business needs may be considered a logical decision, e.g., a software company may consider CMMI-Dev as the reference model and ISO9001 and ISO12207 as secondary models. Thus, the mapping should be established using as reference model CMMI-Dev and ISO9001 and ISO12207 as mapped models.

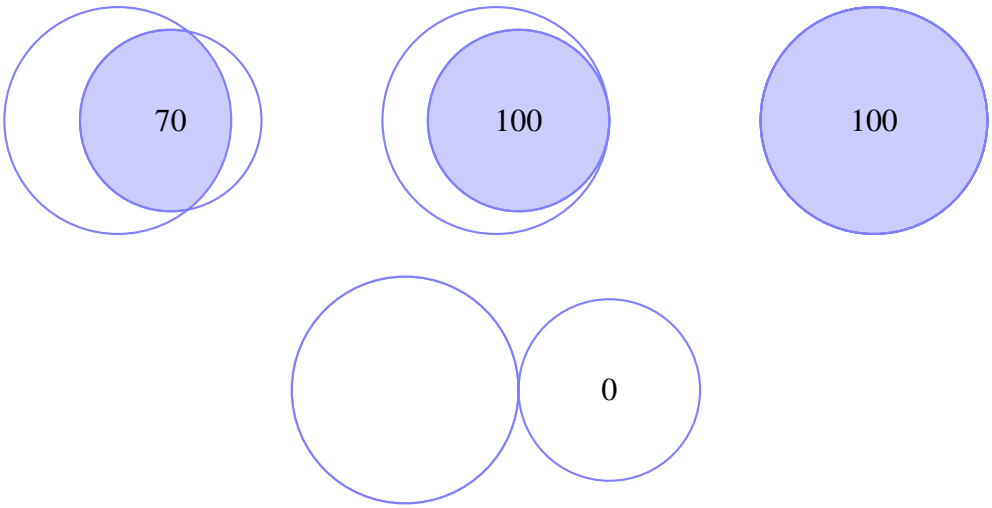

Figure 4.3: Quality requirements mappings scenarios

Traceability between quality requirements and implemented practices

Audits and assessments require objective evidence to establish conformance of implemented practices with reference standards, regulations, plans, specifications and capability frameworks and other relevant reference guidelines.

Objective evidence is any result or by product of implementation or institutionalization of practices. Objective evidence is mentioned in ISO1028 IEEE Standard for Software Reviews and Audits [IEE04], Standard CMMI Appraisal Method for Process

Improvement [Tea11] and ISO15504-2 - Process assessment [ISO04a] to represent any relevant work product that may be used to evaluate compliance.

Quality requirements provide guidance for defining and implementing needed organizational practices. Also, quality requirements from different models may be compared by relating their purpose and expected outcomes and characterize them according their degree of similarity. It is now relevant to detail how implemented practices can be linked back to quality requirements.

According to IEEE 1028 - Standard for Software Reviews and Audits the purpose of a software audit is to provide an independent evaluation of conformance of software products and processes to applicable regulations, standards, guidelines, plans, specifications, and procedures [IEE04]. Concerning evidence collection for evaluation purposes, the standard makes reference to interviews, examination of documents and witnessing processes as means to gather *objective evidence* of non-conformance or exemplary conformance. Audit observations are documented based on *objective evidence* and are classified as major or minor. It does not provide any detail on how and where objective evidence should be looked for.

According to ISO15504 [ISO04a] process assessments have two primary contexts for their use: *process improvement* and process *capability determination*. Process assessments aim to find strengths, weaknesses and risks inherent to processes providing drivers for improvement of processes. *Process capability* is determined by analysing organizational processes against a capability profile. A capability profile is based on a measurement framework that defines a set of attributes that characterize the capability of a process to fulfil its goals.

Three entities are relevant in performing process assessments:

- A measurement framework provides the capability profile and is used to derive a capability rating.
- A process reference model or models e.g., CMMI-Dev or ISO12207, provide the necessary process descriptions that will be used as frame of reference for organizational practices capability determination.
- An assessment model defines elements to relate processes of the process reference model(s) chosen as reference and the measurement framework process attributes to produce a capability rating. According to ISO15504-5 - An exemplar Process Assessment Model, elements of the assessment model can be indicators of performance and capability.

A process assessment model forms a basis for the collection of evidence and rating of process capability. It requires establishing a mapping between organizational processes to be assessed and the process reference model(s) process definitions [ISO04b].

ISO15504-2 refers to the suitability of a process model as a function of the degree of focus of assessment model indicators on observable aspects of process enactment and the assessment model degree of alignment with relevant process reference model [ISO06].

An *appraisal* is defined as an examination of one or more processes using as reference an appraisal reference model as a basis for determining, as a minimum, strengths and weaknesses [Tea11]. It can be considered a type of assessment if it is performed internally by the organization. One underpin of the Standard CMMI Appraisal Method for Process Improvement (SCAMPI) appraisal method is the link between CMMI-Dev process goals and implemented organizational practices. Goals are satisfied by gathering objective evidence of each generic and specific practice implementation. Objective evidence is expected to be from different types *e.g.*, *oral affirmations* must be collected to support objective evidence concerning practices implementation. The SCAMPI method defines the concept of practice implementation indicator to support the evaluation of practices implementation. A practice is considered implemented when direct artefacts, indirect artefacts and affirmation are gathered that provide substantiate evidence of practices implementation. *Direct* and *indirect* artefacts can be documents and *affirmations* are oral or written statements that result from interviews, presentations or demonstrations.

Based on the analysis of approaches for audits and assessments the concept of indicator and the concept objective evidence assume a central importance. Indicators are an abstract representation to group objective evidence of organizational practices implementation and establish the association between performed processes and measurement attributes when a capability assessment is to be performed.

Also, affirmations are obtained manly from interviews and are required to substantiate and provide objective evidence of implemented practices. Based on this highlighted concepts, the next section elaborates on a model that relates relevant entities in support of multi-model audits and assessments.

Modelling compliance in multi-model environments

Quality requirements of different models can be related by the amount of shared scope. Purpose and expected outcomes are compared to characterize their degree of similarity. Quality requirements also provide motivation and guidance to define organizational practices and are interpreted considering organizational context and needs. Practices for achieving desired business goals are defined with guidance set by quality requirements adopted.

In the context of multi-model environments, performing an evaluation of areas of concern related to different quality models in a single audit or assessment can reduce

costs and improve efficiency of audits and assessments *e.g.*, in a single exercise evaluate process compliance to ISO12207 and CMMI-Dev by reusing collected evidence.

In order to reuse collected evidence one needs to identify which artefacts are shared among different quality requirements. This is possible by defining maps between organizational practices and quality requirements. With the maps established one can list artefacts relevant to a specific quality requirement implementation. By considering coverage associations between quality requirements it is possible to identify which artefacts can be shared among related quality requirements.

The model in Figure 4.4 introduces relevant entities and how these relate to each other in support of audits and assessments on multi-model environment.

- **QualityRequirement**: *QualityRequirement* is associated to zero or more entities of different origin, e.g., one can map an ISO9001 shall statement to several CMMI-Dev specific practices.

- **Coverage**: the association between quality requirements is characterized by *Coverage*, that defines the degree of shared scope between *QualityRequirement* instances. As an example, in a mapping between ISO9001 and CMMI-Dev, an ISO shall statement, *Establish QMS,* maps to 29 specific practices of CMMI-Dev with different coverage values, defined by a scale of comparison that can have values of 0, 30, 60, 100.

- **Artefact**: *Artefact* refers to any tangible process element used to describe or maintain process related information, *e.g.*, an artefact can be a *Work Product Definition*, *Task Definition* and other process constructs if *e.g.*, Software and Systems Process Engineering Metamodel Specification is used as a modelling language to define an OSSP.

- **Indicator**: an *indicator* is used to group relevant process related artefacts, defined in the OSSP, which are expected to provide objective evidence of quality requirements implementation. This step requires that a mapping between artefacts and related quality requirements is established, e.g., in support of specific practices of CMMI-Dev process areas, a set of relevant work products and task descriptions are identified that are expected to provide evidence of practice implementation, when these are enacted by project or organizational units. With mappings established between OSSP artefacts and *Quality Requirements* along with *Coverage* associations, *Artefacts* used as evidence for a quality requirement implementation can be reused as evidence for mapped quality requirements, e.g., artefacts associated with CMMI-Dev specific practices implementation can be reused to provide objective evidence of ISO9001 shall statements which are

mapped to CMMI-Dev specific practices.

- **Affirmation**: both audit and assessment standards make reference to the need of having supporting oral evidences of performed practices from practice implementers. The element *Affirmation* is associated to *Artefact* to emphasize that artefacts require oral or written statements as supporting objective evidence. An affirmation is a type of objective evidence to confirm artefact related evidence. An *Affirmation* instance is expected mainly as result of interviews when assessments and audits are performed.

- **Scope**: the element *Scope* makes explicit the notion that audit and assessment may be performed with different goals in mind. A *Scope* instance has always an associated *Indicator* instance that is always associated to a *QualityRequirement* instance. By choosing relevant quality requirements from different quality models, associated indicators are automatically identifiable, whether these are obtained directly by the *Indicator/QualityRequirement* association or indirectly by the *Coverage* association defined between *QualityRequirement* instances. An *Affirmation* instance can be associated to multiple scopes. This allows defining different affirmation instances related to a same artefact, providing flexibility in defining different elements to support collection of oral or written statements for different scope scenarios.

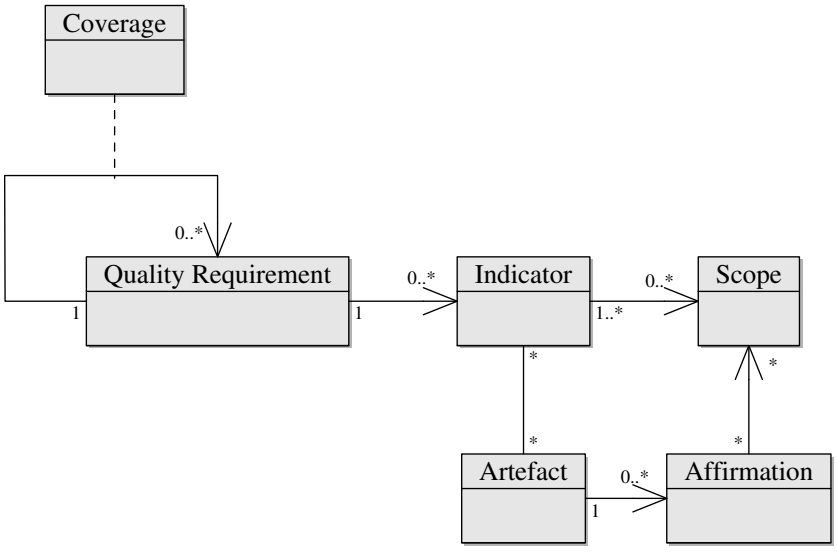

Figure 4.4: A model for compliance in multi-model environments

Figure 4.5 depicts an example of a generic joint multi-model audit scenario based in the perspective of the model proposed in Figure 4.4. The two left columns depict

six *Quality Requirements* considered for a desired scope. The first column represents mapped Quality Requirements and the second column *Quality Requirements* from the reference model. *Coverage* associations are established between selected quality requirements. The quality requirements in the second column have associated *Indicators* defined. Each indicator results from identifying relevant *Artefacts* that are expected to provide objective evidence of implemented practices. Indicators are represented in the third column containing relevant artefacts. It is possible to verify that different indicators may reuse artefacts as evidence for different quality requirements implementation. This is possible as it depends how practices and expected outcomes are organized in the organizational environment.

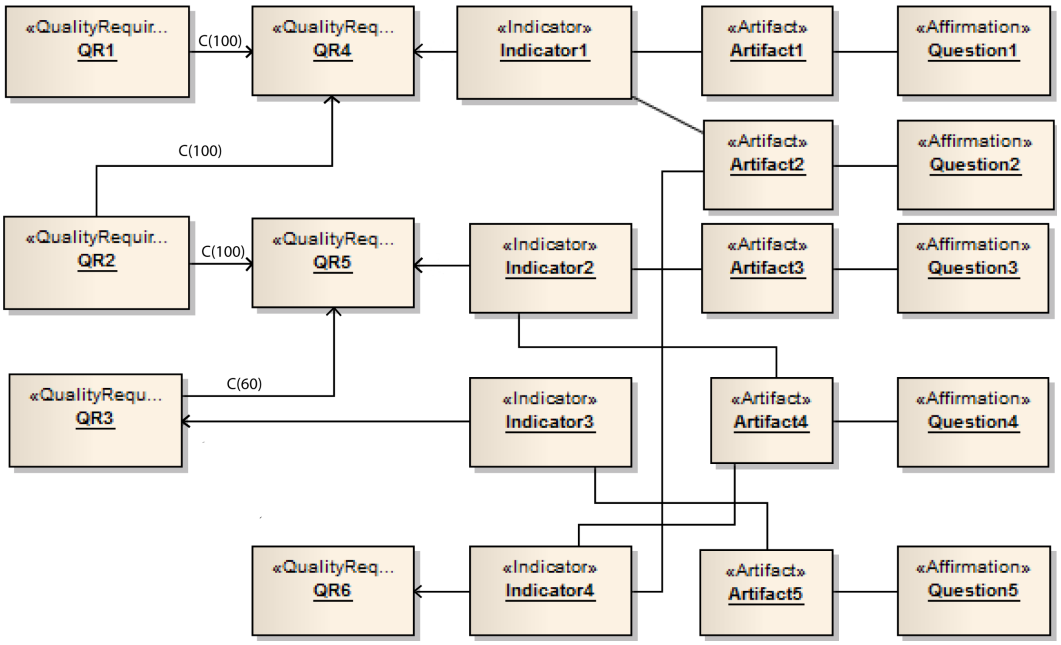

Figure 4.5: Generic joint audit and/or assessment scenario

From the mappings defined between quality requirements it's possible to identify *QR(1)*, *QR(2)* and *QR(3)* from the mapped model have coverage associations defined to *QR(4)* and *QR(5)* of the reference model, respectively.

Five coverage associations are defined with values of *C(100)*, *C(100)*, *C(100)* and *C(60)*. *QR(1)* and *QR(2)* can reuse artefacts from indicators *I1* and *I2* of *QR(4)* and *QR(5)* respectively. *QR(3)* has only a portion of reused scope with *QR(5)* and requires an indicator, *I3* that identifies the set of artefacts to assure full compliance coverage of *QR(3)*.

The fifth column represents affirmations instances associated to artefacts for each indicator. *Questions* are a form of *Affirmations* that can be defined and maintained by organizational process improvement groups to use in collecting required oral or written

statements as support objective evidence, *e.g.*, obtain a confirmation statement that a work product or activity description is implemented as expected.

In support of software audits and assessments, the model in Figure 4.4 can be used to define multiple audit and assessment scenarios, *e.g.*, in performing project or process audits one may define different perspectives of evaluation and chose different scopes for each type evaluation. The scope of the audit is defined by identifying relevant indicators associated to quality requirements from multiple quality models. Questions can be maintained as instances of affirmation which can be associated to multiple different scopes.

In the specific context of assessments and using as example the assessment model proposed in ISO15504-5 [ISO06] an assessment model indicator is refined into performance and capability indicators. The model doe not include this level of refinement by considering that the specialization is associated to the method defined for the assessment model. By considering solely the concept of indicator we leave the possibility of extending the concept of indicator to support possible different assessment methods, e.g., by considering different measurement frameworks and associated capability indicators.

4.4 Multi-model process assessment demonstration case

This section demonstrates how the conceptual model presented in the previous section can be used to support a joint assessment exercise. An organizational scenario is defined to detail how it can be applied to perform an joint assessment. For this purpose we will consider a multi-model environment considering ISO12207, ISO9001 and CMMI-Dev as improvement models in the same organizational environment and a formal definition of organizational processes exists.

The assessment can be put into a context of organizational Quality Management activities performed in the scope of an Audit Process. As result of Quality Management activities and the Audit Process, one can expect an analysis to identify possible similarities between adopted improvement models and then proceed to identifying which OSSP process assets can be used to determine compliance.

Setting up the multi-model compliance Information System

Figure 4.6 depicts a possible example of Quality Management tasks using SPEM 2.0 notation, namely: *Model Mapping* and *OSSP and QR mapping* tasks.

Model Mapping task

The output of the *Model Mapping* task are mapping tables where quality requirements from different quality models are mapped and a coverage function is used to characterize their relationship. To perform this task one needs to determine one of the improvement models as the reference model. By choosing a reference improvement model the direction of the relationship for the mapping function is determined. As an example CMMI-Dev will be considered as the reference model and ISO9001 and ISO12207 as the mapped improvement models. The mappings between ISO12207 and ISO9001 to CMMI-Dev created by [MSS09a, MS09] are output examples of the *Model Mapping* task for this type of scenario.

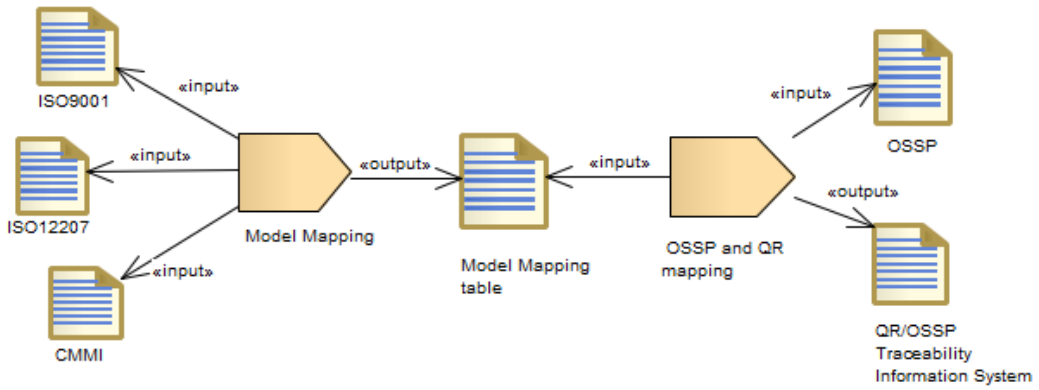

Figure 4.6: Quality Management Process

OSSP and QR mapping task

The mapping tables are used as input to *OSSP and QR mapping* task along with organizational process definitions. The expected output is an information system based on the conceptual model described in Figure 4.4. The *OSSP and QR mapping* task may include the following steps:

1. The mapping tables are used to create *QualityRequirement* and *Coverage* instances (see Figure 4.4). In a first step, all specific practices of CMMI-Dev, activities and tasks of ISO12207 and shall statements of ISO9001 originate QualityRequirement instances. Coverage associations are created based on the Model Mapping tables output of Model Mapping task.

2. In a second step, for all instances of *QualityRequirement* of the reference improvement model, an *Indicator* instance is defined by identifying relevant *Artefacts* in the OSSP, e.g., if SPEM is used as process modelling language to define the OSSP, SPEM instances of constructs like *Task Definition*, *WorkProduct Definition*, *Activity*, among others, can be used as instances of type *Artefact* (see Figure 4.4) to define Indicators for each specific practice of CMMI-Dev.

3. In a third step, the remaining *QualityRequirement* instances not belonging to the reference model require an analysis to evaluate if *QualityRequirement* related instances (resulting of the *Coverage* instance defined) do include all relevant artefacts in the OSSP that can be useful in supporting desired compliance. This is a vital point in the process of mapping quality requirements with OSSP process entities. For the *QualityRequirement* not part of the reference model, not having Indicators associated, they will reuse *Indicators* of mapped quality requirements. This means that if a coverage association exists it will identify which indicators can be reused for the purpose of assessing compliance. This is where the system takes full advantage of shared scope between adopted models. Even so, if mapped quality requirements do not provide full coverage, additional indicator instances need to be defined identifying missing relevant OSSP artefacts, e.g., if an ISO12207 activity is partially covered (e.g., not having as maximum a value of 100) by a related CMMI-Dev specific practices, it will reuse artefacts identified by the indicators associated to CMMI-Dev specific practices and still require extra artefacts to support full compliance for the activity considered. An example is given in Figure 4.4 by the *QR(3)*. It maps partially to *QR(5)* and thus reuses *Indicator2* related *Artefacts* but requires the creation of *Indicator3* to fully collect evidences needed to achieve full coverage of the requirement.

As an outcome of *OSSP and QR mapping* task an *information system* relating improvement models adopted and the respective OSSP entities relevant to ensure compliance is created including indicatoers used to establish compliance with relevant scopes.

Figure 4.7 depicts an example (not exhaustive) of a *QR/OSSP Traceability Information System* describing associations on shared scope between quality requirements and OSSP process related entities. The first and second columns represent quality requirements from ISO9001 and ISO12207 respectively, along with their coverage association with CMMI-Dev practices (depicted in the arrows linking the Quality Requirements instances), represented in the third column.

Compliance assessment

Figure 4.8 depicts two tasks in the scope of an Audit process (aligned with IEEE 1028 standard) to demonstrate the context where an information system of figure 4.7 can be used to support the joint model audit process. As an example we will consider an assessment scope set to evaluate compliance to CMMI-Dev Configuration Management process area. This is depicted in the fourth column of Figure 4.7 along with the Indicators associated to the scope for Configuration Management.

The example of Figure 4.7 is created in line with the mapping tables defined by Mutafelija and Stromberg [MSS09a, MS09] that maps ISO9001 and ISO12207 to

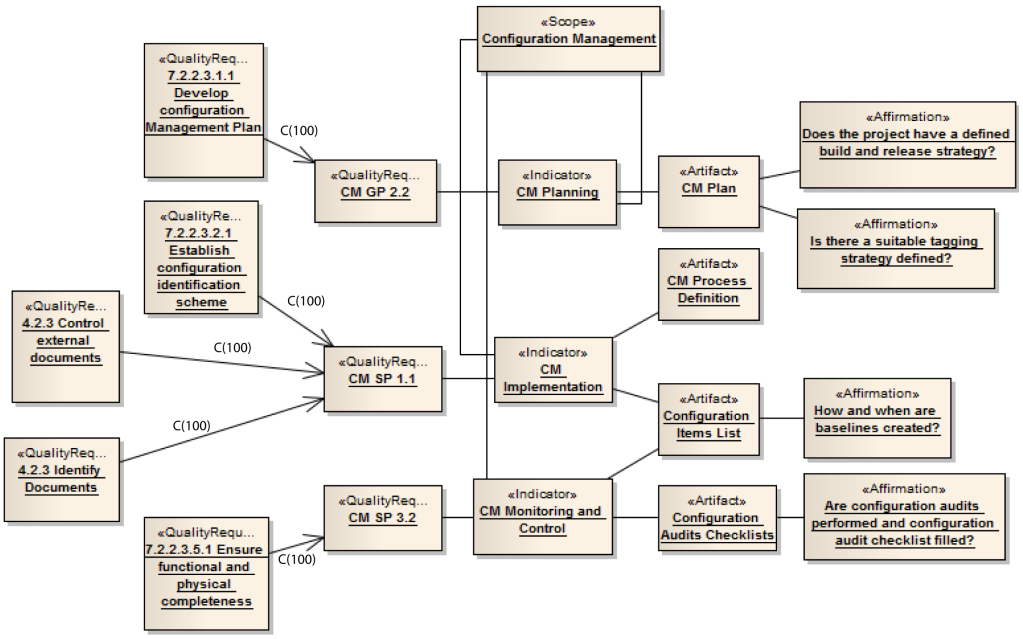

Figure 4.7: Compliance Information System

CMMI-Dev respectively. The mappings provide an example output of the task *Model Mapping table* in Figure 4.6. By analysing the mapping table all ISO12207 configuration management activities and tasks have full coverage by CMMI-Dev Configuration Management specific practices, thus Indicators defined to support CMMI-Dev configuration management evidences can be reused to guide data collection for compliance with ISO12207 configuration management process. By analysing mapping tables mentioned, a similar example can be considered relating Decision Management process of ISO12207 and Decision and Analysis Resolution from CMMI-Dev, the coverage association established allow a full reuse of evidences collection.

If different scopes are defined for Configuration Management and Decision and Analysis Resolution/Decision Management processes from CMMI-Dev and ISO12207 an audit check-list template for evidence collection can be defined in the *Planning the audit task* depicted in Figure 4.8.

The check-list is defined by selecting from the information system the *Indicators* associated to the desired scopes e.g., Configuration Management and Decision and Analysis Resolution.

To further optimise the audit process, questions can be defined for a specific scope to support gathering objective evidence on performed practices. These questions become instances of *Affirmation* and are associated to *Artefacts* of the OSSP and to a *Scope* defined for the audit. The association exists to allow defining different questions for different scopes. By maintaining information linking *Scope* and *Affirmation*

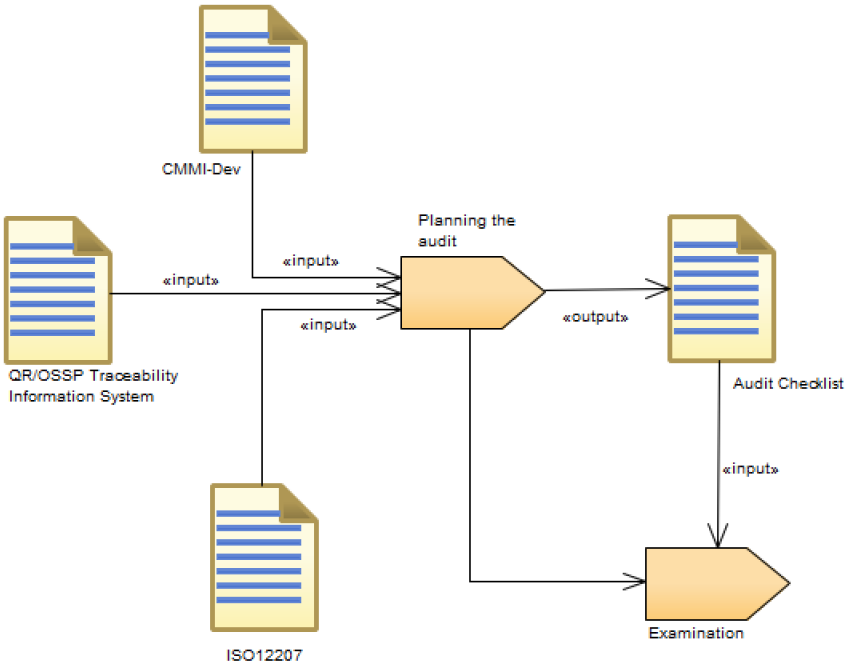

Figure 4.8: Audit Process

instances, it is possible to support automatic generation of data collection check-list in support of multi-model audits and/or assessments.

4.5 Analysis and discussion of results

In the previous section a demonstration of how a joint assessment can be performed reusing most of the effort of evidence collection by highlight an example where full reuse is possible. In the previous chapter a quantitative evaluation is performed based on the mappings considered for this example concluded that ISO9001 and ISO12207 share with CMMI-Dev 83% and 74% of their scope respectively. This provides a measure on the amount of effort that could be saved when performing full compliance evaluations for multiple quality models.

The conceptual model is designed to organize information concerning shared scope between quality requirements and how, in the presence of OSSP formal definition, process related artefacts are being used in support of quality requirements implementation. With the resulting information system, one can improve quality management activities related to multi-model environments by:

- Precisely identify which OSSP related artefacts are involved in quality requirements implementation.

- Define indicators that aggregate information related to shared scope between quality requirements from different quality models.
- Identify which quality requirements are affected by possible changes in process related artefacts motivated by process evolution.
- Identify which organizational practices can be dismissed by dropping specific quality requirements
- In the context of a project enactment system based on OSSP definitions, project audits and capability assessments can be partially automated. This is possible by monitoring process enactment by collecting objective evidence of performed practices.

4.6 Conclusions

In this chapter a conceptual model to support management of information related to quality requirements of multiple improvement technologies was presented. The goal was to support audits and assessments of multiple improvement models in a single effort. The underlining motivation is that different improvement models often share scope of concern providing the opportunity to reuse evidences of organizational practices implementation in evaluating compliance to multiple quality models.

The proposed conceptual model is based on the model mapping technique and in the concept of indicator. The mapping is used to compare quality requirements from different improvement models and to evaluate their degree of similarity concerning purpose and expected outcomes. The concept of indicator is used to group organizational process entities involved in quality requirements implementation, which can be used to guide the collection of objective evidence of their execution.

The model aims to help improve organizational quality management capability in managing multi-model internal audits and assessments. An external multi-model certification scheme is yet impossible as certification bodies do not acknowledge certifications from other certification bodies.

A demonstration of using the concepts introduced in this chapter is provided to exemplify how the model can be useful in providing support to a joint process audit considering ISO9001, ISO12207 and CMMI-Dev. The example documents, how in one single audit iteration one can assess compliance of Configuration Management process area of CMMI-Dev and ISO12207 in a single iteration.

Chapter 5

CMMI-Dev High Maturity and Analytical Process Improvement

Contents

This chapter discusses the similarities of analytical approaches to process improvement and CMMI-Dev high maturity specific goals. The motivation is to analyse to which extent analytical practices are supportive of CMMI-Dev high maturity process goals and propose an approach to implement a process for software process improvement. A comparison using the bilateral mapping method is described and a process improvement process is proposed based on the experience of implementing software process improvement initiatives at CRITICAL Software S.A.

5.1 Analytical and benchmarked based SPI

As detailed in Chapter 2 there are two fundamental approaches to SPI, the benchmark approach and the analytical approach. The benchmark based approach believes that organizations can improve their development process by adopting practices from top performing organizations. These practices are documented and made available in quality models or standards and organizations adopt one or more of these models as a strategy to improve their performance. On the other hand, analytical approaches require that improvement efforts are driven by a quantitative understanding of current organizational performance. Having this quantitative understanding is also useful to validate if improvement efforts were successful at the end of process improvement initiatives. Also, analytical approaches all share the notion of improvement cycles or iterations. Improvement is perceived as a continuous effort that seeks to improve the operation of the organization to better meet business and performance goals. This somehow contrasts with the benchmarked based approaches, which rely on bridging the gaps between organizational practices and prescribed practices, which can be achieved in a one time improvement effort.

CMMI-Dev is a benchmarked based approach that describes best practices for developing quality products and services. It's a prescriptive model in nature and defines a set of goals that must be met by the adopting organization. Considering the staged representation of the model the principles of the analytical approaches are in-line with the practices associated with Level 4 and Level 5 maturity levels (also called high maturity practices). Although benchmarked and analytical are distinct paradigms we will argue that CMMI-Dev prescribes a set of goals and associated practices that can be satisfied by adopting the analytical paradigm for SPI.

Benefits and drawbacks of benchmarked based approaches were discussed in Chapter 2. In summary benchmarked based approaches become very popular supported by considerable number of studies documenting the benefits resulting from their adoption. However, major critiques are that quality is self-fulfilling, meaning an organization can implement the practices prescribed by benchmarked based approaches and do not achieve actual improvements. Also, benchmarked based models seem inadequate to small to medium organizations, as they do not provide adequate guidance on their implementation and require considerable resources and/or knowledge that are scarce in small settings.

CMMI-Dev high maturity practices have also been the centre of considerable debate. Organizations in their journey from maturity Levels 2 through 3 improve in preventing disasters and gain a greater control in the way work is performed. At maturity level 4 and 5, organizations manage against identified quality and process performance objectives to meet business objectives. Maturity levels 4 and 5 require the use of quan-

titative techniques that demand greatly on measurement related activities. Going for high levels of maturity requires further investment on management and support activities (all level 5 practices are associated with Management and Support CMMI process categories). A cost *vs.* benefit analysis raises the issue of the need or benefits to go for high maturity levels, specially when costumer satisfaction apparently did not rise in parallel with maturity level ratings [Cam12]. The recent revision and upgrade of CMMI-Dev version from 1.2 to 1.3 focused on high maturity practices, indicating that high maturity practices is still evolving.

There are only few industry reports that describe the benefits and recommendations of operating at CMMI-Dev high level of maturity. The benefits are typically reported using specific metrics relevant to the organization, e.g, profit per employee, number of defects in production per requirement and number of requirements per release [FSM14]. Campo [Cam12] reported return on investment in high maturity of 38.4:1, and investment was defined as the cost of all activities to incorporate and appraise maturity Level 4 and 5 practices into organizational processes. Girish [Ses12] reported high maturity practices improved the ability to predict the likelihood of a software component to have defects during testing phases and showed how schedule and effort deviation improved as result of high maturity practices. These experience reports provide quantitative evidences of how investment in high maturity can deliver benefits.

Although, in the end of January 2016 a total of 384 organizations are listed as successfully appraised for CMMI-Dev 1.3 Maturity Level 5. As a reference a total 3367 organizations were appraised successfully for Level 3 [Ins15]. These numbers show high maturity practices seem to be a trait for a reduced set of organizations even when CMMI Institute reports in 2014, a 12 percent increase in CMMI appraisals since 2012.

Motivation and research goal

CMMI-Dev high maturity practices are still a trait of few organizations dedicated to software engineering. Also, these practices may not be delivering the expected benefits and recent updates to the CMMI-Dev model focused on the high maturity process areas. An understanding of how these practices can be implemented is therefore valuable to the software engineering community.

This chapter documents a detailed analysis of analytical approaches to SPI in the perspective of CMMI-Dev high maturity goals. A first objective is to demonstrate that both improvement paradigms share the similar goal of driving process improvement against process performance objectives using quantitative techniques and answer the following question:

To what extent the analytical paradigm to software process improvement can support organizations in achieving CMMI-Dev high maturity requirements?

Based on the results of the comparison and the empirical knowledge resulting from implementing improvement projects at CRITICAL Software S.A. over a period of 5 years, an improvement process is proposed to guide software process improvement initiatives.

5.2 CMMI-Dev high maturity and Analytical SPI

By definition, requirements for a CMMI-Dev maturity level 5 organization are much in line with the principles of analytical approaches to SPI. A mapping exercise will be used to compare both approaches and clarify and support this argument, and provide the base for definition of a methodology to SPI in line with CMMI-Dev maturity level 5 requirements.

At CMMI-Dev maturity level 5, by definition, an organization focuses on continually improving process performance through incremental and innovative process and technological improvements. Additionally, improvement initiatives are driven by a quantitative understanding of business objectives and performance needs. The expectation for a level 5 organization aligns with the purpose and goals of analytical approaches to SPI in the sense that analytical approaches require that organizational weakness with respect to improvement or business goals are identified to drive SPI efforts. This relates to the *'quantitative understanding of business objectives and performance needs'* requirement set by CMMI-Dev. Also, the term, *continuously*, relates to the iterative or cyclic principle of analytical approaches of continuous improvement.

The comparison method used for the mapping exercise between CMMI-Dev and analytical approaches to SPI is the *bilateral mapping* introduced in Chapter 2, section 2.3. In bilateral mappings the comparison can be created having two improvement models and taking a point of view of a model to describe the second model in terms of the first, resulting in a textual description of the analysis. To determine the scope of the comparison, a preliminary analysis of the CMMI-Dev model and the descriptions of the analytical approaches was performed to identify which architectural elements of the models best fit the mapping. From the preliminary analysis the decision was to use CMMI-Dev as the reference model and the strategies for analytical SPI as the mapping models. This means mapping will be described taking the point of view of the analytical approaches to SPI and having as a reference the CMMI-Dev high maturity practices. Figure 5.1 presents an overview of the mappings performed.

The outcome expected is a detailed analysis of how CMMI-Dev high maturity practices relate to analytical approaches to SPI and understand which CMMI-Dev high maturity practices can be implemented by adopting an analytical paradigm to SPI, e.g., the outcome of a mapping is expected to clarify the following question: *'What level of alignment with CMMI-Dev high maturity process areas can an organization expect when implementing DMAIC?*'. This information is valuable to provide clarification on what extent a benchmarked based SPI approach like CMMI-Dev embeds some or all of the principles of the analytical paradigm.

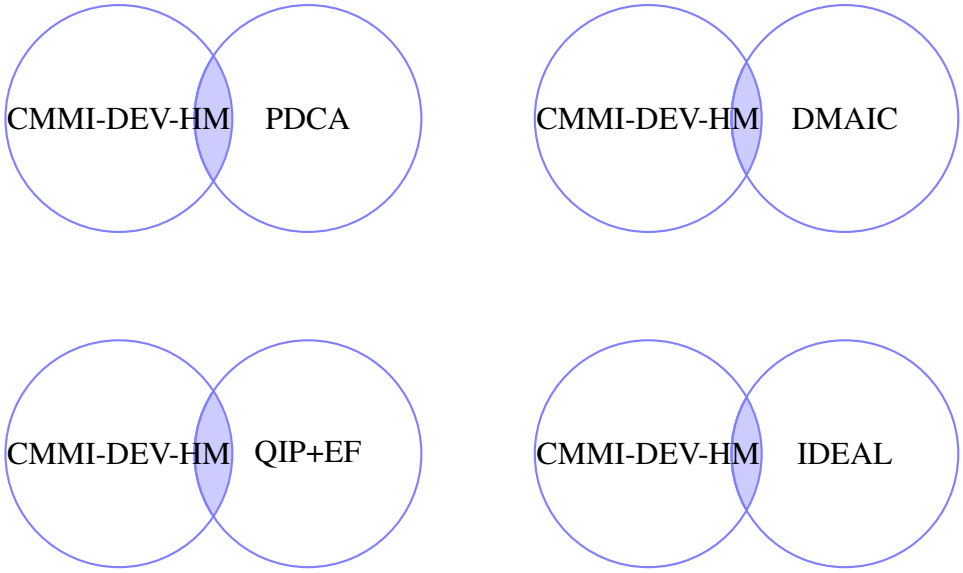

Figure 5.1: Bilateral mappings

5.2.1 Bilateral mappings

To perform a mapping exercise between two models one needs to define the scope and the level of detail for the comparison. The scope defines which model elements or components are to be used in the comparison. Components are the main architectural elements that compose the CMMI-Dev model. Some of the main elements of a CMMI-Dev model include *specific practices*, *generic practices*, *specific goals*, *generic goals*, *process areas*, *capability levels*, and *maturity levels*. Benchmarked based approaches are typically organised in a hierarchical form having their practices or requirements defined in tree like form. In the case of CMMI-Dev, Figure 5.2 details the relation between model components. At lower hierarchical levels e.g., *Example Work Products*, the level of information or detail used to describe a quality requirement is higher when compared to e.g., *Process Area*. Descriptions used by analytical approaches are less

structured, except for the IDEAL model. The notion of hierarchical refinement of quality requirements is not as tangible as in CMMI-Dev, however the notion of phase, along with textual descriptions is shared among all analytical approaches.

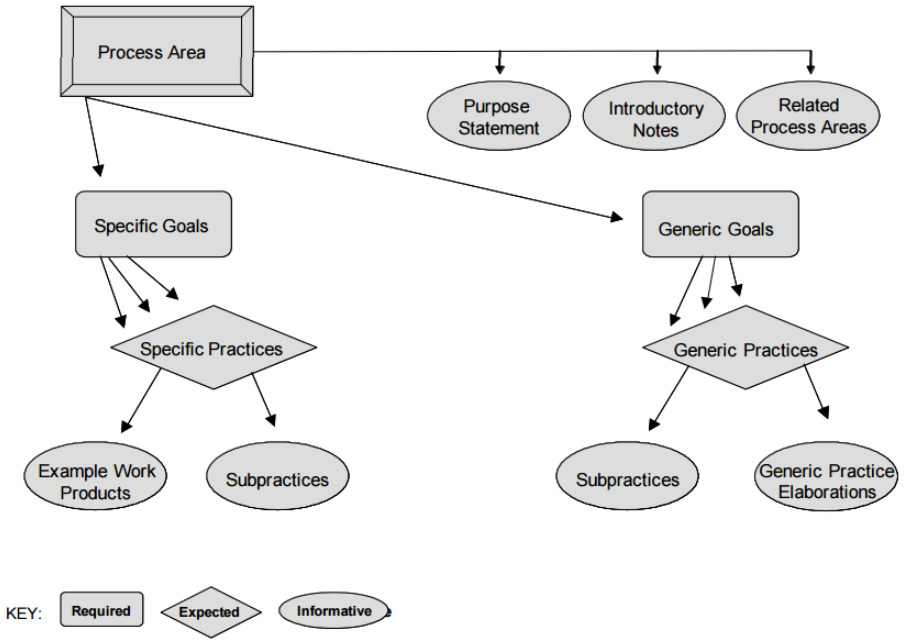

Figure 5.2: CMMI-DEV model components (in [CKS11])

Based on this rationale, the scope of the bilateral mappings was set to compare analytical approaches at a *phase/step* level considering their textual content with CMMI-Dev at the *Specific Goal* level for each *Process Area* included in maturity level 4 and 5. A practical result of the bilateral mapping will identify a set of triples, comprising the analytical approach phase/step, the corresponding Process Area/Specific Goal of CMMI-Dev and a measure indicative of how strong is the relation between the elements. Along with the triple, a corresponding detailed analysis and justification of the mapping is given. The measure used is qualitative in nature consisting of four values, defined in Table 5.1.

When the result of the mapping between a Phase/Step is *No Mapping* the result will not be depicted for ease of readability of the mapping. Only when there is mapping equal or above *Low* the result will be identified. Additionally, for computing the final mapping result between Phase/Step of an analytical approach and CMMI-Dev the highest mapping classification is taken. As an example, if a Phase/Step is classified *Low* in relation to a CMMI-Dev specific goal and a second Phase/Step also relates to the same CMMI-Dev specific goal with *High*, the high relation is taken for defining the relation between the Phase/Step and the specific goal of CMMI-Dev.

Metric	Rationale
No Mapping (NA)	The analytical approach phase does not relate to any the CMMI-Dev goal or practice. There is **no clear evidence** by analysing the description of a phase to conclude that when implemented will support the achievement of a any CMMI-Dev Specific Practices belonging to the corresponding Specific Goals.
Low (L)	The analytical approach phase provides information indicating that when implemented only a **minimal set** of CMMI-Dev Specific Practices within a Specific Goals can be met.
Medium (M)	The analytical approach phase provides information indicating that when implemented, at **least an half** of CMMI-Dev Specific Practices belonging to a Specific Goals can be met.
High (H)	The analytical approach phase provides information indicating that when implemented **more then a third** of CMMI-Dev Specific Practices belonging to a Specific Goals can be met.

Table 5.1: Bilateral mapping comparing measures

5.2.2 PDCA vs CMMI-Dev High Maturity

The Plan-Do-Act-Check cycle is a continuous improvement strategy that aims to guide the improvement effort of a process or a product. As described in chapter 2 the PDCA cycle is based on the Shewhart cycle and is part of the 14 principles for transformation of management. The PDCA cycle is part of the 14th principle, *'Take action to accomplish the transformation'* to help an organization to establish a focus on a continuous improvement of quality. The Shewhart cycle was adapted by Demming to originate the Plan-Do-Study-Act (PDSA) cycle [Dem00a].

Plan step

The PDSA cycle begins with **Plan** step were someone identifies an idea for improvement leading to a plan for a test, comparison or an experiment. Several suggestions of improvement should be identified and possible outcomes considered and compared. The Plan step starts when a choice between several suggestions is made and an actual plan is defined. This description for the Plan step aligns with *CMMI-Dev OPM.SG*

1 Manage Business Performance specific goal, namely in the scope of the specific practice *OPM.SP 1.3 Identify Potential Areas for Improvement* where possible improvements areas are identified within the organization. The mapping is classified as *Low* as only one out of three Specific Practice (SP) is related to the *Plan* step. Also, the Plan step aligns with *OPM.SG 2 Select Improvements* specific goal in the scope of *OPM.SP 2.1 Elicit Suggested Improvements* and *OPM.SP 2.2 Analyse Suggested Improvements* specific practices. The expected work products of these practices are suggested innovative and incremental improvements and actual proposals are selected for implementation based on these improvements. The mapping of the *Plan* step with OPM.SG 2 is classified as *Medium* as two out of three of the practices relate to the *Plan* step. Table 5.2 summarizes the mapping for this step.

Component	**Measure**	**CMMI-Dev Specific Goal**
Plan	Low	OPM.SG 1 Manage Business Performance.
	Medium	OPM.SG 2 Select Improvements.

Table 5.2: *Plan* step mapping to CMMI-Dev practices

Do step

In the **Do** step a test, experiment or comparison is carried out, preferably on a small scale. This description maps to CMMI-Dev *OPM.SG 2 Select Improvements* specific goal, namely *OPM.SP 2.4 Select and Implement Improvements for Deployment* specific practice where improvements are selected and changes to the process are implemented. The mapping is classified as *Low* as one out of four practices may relates to the *Do* step. Table 5.3 summarizes the mapping of the Do step with CMMI-Dev high maturity practices.

Component	**Measure**	**CMMI-Dev Specific Goal**
Do	Low	OPM.SG 2 Select Improvements.

Table 5.3: *Do* step mapping to CMMI-Dev practices

Study step

In the **Study** step the results of the test, comparison or experiment are analysed to evaluate if they are in line with improvement expectations. This evaluation is also expected

in CMMI-Dev *OPM.SG 2 Select Improvements* specific goal, namely in the scope of *OPM.SP 2.3 Validate Improvements*. In validating improvements, an organization is required to have an assessment on the results of the improvement proposal against initial expectations. CMMI-Dev also expects that statistical approaches are used in the assessment and these may include modelling and simulation of process behaviour. Other assessment methods may focus on formal reviews, prototype demonstrations and pilots of improvement. Table 5.4 summarizes the mapping.

Component	**Measure**	**CMMI-Dev Specific Goal**
Study	Low	OPM.SG 2 Select Improvements.

Table 5.4: *Study* step mapping to CMMI-Dev practices

Act step

The last step, **Act** the improvement is adopted or abandoned or a decision is made to run a new iteration of improvement. The case where the improvement is adopted successfully maps to the expectation for *OPM.SG 3 Deploy Improvements* more specifically in the scope of *OPM.SP 3.1 Plan the Deployment* and *OPM.SP 3.2 Manage the Deployment* specific practices. These involve developing a deployment plan of the improvement and monitor the deployment of improvements using deployment plans. The mapping is classified as *Medium* as two out of three practices are covered in CMMI-Dev OPM.SP 3.1 specific goal. Table 5.5 summarizes the mapping.

Component	**Measure**	**CMMI-Dev Specific Goal**
Act	Medium	OPM.SG 3 Deploy Improvements.

Table 5.5: *Act* step mapping to CMMI-Dev practices

The PDSA is the oldest analytical approach and is less detailed when compared with other analytical approaches. PDSA defines an iterative structure for identification, planning, experimentation, assessment and deployment of improvements. Table 5.6 summarizes the mapping between PDSA and CMMI-Dev providing overall evaluation of coverage considering the highest value of a mapping for each CMMI-Dev high maturity specific goals. Considering the highest mapping possible one can conclude that based on this assessment, organizations following PDSA address partially the Organizational Performance Management Process Area of CMMI-Dev.

	CAR.SG 1	CAR.SG 2	OPM.SG 1	OPM.SG 2	OPM.SG 3	OPP.SG 1	QPM.SG 1	QPM.SG 2
Plan	–	–	L	M	–	–	–	–
Do	–	–	–	L	–	–	–	–
Study	–	–	–	L	–	–	–	–
Check	–	–	–	–	L	–	–	–
Highest mapping	–	–	L	M	L	–	–	–

Table 5.6: Summary of PDSA vs CMMI-Dev High maturity mapping

5.2.3 QIP+EF vs CMMI-Dev High maturity

The QIP methodology was first detailed by Basili *et al.* [Bas85] in what was named as a *methodology improvement paradigm*. The methodology included six steps where the last step had an explicit reference to the first step, to define the cyclic characteristic of the improvement paradigm. Later the paradigm was described as a process model in the scope of the TAME (Tailoring A Measurement Environment) project [BR88]. In [BCR02a] the concept of Experience Factory was combined with the improvement paradigm originating the QIP+EF. The QIP process model used in the mapping with CMMI-Dev is the one described in the scope of the TAME project [BR88] and includes a set of five components, namely: *Characterize*, *Plan*, *Execute*, *The Experience Base* and *Learning and Feedback*. The mapping between QIP+EF and CMMI-Dev will consider these components and evaluate the coverage obtained in CMMI-Dev high maturity specific goals in a scenario of an organization adopting QIP+EF.

Characterize component

The ***Characterize*** component in QIP+EF involves identifying factors that influence the current project environment. Constructive and analytic aspects of the project environment are identified and analysed considering data data from previous projects. State of the practice, when available, is used in the analysis of the project environment. The characterize step purpose is to define a starting point for improvement. This step does not have have an explicit map to a goal in the high maturity landscape of CMMI-Dev. It somehow relates to the specific goal *QPM.SG 1 Prepare for Quantitative Management* (Table 5.7). The goal preparation for quantitative management involves establishing

project objectives for the project in the scope of the specific practice *QPM.SP 1.1 Establish the Project's Objectives* which is further detailed in a sub-practice that involves writing objectives that reflect the quality and process performance needs and priorities of the customer, end users, and other relevant stakeholders. The reasoning for the mapping (classified as Low) is that *needs* and *priorities* may relate to project factors that will impact the project environment, as defined by QIP+EF and only one out of four practices QPM.SG 1 is related to the characterize step.

Component	Measure	CMMI-Dev Specific Goal
Characterize	Low	QPM.SG 1 Prepare for Quantitative Management.

Table 5.7: *Characterize* component mapping to CMMI-Dev specific goals

The Plan component

The **Plan** component involves defining project goals, execution needs and project focus for learning. The plan step is divided in two major aspects, the *'what'* planning and the *'how'* planning. The *what* planning deals with choosing, assigning priorities and operationally defining the project goals. The *how* planning involves choosing the appropriate execution model, methods and tools that best suite the development needs. Execution involves both construction and analysis methods, the former relate to actual product development and the later are methods for evaluating if project goals are being met.

This step has a strong map to the *QPM.SG 1 Prepare for Quantitative Management* specific goal and the respective practices *QPM.SP 1.1 Establish the Project's Objectives*, *QPM.SP 1.2 Compose the Defined Process*, *QPM.SP 1.3 Select Sub-processes and Attributes* and *QPM.SP 1.4 Select Measures and Analytic Techniques* (Table 5.8). In the plan step one defines the project objectives and then define the best suitable execution model, methods and tools to meet the objectives. Additionally, metrics need to be defined and later collected based on the project goals identified to assess is project goals are being met. In CMMI-Dev it is also expectable that projects define their project goals (QPM.SP 1.1) and compose the define process accordantly by identifying the best set of processes (QPM.SP 1.2 and QPM.SP 1.3). Measure and analytic techniques are selected for quantitative management (monitor project performance).

The Plan component of QIP+EF is strong in practices that relate to the identification of project goals and planning the execution process and also the analysis process required to ensure project goals are being met. The CMMI-Dev is similar and the

emphasis is in defining the execution process based on the use of statistical and quantitative techniques applied on existing performance metrics from past projects.

Component	Measure	CMMI-Dev Specific Goal
Plan	High	QPM.SG 1 Prepare for Quantitative Management.

Table 5.8: *Plan* component mapping to CMMI-Dev specific goals

The Execution component

In the **Execution** component, the plan developed in the previous is executed along the planned analysis activities. Analysis should be an integral part of the execution process and drive the construction process. This description relate partially to the *QPM.SG 2 Quantitatively Manage the Project* Specific Goal (Table 5.9). In QPM.SG2 use of statistical techniques is expected to evaluate if project objectives will be satisfied considering current project performance. This is in-line with what QIP+EF defines as analysis activities, however CMMI-Dev includes further detail, namely, prescribes practices that focus on acting or changing the execution plan as result of analysis, and also practices related to identifying and removing causes of inefficiencies.

Component	Measure	CMMI-Dev Specific Goal
Execute	Medium	QPM.SG 2 Quantitatively Manage the Project.

Table 5.9: *Execute* component mapping to CMMI-Dev specific goals

Experience Base component

The **Experience Base** component summarizes the experience of the organization and makes it available to a initiating project. Experience is captured when project close and result can be captured considering two dimensions, first, the degree of precision/detail and second, the context dimension, defines the degree to witch experience is tailored to meet the specific needs of a project context. The dimension degree of precision/detail can range from a mathematical model developed over detailed quantitative data of project executed practices or qualitative acquired from project lesson learned. The context dimension defines if the experience can be generalized to be applicable to a broader set of project or if it's only applicable to a narrower projects contexts. This component relates partially to the CMMI-Dev high maturity process area Organizational Process Performance (OPP) (Table 5.10).

The purpose of OPP is to characterize the expected process performance of the organization's set of standard processes through the development of process baselines and models. Baselines and model relate to the precision/detail dimension of the Experience base component, in the use case of having detailed quantitative data that supports the development of mathematical models of performance. Development and analysis of baselines and models are expected as result of implementing OPP specific practices of *OPP.SP 1.3 Establish Process Performance Measures* and *OPP.SP 1.4 Analyse Process Performance and Establish Process Performance Baselines* and *OPP.SP 1.5 Establish Process Performance Models*. CMMI-Dev defines the requirements for a form of packaging experience at the organizational level resulting from performing projects that is later used by initiating projects as a form of performance benchmark. This requirement relates to what is expected by the Experience Base component of QIP+EF.

Component	**Measure**	**CMMI-Dev Specific Goal**
Experience Base	Medium	OPP.SG 1 Establish Performance Baselines and Models.

Table 5.10: *Experience Base* component mapping to CMMI-Dev specific goals

Learning and Feedback component

The last component of QIP+EF is the **Learning and Feedback** component and it's embedded in the paradigm as its considers the performing project as the learning vehicle for improving software engineering practices. Experimentation is supported by the use of the Goal Question Metric (GQM) paradigm which is used for defining goals, hypothesis and metrics to assess hypothesis. The feedback is possible by the integration of all components where information present in the Experience Base component is made available to initiating projects.

CMMI-Dev also promotes learning and feedback in the scope of Organizational Process Performance process area. Organizational performance shortfalls are identified by studying how current performance can hamper meeting established organizational objective, namely in the scope of *OPM.SP 1.2 Analyse Process Performance Data* and *OPM.SP 1.3 Identify Potential Areas for Improvement* of *OPM.SG 1 Manage Business Performance* specific goal. Two out of three practices of OPM. SG 1 are related to learning and feedback as described by QIP+EF. Thus the mapping with this specific goals is classified as *Medium*.

Performance shortfalls are used to identify possible improvements that are evaluated using statistical techniques and other quantitative techniques by means of experimentation and piloting in projects. This is expected in the scope of practices of *OPM.SP 2.2 Analyze Suggested Improvements*, *OPM.SP 2.3 Validate Improvements* and *OPM.SP 2.4 Select and Implement Improvements for Deployment*of *OPM.SG 2 Select Improvements* specific goal. Analysis and validation of improvements by means of statistical techniques relates to QIP+EF practice of using the Goal Question Metric (GQM) paradigm to define metrics that alight with project goals which are used to monitor and evaluate if improvements are being met. The mapping to OPM.SG 2 is classified as *High* as three out of four practices are related to the Learning and Feedback component.

Successfully piloted improvements are then deployed in the organizational standard set of processes in the scope of *OPM.SP 3.1 Plan the Deployment* and *OPM.SP 3.2 Manage the Deployment* of *OPM.SG 3 Deploy Improvements* specific goal. Two out of three practises of OPM.SG 3 relate to Learning and Feedback component and thus mapping is classified as *Medium*.

Learning and feedback relates also with *OPP.SP 1.4 Analyse Process Performance and Establish Process Performance Baselines* of *OPP.SG 1 Establish Performance Baselines and Models* as baselines and models serve as vehicle to retain knowledge of performance of the organization. This information is then used in the scope of new projects. These practices relate to the feedback part and thus map only to one practice of OPP.SG 1 resulting in a *Low* degree mapping (Table 5.11)

Component	**Measure**	**CMMI-Dev Specific Goal**
Learning and Feedback	Low	OPP.SG 1 Establish Performance Baselines and Models.
	Medium	OPM.SG 1 Manage Business Performance.
	High	OPM.SG 2 Select Improvements.
	Medium	OPM.SG 3 Deploy Improvements.

Table 5.11: *Learning and Feedback* component mapping to CMMI-Dev specific goals

Is summary QIP+EF relates to specific goals set in *Organizational Performance Management*, *Organizational Process Performance* and *Quantitative Project Management* process areas. Table 5.12 summarizes the mapping between QIP+EF and CMMI-Dev. QIP+EF has no explicit reference to goals of *Causal Analysis and Resolution* process area which address identification and resolution of causes of defects but Organizational Performance Management, Organizational Process Performance and Quan-

titative Project Management specific goals are much in line with QIP+EF. Organizations using QIP+EF to drive process improvement are mostly aligned with CMMI-Dev high maturity practices.

	CAR.SG 1	**CAR.SG 2**	**OPM.SG 1**	**OPM.SG 2**	**OPM.SG 3**	**OPP.SG 1**	**QPM.SG 1**	**QPM.SG 2**
Characterize	–	–	–	–	–	–	L	–
Plan	–	–	–	–	–	–	H	–
Execute	–	–	–	–	–	–	–	M
Experience Base	–	–	–	–	–	M	–	–
Leaning and Feedback	–	–	M	H	M	L	–	–
Highest mapping	–	–	M	H	M	M	H	M

Table 5.12: Summary of QIP+EF vs CMMI-Dev High maturity mapping

5.2.4 IDEAL vs. CMMI-Dev High maturity

IDEAL[McF96] is a software process improvement program model that describes a set of phases on how to implement a SPI program and their target audience are process improvement managers in organizations that want to roll-out a SPI initiative. The main organizational architectural component of the program model are *Phases* which are organized in sequence, namely: *Initiating*, *Diagnosing*, *Establishing*, *Acting* and *Leveraging*. The model organizes the sequence of steps in two operational levels, first, a strategic component and second, a tactical component level. The strategic component, based on the organization business needs and drivers, provides guidance and prioritization for the tactical component activities. The first and last phases, Initiating and Leveraging, are at strategical level and the remaining compose the tactical component. IDEAL also defines *Tasks* as a architectural component for delivering further detail on what is expected to be performed during a phase. Together with Phases, Tasks are used to describe the how to implement the program model.

In order to understand how IDEAL relates to CMMI-Dev, the mapping between IDEAL and CMMI-Dev consider IDEAL at the phase level and CMMI-Dev high maturity specific goals architectural components. The rationale is that they are have a similar level of detail considering the architecture of each model. The result of the

mapping will assess the level of coverage expectable for CMMI-Dev high maturity specific goals if an organizational adopts IDEAL as a strategy for process improvement.

Initiating Phase

In the *Initiating* phase an organization acknowledges the need for SPI and establishes an infrastructure for SPI. The infrastructure is composed of members of the organization that will sponsor and act on behalf of the SPI program. A SPI plan is drafted to guide SPI efforts through the subsequent phases of the model. The purpose is to assess and ensure that the need for SPI is understood and accepted by an organization. Further, an initial SPI proposal is created and approved with business needs and drivers that motivate the improvement initiative.

This phase relates to CMMI-Dev in the sense that it involves identifying business needs and drivers for improvement. However there is not a specific goal in CMMI-Dev dedicated to bootstrapping a SPI initiative. CMMI-Dev defines in *OPM.SP 1.1 Maintain Business Objectives* specific practice of *OPM.SG 1 Manage Business Performance* specific goal the need to have a explicit link between business objectives and quality and process performance objectives, which are then used to drive, in a more tactical level, SPI efforts. Therefore OPM.SG 1 relates to the strategical concerns addressed at an Initiating phase of IDEAL. Only one out of three practices of OPM.SG 1 practice is relate to IDEAL Initiating phases thus the mapping is classified as *Low*.

Phase	**Measure**	**CMMI-Dev Specific Goal**
Initiating	Low	OPM.SG 1 Manage Business Performance.

Table 5.13: *Initiating* phase mapping to CMMI-Dev Specific Goals

Diagnosing Phase

In the **Diagnosing** phase, the goal is to diagnose strengths and weaknesses of the organization by developing baselines of how processes and organizational interactions contribute to the organization's business objectives. This implies gathering information from the organization and delivering reports on the findings. This phase relates to CMMI-Dev *OPP.SG 1 Establish Performance Baselines and Models*. The purpose of OPP.SG 1 is to characterize the expected process performance of the organization's set of standard processes by developing baselines and models of performance. This involves defining Quality and Process Performance Objective (QPPO)s, selecting which

processes to be considered to monitor identified QPPOs, establish performance measures of selected processes and develop baselines and models of process performance.

Diagnosing phase tasks relate to three specific practices of OPP.SG 1, namely: *OPP.SP 1.2 Select Processes*, *OPP.SP 1.3 Establish Process Performance Measures*, *OPP.SP 1.4 Analyse Process Performance and Establish Process Performance Baselines*. OPP.SP 1.5 is not addressed in the diagnosing phase as models of performance are not required in the diagnosing phase. Thus the mapping is classified as *Medium*.

Phase	**Measure**	**CMMI-Dev Specific Goal**
Diagnosing	Medium	OPP.SG 1 Establish Performance Baselines and Models.

Table 5.14: *Diagnosing* phase mapping to CMMI-Dev Specific Goals

Establishing Phase

In the Establishing phase the organization develops a SPI strategic action plan for a period of 3 to 5 years that provides guidance and direction to the SPI program. Business objectives and vision along with previous improvements efforts provide the input to develop the plan. In this phase, business issues and current and future improvement efforts are identified to be part, or reconciled with existing SPI strategic plan.

CMMI-Dev also expects that business issues are identified that are later used to identify and select improvement initiatives. Organizational shortfalls in meeting business and quality process performance objectives are expected to be identified in the scope of *OPM.SG 1 Manage Business Performance* specific practice *OPM.SP 1.3 Identify Potential Areas for Improvement*. Also, identification and selection of improvement efforts is expected to be performed in the scope of *OPM.SG 2 Select Improvements* namely the practice *OPM.SP 2.1 Elicit Suggested Improvements*. Based on the relations identified with CMMI-Dev, the Establishing phase relation to OPM.SG 1 is *Low* and to OPM.SG 2 also *Low* as only one practice of the specific goals is related to the Establishing phase.

The Establishing phase in IDEAL has a strong managerial perspective on the steps needed to develop a strategic plan providing a lot of detail on what needs to be performed to develop such plan. CMMI-Dev identifies the core elements of a strategic plan, but does not provide the level of detail the IDEAL describes for the establishing phase.

Phase	Measure	CMMI-Dev Specific Goal
Establishing	Low	OPM.SG 1 Manage Business Performance.
		OPM.SG 2 Select Improvements.

Table 5.15: *Establishing* phase mapping to CMMI-Dev Specific Goals

Acting Phase

In the Acting phase improvements are developed, piloted and deployed across the organization to address process issues found during the baselining phase. This phase when implemented provides a high coverage of CMMI-Dev *OPM.AG 1 Organizational Performance Management* process area specific goals. Specific goals covered by the acting phase are *OPM.SG 2 Select Improvements* and *OPM.SG 3 Deploy Improvements*. In the scope of OPM.SG 2 improvement efforts are elicited, analysed and validated in accordance with their improvement proposals. These are detailed in specific practices *OPM.SP 2.2 Analyze Suggested Improvements*, *OPM.SP 2.3 Validate Improvements*, *OPM.SP 2.4 Select and Implement Improvements for Deployment* specific practices. In the scope of OPM.SG 3, validated improvements are deployment and the effects of the improvements are measured and evaluated as to how well they contribute to meeting quality and process performance objectives. Overall the Acting phase is inline with CMMI Organizational Process Management process area, except for OPM SG.1 (Table 5.16).

Phase	Measure	CMMI-Dev Specific Goal
Acting	High	OPM.SG 2 Select Improvements Performance.
		OPM.SG 3 Deploy Improvements.

Table 5.16: *Acting* phase mapping to CMMI-Dev Specific Goals

Leveraging Phase

In the Leveraging phase, the organization improves the SPI process by reviewing and analysing lessons learned from prior phases and incorporating changes into new SPI cycles. The context for SPI is re-assessed, motivation and goals are reviewed and a plan is developed to continue the SPI program. CMMI-Dev elaborates on the need to monitor control an improve the process in the scope of the *Generic Goal 2 Institutionalize a Managed Process*. In Generic Practice *GP 2.8 Monitor and Control the Process* one expects to take corrective actions as result of monitoring process execution against

the plan. Corrective actions can impact the process being executed or new instances of a process. In this sense, the Leveraging phase of IDEAL for SPI process relates to the Generic Goal 2 of CMMI-Dev applied to the OPM Process, where the improvement strategy is updated by monitoring and controlling improvement initiatives implementation. The scope of the mapping exercise shows that the Leveraging phase does not support directly the implementation of any CMMI-Dev high maturity practices, thus no mapping is established.

In summary, an organization using IDEAL for pursuing SPI is inline with several CMMI-Dev goals, namely goals related to OPM and OPP process areas. This evaluation highlights the more organization/process driven perspective for SPI set by the IDEAL model. Explicit links to CAR and QPM Process Area goals are not present. Table 5.17 summarizes the mapping between IDEAL phases and CMMI-Dev Specific Goals.

	CAR.SG 1	**CAR.SG 2**	**OPM.SG 1**	**OPM.SG 2**	**OPM.SG 3**	**OPP.SG 1**	**QPM.SG 1**	**QPM.SG 2**
Initiating	–	–	L	L	–	–	–	–
Diagnosing	–	–	–	–	–	M	–	–
Establishing	–	–	–	L	–	–	–	–
Acting	–	–	–	H	H	–	–	–
Leveraging	–	–	–	–	–	–	–	–
Highest mapping	–	–	L	H	H	M	–	–

Table 5.17: Summary of IDEAL vs. CMMI-Dev High maturity mapping

5.2.5 DMAIC vs. CMMI-Dev High maturity

DMAIC is the continuous improvement methodology part of the Six Sigma approach for process improvement. DMAIC has a problem solving perspective to improvement, it requires that a problem is identified and then uses a set of tools and techniques to arrive to a possible solution. The goal is that the solution will minimize or eliminate the problem and as result improve overall performance. DMAIC is composed of five steps, namely: *Define*, *Measure*, *Analyse*, *Improve* and *Control*. Improvement using DMAIC is established by executing improvement projects that typically start with a

problem and finish with establishing controlling mechanisms to make sure the problem is under control or eliminated. The mapping between DMAIC and CMMI-Dev will consider the DMAIC at the phase level and CMMI-Dev high maturity process areas at the specific goal component. The result will assess how CMMI-Dev specific goals are covered when DMAIC is used to guide process improvement.

Define Phase

The Define phase purpose is to identify a problem in line with organizational business priorities and make sure management support for solving the problem is present. Even though the emphasis is on identifying a problem, an opportunity for improvement can also be used to start a DMAIC improvement project. At the end of the define phase the scope of the improvement project is clearly identified and documented, the improvement project team is identified and a project charter is created.

In the context of CMMI-Dev high maturity, identifying a problem in performance is in the scope of Causal Analysis and Resolution (CAR) process area, namely the specific goal *CAR.SG 1 Determine Causes of Defects*. In the scope of *CAR.SP 1.1 Select Outcomes for Analysis* one expects to identify and analyse in detail a problem. As only one out of two practices from CAR.SG 1 are addressed, the classification is set to *Low*.

The Define phases also acknowledges that an improvement project can be grounded in the identification of an opportunity rather than a problem. In this perspective, the Define phase relates to the *OPM.SG 1 Manage Business Performance* namely the *OPM.SP 1.3 Identify Potential Areas for Improvement* specific practice where one expected that improvements are pro-actively identified. However the OPM.SG 1 is composed of two specific practices, that are not mentioned or expected as result of the Define phase, thus the mapping between Define phase and OPM.SG 1 is classified as *Low*.

Phase	**Measure**	**CMMI-Dev Specific Goal**
Define	Low	OPM.SG 1 Manage Business Performance
		CAR.SG 1 Determine Causes of Defects

Table 5.18: *Define* phase mapping to CMMI-Dev Specific Goals

Measure Phase

The purpose of the Measure phase is to collect information about the problem identified in the Define phase. A performance baseline is created for the process where the

problem or opportunity was identified. The baseline delivers a better understanding of what is happening in the process. Data is collected to develop such baseline and the problem is identified and quantified by understanding the impact of process inputs on the outputs. At the end of the Measure phase the process under study is understood and the possible causes of the problem are identified.

In CMMI-Dev, identifying causes of a problem is in the scope of *CAR.SG 1 Determine Causes of Defects* process area. Practices related to identifying process outcomes for analysis, analyse causes and identify possible corrective actions are expected within CAR.SG 1. The practice of identifying process outcomes for analysis is in-line with the measure phases purpose, namely, collect data on the problem by developing baselines. Identifying outcomes may require data collection, thus Measure phase mapping to CAR.SG 1 is classified as *Low* as analysing and identifying root causes is out of scope of Measure phase.

Also, when the improvement is based on an identified opportunity rather than a problem, the Measure phase aligns with *OPM.SG 1 Manage Business Performance*. In OPM.SG 1, *OPM.SP 1.2 Analyze Process Performance Data* purpose is to analyse process performance data and identification of potential areas of improvement. These practices align with the Measure phase if a baseline needs to be created to perform a detailed analysis of process performance.

The Measure phase is much in-line with the *OPP.SG 1 Establish Performance Baselines and Models* specific goal. Creation and analysis of baselines and process modelling are expected practices of OPP.SG 1, which is the main purpose of the Measure phase for understanding process behaviour. The Measure phase focus on creation of process baselines, similar to OPP.SG 1 but modelling process performance is out of scope and thus the mapping is classified as *Medium*. Table 5.19 summarises the mapping between Measure phase and CMMI-Dev.

Phase	**Measure**	**CMMI-Dev Specific Goal**
Measure	Low	CAR.SG 1 Determine Causes of Defects
	Low	OPM.SG 1 Manage Business Performance
	Medium	OPP.SG 1 Establish Performance Baselines and Models

Table 5.19: *Measure* phase mapping to CMMI-Dev Specific Goals

Analyse Phase

The purpose of the Analyse phase is to understand cause and effect relationships in the process under study, namely, how changes in the inputs affect the outputs. The assessment is supported by analysis of data collected in the Measure phase. The Analyse phase objective is to identify which factors and how these impact the performance of the process. The analysis phase may require collecting further data on process performance.

In CMMI-Dev, analysing the process behaviour in the scope of problem elimination occurs in the scope of *CAR.SG 1 Determine Causes of Defects* specific goal, namely the *CAR.SP 1.2 Analyse Causes* specific practice is in-line with Analyse phase of DMAIC and thus are classified as *Medium* as only half practices of the goal are met. Analysing the process in the context of improving process performance is expected in the scope of *OPM.SG 1 Manage Business Performance*. Expected practices for this goal *OPM.SP 1.2 Analyse Process Performance Data* and *OPM.SP 1.3 Identify Potential Areas for Improvement* are also in-line with the analysis expected in the Analyse phase of DMAIC. The relation is classified as *Medium* as two out of three practices are covered.

The Analyse phase relates also to *QPM.SG 2 Quantitatively Manage the Project* specific goal, namely with the *QPM.SP 2.3 Perform Root Cause Analysis* specific practice, where project issues are analysed to derive possible corrective actions. The mapping with QPM.SG 2 is classified as *'low'* as only one of the three practices os related to the Analysis phase.

The Analyse phase also relates to *OPP.SG 1 Establish Performance Baselines and Models*. A possible outcome of the Analyse phase are models of process performance derived from the analysis of process data and these provide the highest level of understating on process behaviour. Models are also expected as result of *OPP.SP 1.5 Establish Process Performance Models* that provide insight into process behaviour and also the ability to predict process and product characteristics relevant to the organization. However, the Analyse phase allows a *Low* coverage to CMMI-Dev OPP.SG 1 as only one sub-practice out of five is related to the Analyse phase. Table 5.20 summarizes the mapping between Analysis phase and CMMI-Dev.

Improve Phase

In the improve phase the goal is to change the process in line with the understanding obtained in the Analyse phase. Solutions for the problem under analysis are tested and evaluated. A pilot implementation is often conducted prior the full scale deployment of the change.

In CMMI-Dev changing the process to address process related issues is expectable

Phase	Measure	CMMI-Dev Specific Goal
Analyse	Medium	CAR.SG 1 Address Causes of Selected Outcomes
	Medium	OPM.SG 1 Manage Business Performance
	Low	QPM.SG 2 Quantitatively Manage the Project
	Low	OPP.SG 1 Establish Performance Baselines and Models

Table 5.20: *Analyse* phase mapping to CMMI-Dev Specific Goals

in the scope of *CAR.SG 2 Address Causes of Selected Outcomes*. Practices related to implementing selected action proposals developed in causal analysis (*CAR.SP 2.1 Implement Action Proposals*) and evaluation (*CAR.SP 2.2 Evaluate the Effect of Implemented Actions*) deal with solutions implementation and evaluation. If successful, these solutions can originate improvement proposals to be deployed at the organizational level. The mapping with the Improve phase is classified as *High* as most relevant practices from CAR.SG 2 are addressed.

Changing the process with the purpose of addressing improvement opportunities is in the scope of *OPM.SG 2 Select Improvements*. OPM.SG 2 expected practices relate highly with the Improve phase. In OPM.SG 2 improvements are identified and characterized in the scope of *OPM.SP 2.1 Elicit Suggested Improvements*, analysed for their ability to improve the process in *OPM.SP 2.2 Analyse Suggested Improvements*, and validated by performing pilot studies if necessary, in *OPM.SP 2.3 Validate Improvements*. Finally, selection and implementation of improvements for deployment throughout the organization is expected in the scope of *OPM.SP 2.4 Select and Implement Improvements for Deployment*. OPM.SG 2 is much aligned with Improve phase and is classified as *High*.

Full scale deployment of improvements and their evaluation is also expectable in the scope of *OPM.SG 3 Deploy Improvements* specific goal. This CMMI-Dev goal maps to the full scale deployment also expectable in the Improve phase and thus the mapping is classified as *'High'*.

Finally, corrective actions on issues identified in the scope of running projects are also expectable in the scope of *QPM.SG 2 Quantitatively Manage the Project* specific goal. The expected practice *QPM.SP 2.3 Perform Root Cause Analysis* involves identifying and analysing potential actions, implement selected actions and assess their impact. The mapping with QPM.SG 2 is classified *Low* as the scope of QPM.SG 2 includes two specific practices that are out of scope of the Improve phase. Table 5.21 summarizes the mapping exercise.

Phase	Measure	CMMI-Dev Specific Goal
Improve	High	CAR.SG 2 Address Causes of Selected Outcomes
	High	OPM.SG 2 Select Improvements
	High	OPM.SG 3 Deploy Improvements
	Low	QPM.SG 2 Quantitatively Manage the Project

Table 5.21: *Improve* phase mapping to CMMI-Dev Specific Goals

Control Phase

The control phase purpose is to ensure that improvements obtained during the Improve phase are maintained after the improvement project has ended. This involves standardisation and training necessary to make effective the changes resulting from the improvement project. Also, a plan for monitoring the process and reacting to any problems that arise needs to be put into practice.

In CMMI-Dev the monitoring and controlling of process performance is expectable in the scope of *OPM.SG 1 Manage Business Performance* specific goal, namely in the practice *OPM.SP 1.2 Analyse Process Performance Data*. In this practice, the capability of the organization to meet business objectives is assessed by analysing processes performance baselines to identify performance shortfalls. The mapping is classified as *Low* as only one out of three practices of OPM.SG 1 is related to the scope of the Control phase.

Phase	Measure	CMMI-Dev Specific Goal
Control	Low	OPM.SG 1 Manage Business Performance

Table 5.22: *Control* phase mapping to CMMI-Dev Specific Goals

Table 5.23 summarizes the coverage of CMMI-Dev high maturity specific goals an organization using DMAIC can expect to achieve. Most of the specific goals are addressed by DMAIC, namely Analyse and Improve phases provide good coverage for CAR and OPM process goals. DMAIC acknowledges the process as the focus for improving performance by elimination of problems or improvement opportunities. CMMI-Dev assumes that organizations are mainly project oriented and thus defines specific goals for quantitative project management, namely QPM.SG 1 that is out of scope of DMAIC.

Based on the mappings described in the previous sections the overview of the level

	CAR.SG 1	CAR.SG 2	OPM.SG 1	OPM.SG 2	OPM.SG 3	OPP.SG 1	QPM.SG 1	QPM.SG 2
Define	L	–	L	–	–	–	–	–
Measure	L	–	L	–	–	M	–	–
Analyse	M	–	M	L	–	–	–	L
Improve	–	H	–	H	H	–	–	L
Control	–	–	L	–	–	–	–	–
Highest mapping	M	H	M	H	H	M	–	L

Table 5.23: Summary of DMAIC vs. CMMI-Dev High maturity mapping

of association between analytical approaches to SPI and CMMI-Dev high maturity process areas is summarized in 5.24. Based on the highest mapping score one can conclude that PDSA is the analytical approach when implemented provides the smallest set of practices that can be used to fulfil CMMI-Dev high maturity Specific Goals. PDSA is the oldest approach to analytical SPI and is the predecessor of more recent analytical approaches.

Only one approach when implemented seams to provide coverage for the scope set by CAR PA Specific Goals. This fact relates to the perspective given by DMAIC, which is mainly focused on driving process improvident by solving existing problems.

QIP+EF is mainly focused on improving the project and thus organizations pursuing a QIP+EF approach are mostly alighted with QPM process area specific goals. For the process areas of OPM and OPP all analytical approaches, apart from PDSA on OPP, are much in-line on what is expectable for these process areas. This result supports the rationale for the affirmation that CMMI-Dev high maturity process areas are much alighted with what can be expected from pursuing an analytical paradigm to SPI.

Further, high maturity process areas are supportive of different perspectives to process improvement. The first is more focused on improving the project, by having a structured approach to solving issues that arise during the project. A second one, with an emphasis on the organizational perspective, involves building and sharing new knowledge from closing projects or by identifying and deploying incremental or innovative improvement to existing processes of the organization.

	CAR.SG 1	CAR.SG 2	OPM.SG 1	OPM.SG 2	OPM.SG 3	OPP.SG 1	QPM.SG 1	QPM.SG 2
PDSA	–	–	L	M	L	–	–	–
QIP+EF	–	–	H	H	M	M	H	M
IDEAL	–	–	L	H	H	M	–	–
DMAIC	M	H	M	H	H	M	–	L

Table 5.24: Summary of analytical approaches to CMMI-Dev High maturity mapping

5.3 A process for Software Process Improvement

This section describes in detail a process improvement process alighted with CMMI-Dev high maturity process area of Organizational Process Management specific goals. The activities that compose the process are described using UML 2.0 activity diagrams, detailed in Figure 5.3. Each activity will be detailed using Entry Criteria, Tasks, Verification and Validation and Exit Criteria (ETVX) defined by Radice*et.al* [RROC85] process modelling method.

ETVX method is suitable for describing activities at a sub-process level. The method is based on four concepts that are applicable to any activity, namely: *Entry Criteria*, *Tasks* (to be performed), *Exit Criteria* and *Validation* conditions for each task. The purpose of using ETVX is to detail an activity at an operational level to serve as a guide to process improvement teams.

In detail, the four concepts defined by Radice are:

- **Entry criteria**: describes the necessary conditions to begin an activity. A set of inputs and their status is defined and traceability from previous activities is identified if applicable.

- **Tasks**: describe what needs to be accomplished, including an enumeration of procedures or steps. Also, organizational roles, units or automation performing the tasks can be provided.

- **Validation**: defines the procedures or tasks to verify the quality of outputs produced by tasks.

- **Exit criteria**: defines the states and condition of outputs and identify, if applicable, next activities.

Activity A1: Define problem or improvement opportunity

An iteration or project for process improvement can be triggered with the identification of a problem that needs to be solved or an opportunity to improve performance of the organization. Expectations can be used to differentiate between and opportunity or problem. If an improvement is foreseen when current expectation are being met then the organizations faces an opportunity for improvement otherwise there is a problem that needs to be resolved to meet current expectations.

Relevant inputs for this activity are *process improvement proposals* that provide a description of current problems or opportunities for the organization. Also, *objectives of performance* for the organization need to be available. These provide the driver for improvement to which the result of the improvement initiative is evaluated for success. The output of this activity is a detailed description of the problem or opportunity identified, including related outcomes, processes directly related to the outcomes and resources used. Analysing the problem or opportunity involves considering three relevant perspectives: *efficiency* (effort taken to deliver an outcome), *effectiveness* (considering a measure of quality of the outcome, e.g. the number of defects found) and *resources* (tangible or intangible products or services involved). Based on this information a decision needs to be made concerning the necessity to collect data related to process performance. If data exists then an analysis of process related data is possible and the next activity is to *Analyse process performance*, otherwise data needs to be collected and the process should follow to *Establish performance baseline*.

- **Entry criteria**: A problem or opportunity is identified by the organization. Business objectives or quality process performance objectives for the organization are defined.

- **Tasks**: An initial analysis of the problem or improvement is performed and an associated business or quality performance objective is identified to be impacted by the problem or opportunity.
  Outcomes and associated process activities involved are identified. An estimate of impact of the change in business or quality performance objectives is computed.

- **Validation**: A clear link between the problem identified and a business or quality performance objective is identified.

- **Exit criteria**: The problem or opportunity is described identifying the tasks and associated expected outcomes of the improvement initiative. There is a performance objective in use that is linked to the problem or opportunity identified. The outcomes, processes and resources associated to the problem or opportunity

are identified and documented. An estimate of the benefit for the improvement or cost associated to a problem is provided. The process should follow to the next activity based on the existence or absence of performance baseline data.

Activity A2.1: Analyse process performance

Analyse process performance purpose is to derive relevant information about possible theories of how process inputs relate to process outputs. Models are the best approximation for predicting process behaviour but are hard to identify. Modelling involves identification of characteristics of inputs and outputs and how these relate to each other. Ideally, one process has one model that explains how inputs are converted to outputs. When deriving a model is not feasible, several performance baselines should be identified. This involves choosing relevant measures that characterize process execution, e.g., effort spent on a specific process step and measures related to characteristics of the outputs. In both cases, there is strong need to apply statistical techniques when developing models or computing performance baselines thresholds.

- **Entry criteria**: A process performance baseline is available. A problem or improvement opportunity is identified.
- **Tasks**: Exploratory studies are performed to understand the relation between process inputs and outputs. Statistical techniques are used to derive models that explain process performance, e.g., regression analysis and simulation.
- **Validation**: Statistical techniques are used to support the analysis and generation of information about process performance.
- **Exit criteria**: Process performance models are defined and/or baseline performance thresholds are identified. The next activity is A3: Identify and select improvement.

Activity A2.2: Establish performance baseline

Establishing a process performance baseline involves defining relevant process measures that reflect appropriate attributes of the selected processes. The purpose is to characterize the process objectively. Whoever performs the process, e.g., a set of projects in an organization or a single team, these measures should be collected systematically.

Software measurement is by itself a large topic highly relevant in the scope of this activity. Measurement, has a decisive factor in the success of process improvement

initiatives and several strategies exist that help organizations develop software measurement programs. The Goal Question Metric paradigm provides a method to identify such measures [BHL+07][BLR+10]. When establishing process measures several factors are relevant for a successful measurement program, namely; measures should be driven by business goals, an understanding of the value of a measure is clear for whom performs the measurement, ensuring data quality and provide adequate training are decisive factors in establishing process measures [ED07]. Additionally, the value systems of an organizations or team are crucial in achieving honest and professional behaviour the when implementing measurements.

Performance baselines are a by product of measurement programs. Baselines are valuable in providing a measure of variability of the process allowing an understanding of the expectable range of performance.

- **Entry criteria**: a process or set of processes are identified that relate to the problem of opportunity identified.
- **Tasks**: The process target of analysis is characterized and analysed. Process measures are defined and established along with data quality criteria. Operational definitions are developed to perform measurement. Training is provided on how to collect measurements. Data collection is performed.
- **Validation**: Data quality criteria is verified.
- **Exit criteria**: A dataset or baseline of collected measures is available. Data quality criteria is met.

Activity A3: Identify and select improvements

Having performed a process performance analysis the next step is to identify and select a solution for implementation. The expectation is that a deep knowledge of how process attributes in combination with process inputs relate to process outputs, guides the exercise of identifying relevant changes to the process. Additionally, having a model of process performance, a process change can be simulated to an estimate of the possible output. This estimate can then be used as criteria for selecting the most promising change. Additionally, different solutions have different costs associated. A cost benefit analysis at this point is performed to decide which solution may provide the best return on investment.

Along with the decision to implement a solution an additional decision must be made, whether the solution is to be tested first in a controlled environment or if it is deployed to all process performers. The decision may rely on several factors but two factors play a relevant role in the decision. First, the cost associated to implementing

the solution and secondly the expected impact on performance. Lower cost solutions with a low operational impact on process may justify skipping a pilot study of the solution, otherwise the option for performing a pilot study should be selected.

- **Entry criteria**: An analysis of process performance is available, including analysis of models of process performance and process performance baselines.
- **Tasks**: Possible improvements are elicited from the organization or improvement work groups. Improvement suggestions are analysed by considering their expectable impact on process performance making use of existing process performance models and baselines. A cost estimate of the process change is computed. A decision is made for which solution best suites improvement goals.
- **Validation**: Criteria for selecting the best solution should consider information derived from process performance models and baselines.
- **Exit criteria**: A process change is selected and a detailed description of what need to change is available. The overall estimated cost of implementing the process change is available. The next activity is to perform a pilot project to test the process change or to proceed with a deploy of the improvement.

Activity A4.1: Implement improvement and plan pilot

A pilot project provides an mechanism for validating a solution in a controlled environment and minimizes the risk of investing in an unsuccessful process change.

Preparation of the pilot involves implementing the desired process change and later defining and preparing the environment necessary to conduct the pilot. Implementing a solution may involve different forms of process updates, e.g., update existing workflows by adding, removing or updating existing process steps and/or introduction of new support tools.

Success of a process improvement initiative relies heavily on the result of a pilot experiment. The soundness of the result provides the necessary confidence to proceed with an organization wide process change. Costs associated to a full scale deployment can be extremely high and a pilot provides information that supports decision making and minimize the risk of investments in unsuccessful initiatives.

The goal of Experimental Software Engineering is the validation of software engineering claims and theories by designing and executions of experiments [WRH+12]. Pilots are also experiments that aim to validate software process improvement proposals, thus models of the experimental process can be helpful in supporting the execution of pilot experiments.

An experimental process model proposed by Goulão and Abreu [GBEA07] defines a structured approach for planning and execution of experimental software engineering studies. It defines a set of activities and deliverables in accordance with guidelines for reporting controlled experiments proposed by Jedlitschka and Pfahl [JP05]. The model defines the following activities (bullets) and sub-activities (inner bullets):

- Requirements definition
  - Problem statement
  - Context definition
  - Objectives definition
- Design planning (Experiment planning)
  - Hypothesis formulation
  - Variable selection
  - Subjects selection
  - Experiment design
  - Collection process definition
  - Analysis techniques
  - Instrumentation
- Data collection (Experiment execution)
- Data analysis
  - Data description
  - Data set reduction
  - Hypothesis testing
- Results packaging
  - Results interpretation
  - Threats identification
  - Inferencing
  - Identification of lessons learned

We argue that experimental process models provide the base structure for planning pilot projects in the scope of the process improvement initiatives. With roots in the scientific method along with the structured approach to derive knowledge from empirical evidence, the experimental process increases the confidence of results obtained from pilot projects. In the scope of this activity - *Activity A4.1: Implement improvement and plan pilot* - for implementing the improvement and planning the pilot, experimental process activities of *Requirements definition* for the experiment, *Experiment planning* are applicable. The remaining activities, *Experiment execution* relates to activity - *A4.2-Conduct pilot* - and *Data analysis* and *Packaging of results* relate to - *A4.3-Evaluate pilot results* - and will be described in following subsections.

The *experiment requirements definition* involves defining the context of the problem and objectives of the experiment. In more detail, *Planning the experiment* involves the following: *context parameters definition, hypothesis formulation, variables selection, subjects selection, experiment design, collection process definition, analysis techniques and instrumentation.*

It is also relevant to define the schedule for the pilot experiments, namely, defining when and where the experiments will take place. The activity is summarized as follows:

- **Entry criteria**: A description of the solution to be implemented is available.
- **Tasks**: Plan pilot experiment in line with experimental software engineering process activities of: define requirements for the experiment, plan the experiment. Develop the schedule for the pilot. Implement a prototype solution for the experiment by updating or definition process related guidelines, tools and templates.
- **Validation**: Verify if plan meets the criteria/guidelines to perform an experimental process. Verify the correct implementation of the pilot prototype solution.
- **Exit criteria**: The solution is implemented and a plan to carry out the pilot is created.

Activity A4.2: Conduct pilot

Conducting the pilot involves executing the plan created in *Activity 4.1: Implement improvement and plan pilot*. The relevant activities identified in the experimental process model by Goulão and Abreu [GBEA07] relate to: *collection clearance*, *motivation of participants*, *experiment execution*, *data validation* and *problem reporting*.

In the context of a process improvement process two of the activities in the experimental process model require special analysis. The *collection clearance* addresses the

issue of access and use of private data and *motivation of participants* concerns creating the favourable context for conducting experiments. When experiments are framed in an organizational setting, participants are performing experiments in the scope of their job and *collection clearance* and *motivation of participation* concerns are minimized or non-existent. Nonetheless, private information should not be disclosed or used to discriminate participants in any form.

Executing the pilot experiments requires that a person or group is identified as responsible for enacting the plan. The responsible should focus on operational requirements, namely ensure all resources are available and the experiment design is executed without deviations. If training is necessary it should be provided to the participants before the experiment. This often involves clarification of the change in the process and training in using new tools and templates.

Also relevant is validation of collected data and any deviation of the collection protocol. Misinterpretation and error in performing the data collection must be identified to allow, if possible, any correction or otherwise data should not be considered valid. Any deviation to the initial plan should be registered. Deviations must be considered for the purpose of evaluating the validity of the experiment. Often, organizational changes have impact on the planned schedule and availability of resources and differences to initial plan can invalidate the experiment design.

- **Entry criteria**: The solution is implemented and a plan to carry out the pilot is available.
- **Tasks**: Prepare experiment execution. Provide training to participants if necessary. Supervise experiment execution if possible. Perform data validation.
- **Validation**: Verify is experiments were performed according to plan. Verify if data validation was performed and any deviation to the plan reported.
- **Exit criteria**: Data resulting from the experiments is available. A report of deviations to the plan and a data validation report are created.

Activity A4.3: Evaluate pilot results

In alignment with the experimental process model *data analysis* begins by computing *descriptive statistics* of the data set resulting from the experiment. Next a *data set reduction* is executed by removing cases that represent atypical process behaviour. Extreme outliers and poor data quality, e.g., data resulting from wrong process execution, can considerably impact the quality of the information derived from a statistical analysis. Data analysis ends by performing the statistical test that evaluates the hypothesis under test.

This activity ends with *results packaging* that includes, *results interpretation*, *threats identification*, *inferencing* and *identification of lessons learned*. Results interpretation involves interpreting the outcomes of the statistical tests and confirm the result of the process change is meaningful or significant. Threats identification is relevant for evaluation the soundness of the experiment. Inferencing is supported by results of the statistical test and validity threats identified.

Lessons learned are useful to support planning and execution of future pilot projects. There may result from identifying and documenting unexpected or unplanned events that impacted the experiment. Whether negative or positive, they are valuable for future planning and execution of pilot projects.

Based on the pilot results, a decision is made about the continuity of the process improvement project. If successful, the process change is fully deployed to the organization. If not, one possible decision is to look for other solution of improvement or otherwise cancel the project.

- **Entry criteria**: Planned experiments in the scope of the pilot are performed. Data set resulting from the execution of the pilot is available.
- **Tasks**: Pilot results by considering descriptive statistics are analysed. Datapoints that may represent atypical process behaviour are identified and removed. Hypothesis is tested. Results are interpreted and threats to validity are identified. Infer knowledge and document lessons learned.
- **Validation**: All steps of the experimental process are performed.
- **Exit criteria**: The pilot is considered successful or unsuccessful. A decision is made on whether the process change is deployed, a new solution is tested or the improvement project is cancelled.

Activity A5.1: Plan improvement deployment

Having a decision to proceed to deployment, a plan is required to define the operational needs to successfully introduce the change into the organization or team. Planning the introduction of an organizational or team change is strongly related to change management topics [Kot96]. Change management relates to the steps taken for transitioning from a current state to a desired state. In the scope of a process improvement initiative, to successfully deploy a new organizational process or updating an existing one, process performers need to adopt the new process. This is most of the times a challenge as resistance to change is most likely to occur. For this reason deployment plans should consider change management approaches to structure the work to be performed during a roll-out to an improvement initiative.

Aside human factors, planning the deployment of a new process, also involves estimating the effort to implement the solution selected, define the steps to ensure an adequate communication of the process change and schedule training for the ones that are adopting the process.

Finally and of most relevance there is the need to establish the criteria for evaluation of the success of the process change. This involves defining and establishing measures and objectives for determining if the process change was effective and how to evaluate the impact in the organization's quality and process performance objectives.

- **Entry criteria**: An improvement is selected to be deployed to the organization.
- **Tasks**: Identify which process related documentation impacted by the process change requires updates and/or if new process definitions are necessary. Define the strategy for communication of the process change to the personal executing the process and also to the organization. Identify and plan necessary training to implement new or updated process definitions. Define the criteria for evaluating the success of the process change.
- **Validation**: Review and obtain commitment to the plan with involves stakeholders.
- **Exit criteria**: A plan for deployment is available. The next activity is to proceed with plan execution.

Activity A5.2: Deploy improvement

Having defined a deployment plan, the next step is putting the plan into practice. This activity involves implementing the process change that may require just updating existing process definitions with new guidelines to deployment of a completely new process.

Communicating the process change is important to make the organization aware of the change in progress. This relates to change management and is relevant not only to keep the organization informed but to create the necessary engagement for a successful change.

Training is a critical step for a successful deployment. Underestimate the value of training may considerably impact the adoption rate of the process. Training involves costs that need to be seen as a necessary investment for a successful deploy. Otherwise costs can be greater by having process performers under-performing the process, impacting directly efficiency and effectiveness.

Monitoring and controlling of the deployment plan is critical to ensure the success of the improvement initiative. Especially relevant is to monitor the collection of the

measures defined to evaluate the success of the improvement initiative. Without data, evaluation of deployment is not possible.

- **Entry criteria**: An plan for deployment is available. Necessary resources are available to execute the plan.
- **Tasks**: Create or update process documentation impacted by the process change. Communicate the process change to personal involved in the process and to the organization. Provide necessary training to support process changes. Monitor deployment plan execution and evaluate process performance impact on associated QPPOs.
- **Validation**: Check if deployment tasks are executed as planned. Progress reports are created.
- **Exit criteria**: New process documentation is created and/or existing process documentation are updated as planned. Training is provided if necessary. Process performance data is available. Progress reports are available that document verification of executed tasks and an assessment on the impact of process performance in QPPOs.

Activity A5.3: Evaluate improvement results

Evaluation of improvement results can be performed several times during the deployment phase but there is a point in time where a final evaluation is made on the success or not of the improvement initiative. This is expectable even if collection and evaluation of the measures established during the initiative are maintained beyond the scope of the improvement initiative.

Evaluation should consider the criteria defined in the deployment plan and the minimal requirement for having any form of evaluation is to assess if the process change was effective or not, meaning, the result during deployment corroborates the result obtained during the pilot. This should be feasible following the rationale used during the pilot. An important but more complex evaluation is measuring the impact on QPPOs. QPPOs may result from several factors that are not directly in control of the improvement initiative and for this reason the evaluation of the impact in QPPOs may be considerably constrained. QPPOs may represent the outcome of a complex systems of factors that are very difficult to control in an industrial setting. When this is the case, conclusion of whether or not variation in the QPPOs relates to the change in the process is hard to establish.

This activity requires that a data set of performance measures are available for evaluation.

- **Entry criteria**: The deployment plan is executed. Process performance data is available.
- **Tasks**: Perform an assessment of the impact of the process change in process performance. Evaluate impact of the change in associated QPPOs.
- **Validation**: At least the assessment of the impact of the process change in process performance is performed.
- **Exit criteria**: An evaluation of the impact of the change in process performance is available. An assessment of the impact of the process change in associated QPPOs is performed. A conclusion is derived on the success of the improvement initiative.

5.4 Analysis and discussion of results

Bilateral mappings were performed between analytical approaches to SPI and CMMI-Dev high maturity specific goals. The bilateral mapping is useful in providing a detailed description of the similarities and differences between mapped models. The technique requires that one of the model is considered as the reference model and the other model the mapped model. For the purpose of this exercise each of the analytical approaches was used as the reference model against CMMI-Dev as the mapped model.

Analytical paradigms were analysed in detail and corresponding CMMI-Dev specific goals were identified as corresponding goals. A categorical measuring scale, based on deterministic rules, was defined to categorize each compared architectural component translating the degree of similarity of expectable outcomes of both models. The categorization was subjective and thus questionable.

Four bilateral mappings were performed, where PDSA, QIP+EF, IDEAL and DMAIC analytical approaches to process improvement were used as the reference models when compared to CMMI-Dev high maturity process areas. The bilateral mappings focused on delivering evaluation, even though subjective, of how practices from the analytical approaches can be used to satisfy specific goals and expectable sub-practices of CMMI-Dev high maturity practices.

The results show that analytical approaches relate to CMMI-Dev high maturity in different extents. Based on the knowledge obtained from the detailed analysis of process analytical approaches and CMMI-Dev high maturity specific goals a process for software process improvement is proposed. The activities from the process resulted not only from the theoretical study of the analytical approaches and CMMI-Dev but

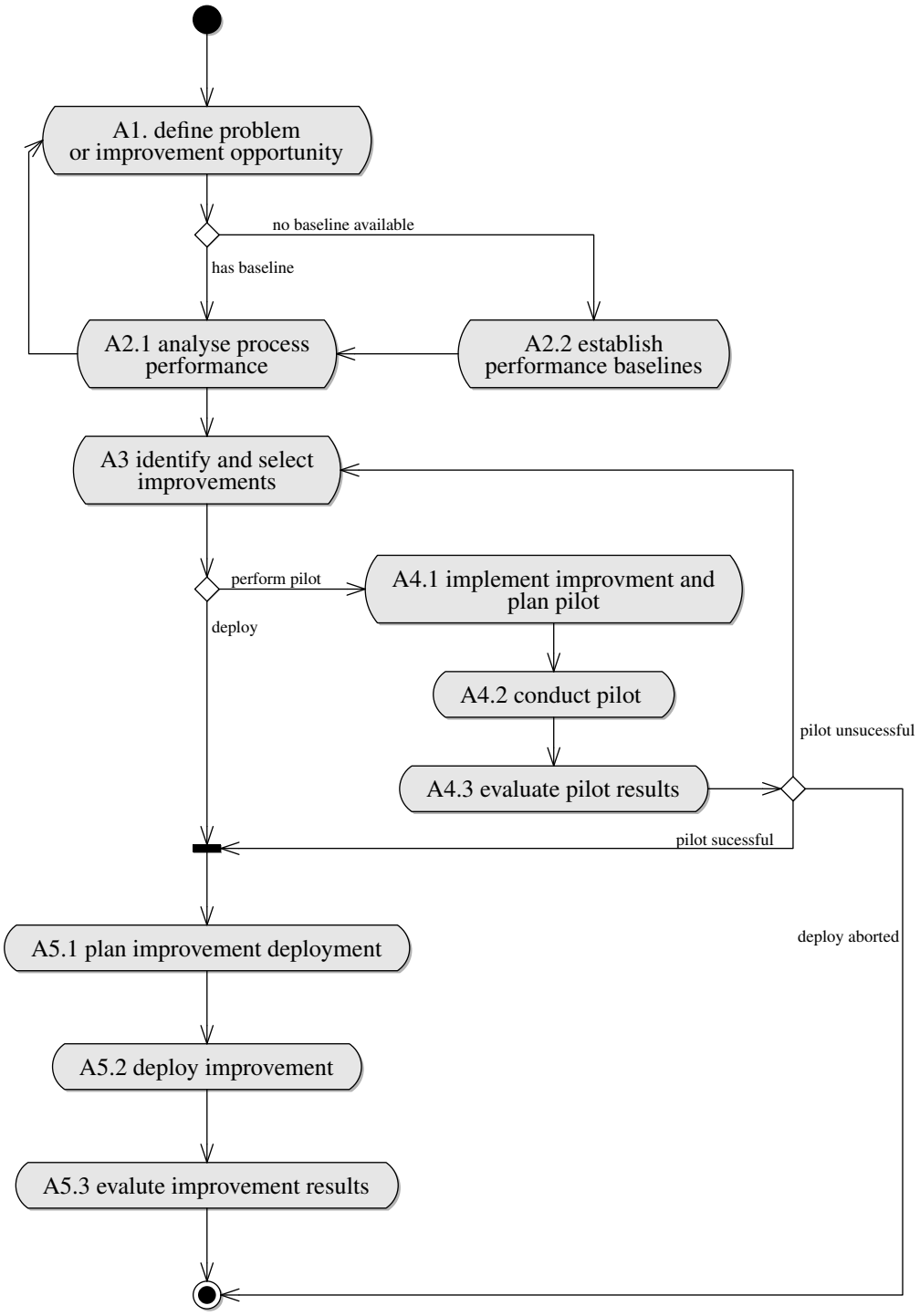

Figure 5.3: Overview of a process improvement process

also from analysis of organizational process improvement project performed in by CRITICAL Software.

5.5 Conclusion

In this chapter analytical approaches to SPI and CMMI-Dev high maturity process areas are compared using the bilateral mappings method. From the comparison it is possible to conclude that all analytical approaches relate to CMMI-Dev Organizational Performance Management process area. As summarized in Table 5.24, PDSA provides less coverage of CMMI-Dev goals and QIP+EF, IDEAL and DMAIC all provide a good base of alignment to Organizational Process Performance process area of level 4. DMAIC practices are alighted with Causal Analysis and Resolution process area specific goals (level 5 process area). Also QIP+EF is well suited for organizations that have project oriented operational strategies.

The outcome from the bilateral mapping can inform organizations in understanding how can they leverage an analytical approach to process improvement and satisfy CMMI-Dev high maturity requirements. The process improvement process proposed can be used by organization wanting to deploy an approach to software process improvement in line with an analytical paradigm and satisfying CMMI-Dev high maturity process areas.

Chapter 6

A Experience Report on Improving Code Inspections

Contents

This chapter contributes to the understanding of what is involved in implementing a software process improvement initiative in line with CMMI-Dev high maturity requirements in an industrial setting. The initiative was carried in a Portuguese software house, CRITICAL Software S.A. The focus of the initiative was two fold. First, describe how to identify the optimal review speed for a code inspection process and how this information can be used to improve process effectiveness. Second, provide a demonstration case of the applicability of a structured approach to software process improvement in line with CMMI-Dev high maturity process requirements.

6.1 Improving the inspection process

It is generally accepted that quality in software remains a challenge. A major quality issue with software is that defects are a by-product of the complex development process and the ability to develop defect free software remains a big challenge for the software community. It is possible to improve the quality of software product by injecting fewer defects or by identifying and removing defects injected. Software testing is an activity for defect identification and removal of these defects. Testing can be classified as static if a simple examination of the software artefact is performed to find problems or dynamic if an actual run of the software is required [Hat08].

A form of static testing is software inspections or sometimes referred as code reviews. An inspection is characterized *as a systematic approach to examine a product in detail, using a predefined sequence of steps to determine if the product is fit for its intended use* [PL03]. Software inspections require that a person, usually called a reviewer, spends and amount of time analysing the product to find defects. Whatever the structure/steps of the inspection process assumes, there is an amount of time spent by a reviewer inspecting the product and a corresponding number of defects found resulting from the review.

It is recognized that software inspections are a simple and cost effective approach to detect and eliminate defects [Lai98].They require less training when compared to other defect detection techniques. Inspections are cost effective in the perspective that detecting and fixing defects at earlier phases of development requires less effort when compared to finding and fixing these same defects at later phases of development. Moreover, inspections provide means to improve maintainability of software by allowing to detect certain types of defects that are not detectable using other defect detection techniques [SV01]. An example are evolvability defects found in a ratio between 5:1 and 3:1 to functional defects that otherwise would never be identified [ML09].

It is also general accepted that performance of software inspections is affected by several factors. Inspection performance is associated with the effort spent to carry out the process and/or the number of defects found. A wide set of empirical studies and new approaches have been proposed to understand and improve the inspection process since it was introduced by Fagan [Fag76]. Sources of process variability range from structure (how steps of the inspection are organized), inspection inputs(reviewer ability and product quality) techniques (applied to defect detection that define how each step is carried out), context and tool support [PPW$^+$02]. Most empirical studies try to assess the impact of specific process settings on performance. Some sources of variation are: absence or presence of inspection meetings, experience of the reviewers, initial code quality, number of reviewers participating in the inspection, time spent to detect defects and the rate at which products are inspected. Despite the efforts, a

general 'theory' that combines this sources of variability into a comprehensive set is still to be unveiled [PV97].

Review rate seems to be an important factor affecting inspection performance. High review rates have been conceptually and empirically associated to a decrease in inspections effectiveness [KP09]. When considering code as the inspected artefact, review rate establishes the number of lines of code (LOC) each reviewer reads per hour to find defects. Recommended rates that maximize the number of defects found are around 125 LOC/hour with a fast decline for rates above 200 LOC/hour [Fag86] [Wel93] [FO99]. However, limited investigations are available on the subject of finding the optimal rate to perform code inspections in industrial settings [KP09].

Motivation and research goal

Quality has been of strategic importance to CRITICAL Software since its foundation in 1998. Initiating operations in the Aerospace market segment, Quality has been since the beginning a requirement for growing a successful business.

The experience report documented in this section is part of CRITICAL Software investment in Quality. CRITICAL was the first company in Portugal to achieve CMMI-Dev Level 3 version 1.2) in Portugal and also the first one achieving CMMI-Dev Level 5 (version 1.2) in 2009. The practices described in this section contributed for a successful appraisal for CMMI-Dev level 5 in 2009 and later in 2012 (CMMI-Dev Level 5 version 1.3).

One of the goals driving this research effort relates to documenting an approach to find the optimal speed to perform code inspections in an industrial setting. The second research goal involves conducting an experiment of enactment of a process improvement process aligned with CMMI-Dev process areas of level 4 and level 5 staged representation. The experiment provides insights of how to improve a software process in line with analytical principles and CMMI-Dev high maturity requirements.

6.2 Software inspections related work

Software inspections were introduced by Fagan in 1976 [Fag76]. The goal of performing inspections is to improve the software product quality by finding and removing product defects. Main advantage of inspections when compared to other verification and validation activities is that defect identification and removal occurs closely to the point of injection, thus effort associated with finding and fixing defects is reduced [Fag76].

An inspection is defined as a sequence of steps or operations. The original proposal by Fagan considers five steps, namely: overview, preparation, inspection meeting, re-

work and follow-up [Fag76]. Since their introduction, inspections were subject of considerable research [PV97].The literature makes available a set of experiments designed to identify and quantify sources of process variability with the goal of improving inspection performance. Performance can be characterized as process effectiveness and efficiency. Effectiveness of inspection is the capability of the inspection process to detect all existing defects in the software artefact and efficiency is concerned with the effort spent in finding those defects.

Sources of variability can be associated with process structure, techniques used and process inputs. Studies focusing on process structure often re-define or remove process steps from the initial structure proposed by Fagan. Studies on process inputs involve controlling inputs by engineering or tightly controlling their attributes. A correlation is later established between these attributes and the resulting process performance.

Process inputs account for sources of variation that change across inspections; these are related with who inspects and what is inspected. Process structure and techniques are associated with how the steps are organized and how they are carried out, respectively.

A. Process structure as source of variation

Since the introduction of inspections by Fagan, only few approaches have been proposed by several authors. These focus mainly on re-definition of the process steps. Examples are Active Design Reviews [PW85], Phased Inspections [KM93] and N-fold inspections [SMT92]. These methods rely mainly on the argument that several persons focusing on special inspection techniques are more effective in finding defects than a single large team with no special techniques.

A synthesis on the subject of verification and validation is provided by Westfall [Wes09]. The term peer review is used to name any activity of reviewing software products created during software development. Peer reviews are categorized as deck-checks, walk-through or inspections. The degree of formality may vary in deck-checks and walk through. Inspections are strictly formal reviews and their applicability varies according to the state of the product. These n approaches are examples of formal reviews. According to [Wes09] a formal review or inspection follows the typical Fagan approach with a slight relevant change. The preparation step is used both for understanding and inspection. This simple change motivated, in the past, most empirical studies to evaluate the relevance of conducting the meeting inspection step. Considering a cost efficiency analysis, it is argued that meetings require too much effort for the additional defects found.

An example is a controlled experiment by Johnson and Tjahjono [JT98]. They analysed the impact of executing inspections with and without the inspection meet-

ing step. They analysed the impact on the following variables: *total defects, effort, false positive defects, duplicates and synergism*. They were unable to find significant differences in the total number of defects found when comparing meeting-based with meeting-less-based inspections (the meeting is eliminated and inspection occurs only in the preparation step). Conversely, the meeting-based required more total effort and effort per defect but resulted in significant less false positives defects (defects that were considered by the rework responsible as not true defects).

Also, *synergism* (interaction between reviewers in meetings) as a result of meeting-based inspections resulted in 30% of total defects identified. Meeting-less-based inspections resulted in more issues but one third of these issues were *duplicates* (found by more than one reviewer). Although results in what concerns productivity point in favour of meeting-less-based inspections, as cost per defect is lower, meetings allow participants to share review experiences, obtain insight into overall effectiveness of review, gain additional insight into the work product and its quality and finally it fosters collective ownership and responsibility for the review outcome.

McCarthy and Porter also suggest that meetings are not necessarily essential to successful inspections [PSTV95].They carried out the study to clarify previous studies on the matter that apparently reported meeting gains of 33% on defects found . They measured total defects applying different inspection structures. More defects were identified with meeting-less- based approaches in a context where artefacts inspected were requirements specifications.

Another structural factor subject of study is the optimal number of reviewers performing inspections. In this matter a study by Porter *et al.* [PSTV97] compared the performance of two and four element teams as single inspections. No improved performance in effectiveness was attained with four reviewers when compared with two reviewers. Single inspections were less effective than two elements inspection, but the difference was small.

Kantorowits *et al.* [KKR07] explored the use of an estimator to determine inspections team size. The estimator considers as parameters a characterization of detection ability and code domain knowledge of the reviewers to obtain a desired defect detection probability. The characterization of detection ability is a function of the individuals performing the inspections, type of artifact inspected and the method used for inspection. Code domain knowledge is specified as a continuous value between 0 and 1 and characterizes the limitation resulting from lack of knowledge in the domains by the reviewers in detecting defects. The study showed that values delivered by the estimator are close to the observed values from a controlled experiment. The estimator delivers the required number of reviewers to attain the desired defect detection. A reference to no more than seven reviews per meeting is suggested in [Wes09].

Other aspects of process structure are preparation time and meeting duration. Is

was been suggested that an increase in the preparation time correlated with the number of defects found [PSTV97]. Meeting duration also correlated with the number of defects found and meetings should occur no more than twice a day and duration should not exceed 120 minutes. Another structural related factor is the review rate at which inspection are performed. It relates product size (number of lines of code, requirements pages, etc) with the time each reviewer takes to inspect the product. High reviews rates are associated with a decrease in review effectiveness [KP09]. Recommended rates for preparation are around 100 and 125 LOC/hour with fast decline for higher rates, above 200 LOC/hour [KP09][Fag86][Wel93].

B. Techniques as sources of variation

Hatton *et al.* conducted a rigorous experiment to assess the impact on inspection performance of using checklists [Hat08]. The result was inconclusive on whether the checklist improved the number of defects found when enforcing a code review rate. No statistical significance was obtained from the results of the experiment. However the checklist focused only in one type of defect and code reviewed was relatively short. Reference to the successful use of checklists to aid the inspection process are described in [dACBP03].

A technique to improve the effectiveness of the inspection meeting is explored by Vitharana and Ramamurthy [VR03]. Through a controlled experiment, they studied the influence of anonymity in meeting/team based inspections. Anonymity implies that elements of the inspection team do not know the identity of participating elements. They observed the effect of anonymity in an experiment with a control group, by controlling the following variables: effectiveness (total defects), efficiency (less time) and reviewer attitude (freely express their tasks centred comments and views). They used two distinct groups with different background experiences and samples of code with different levels of complexity. The results showed that anonymity had no impact on efficiency and a significant impact on effectiveness occurred only when the code sample for inspection was classified as more complex. Anonymity also favoured inspector attitude towards the inspection.

C. Inputs as source of variation

Important sources of variability are the artefacts to be inspected and the reviewer inspecting the product. It is expectable that a product with a high number of initial defects be associated with high number of defects found. This association depends also on the reviewers ability to find defects. Reviewers less able or less experienced are expected to find fewer defects. Nair and Suma conducted an empirical experiment to

study the effectiveness of the inspection process. They observed project data from several leading service based and product-based software companies rated at level CMM level 5 [NV10].

Two metrics were considered to quantify the capability of the inspection process in capturing defects within the constraints of parameters affecting inspections. The first, characterized as people metric is the Inspection Performance Metric (IPM) that considers the number of defects caught in the inspection process (NI) over the Inspection Effort (IE). The second is Depth of Inspection (DI) characterized as a process metric, considers NI over the total number of defects (TD), where TD is NI plus the number of defects caught in the testing process. DI is characterized as a measure of effectiveness of inspection, defect prevention metric, quality metric and a measure of the ability of the inspection process in reducing the test effort. From the observed data they concluded that major sources of variability on effectiveness of inspections are the number, experience and skill of the inspectors but also preparation and inspection time. The IPM and DI are proposed as benchmarking tools to improve industry defect management practices. In another study Vitharana, and Ramamurthy also concluded that more experience has a significant impact in efficiency and effectiveness of inspections [VR03].

In summary, it is not clear the degree to which process structure impacts effectiveness but the impact seems to be small. Efficiency seems to be negatively affected by performing the meeting step. It is plausible that meetings improve the number of detected defects but require substantially more effort. Effectiveness also seems too improve when inspections are performed individually when compared with in a team based approach. Scenario-based detection techniques seem to be more effective than *ad-hoc* or check-list approaches and process inputs seam to explain more variation than structural factors [PV97]. Despite the effort for improving inspections, the relevant problem seems to be the lack of adoption of inspections spite the overwhelming evidences of their benefits [DS07][Rem05].

6.3 Improving software inspections

The improvement initiative documented in this section followed the improvement process documented in Section 5.3. Each of the next sub-sections provides a detailed description of what was performed in each of the activities of the improvement process.

A1. Define problem or improvement opportunity

Data collected prior 2008 on code inspections indicated that some inspections were being performed with high review rates ($LOC/hour$). Based on experience and a literature review an awareness was raised regarding inefficiencies in code inspections. The assumption was that high review rates were impacting negatively the ability to identify defects. This fact was reported internally in the organization and addressed to the quality control group of CRITICAL Software.

The ability to defect defects as result of code inspection was relevant in the context of an existing quality objective for controlling the total cost of quality of developing a software product. The *Cost of Quality* governance model aims to minimize the cost in achieving desired quality by categorising project costs on quality activities into four categories: preventive, appraisal, internal failure and external failure [Wes08]. The investment on each category is expected to have the overall result as described in Figure 6.1.

The cost lines show that by increasing appraisal and preventive costs one can decrease the costs associated to external and internal failure costs. The total cost of quality is given by the sum of appraisal, preventive, external and internal failure costs. The assumption of this model is that the external and internal failure costs are typically higher for each unit of defect in the product when compared to appraisal and preventive cost. In summary the model tells that preventing the injection of defects and detecting/evaluating the product for existing defects is the most cost effective approach for a defect free product.

The assumption underling the possible improvement was that one could decrease overall internal failure at later stages of development by investing effort in appraisal activities at earlier stages of development (moving the green line in Figure 6.1 to the right). This assumption holds only if the appraisal cost and the internal failure cost at earlier stages of development is in fact less than the appraisal and internal failure cost at later stages of development.

Code inspections are an appraisal activity and *defects removal*, also known as bug fixing, is an internal failure activity. Internal failure cost is associated to the activity of defects removal and the improvement expectation was that by decreasing the inspection rate a higher percentage of defects could be identified at earlier stages. This would have an overall decrease in cost of quality if:

1. *Additional code inspections are performed.* This is necessary, as by decreasing the review speed the amount of code reviewed also decreases. To ensure the same level of code coverage the appraisal effort needs to increase, meaning more inspections are necessary (moving on the green line in Figure 6.1 to the right).

2. *More effort required to remove defect resulting from code inspections*. This effort is considered internal failure costs and is associated to removing code inspection defects (moving on the yellow line in Figure 6.1 to the left).

3. *Decrease appraisal costs at later stages of development*. Appraisal costs at later stages of development typically run to the point here no more defects are found. Every time a defect is found a new appraisal is executed to verify the removal of the defect possibly, identify new defects. By having less defects at later stages of development, the cost of appraisal activities is expectable to decrease by decrease in the number of appraisal necessary to arrive at zero defects found.

4. *Less effort is spent on defect removal*. At later stages of development the number of defects detected decreases, (typically defects found as result of performing software testing) and the associated cost of removal also decreases.

Overall, cost of quality would decrease if:

$$\Delta_{effort}(1) + \Delta_{effort}(2) < \Delta effort(3) + \Delta_{effort}(4) \tag{6.1}$$

The rationale represented by the equation 6.1 indicates that a decrease in overall cost of quality is possible only if there is an increase of effort in code inspections. More code inspections results in more defects identified and as result an increase in internal failure cost associated to finding and removing inspections defects. The investment in code inspection pays off only if there are less defects in later stages of development.

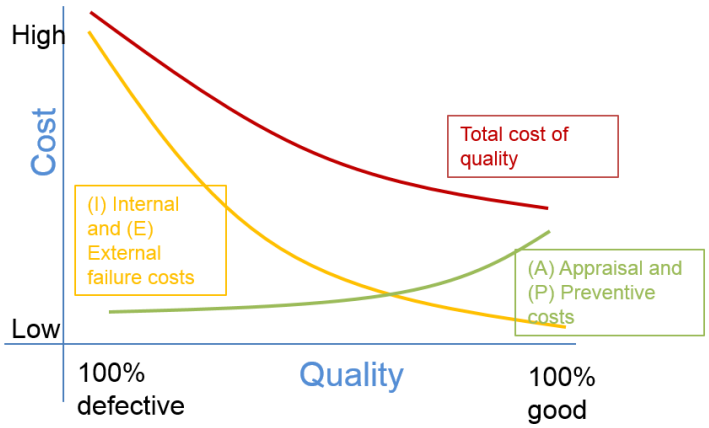

Figure 6.1: Cost of quality governance model

In summary and referencing the process model defined in Figure 5.3, as entry criteria, a problem or opportunity is identified: *improve code review effectiveness by*

decreasing review speed. At least one process performance objective is available: *cost of quality model for each project*.

An analysis was performed on the expected impact of the improvement proposal in the performance objective identified. The rationale supporting the improvement was submitted to a formal review process performed by the elements of the quality control group of CRITICAL Software. The outcomes of *A1* activity are listed in Table 6.1.

Problem identified	High code inspection rate
Goal associated	Cost of quality model for each project (QPPO)
Process / Procedure	Verification / Code inspection
Outcomes	Code inspection and number of defects found
Resources/Tools	Spreadsheet tool
Expected/Estimated benefit	A decrease in review speed will allow **higher defect detection at earlier stages of development** and leverage the decrease of internal failure costs at later stages of development, thus **decreasing overall cost of quality**.

Table 6.1: Summary of A1. define problem or improvement opportunity

To fully understand the impact of the review speed in the defect detection capability a detailed analysts of process performance was necessary. As no baseline was available, the next step was to define the performance baseline for the code inspection process (*A2.3. establish performance baselines*).

A2.2 Establish performance baseline

Formerly as a CMMI-Dev level 3 organization, CRITICAL Software had a standardized process for verification and validation of software products. The process includes a detailed procedure for performing code inspections that is monitored and controlled by a set of metrics. The procedure structure follows similar steps as described in [Wes09], namely:

- **Preparation (P),** each reviewer inspects the code. A code review check-list is used in the preparation step to help the reviewer perform the inspection. A list of defects is expected as a result of this step.
- **Review Meeting (M)**, in the review meeting, a walk-through reading of the code sample is performed to collect and discuss each defect identified in the prepa-

ration step. The author and reviewers participate in the meeting. Additional defects are recorded if identified as result of the group interactions. A full reading of the code may be performed but it's not mandatory.

- **Rework (R)**, the author receives a list of defects that must be removed. In a later step, the removal is validated by the inspection responsible.

Entity	Attribute	Metric
Code	Size	Lines of Code
Code	Language	Categorical
Code	Pre-inspection unit testing	Yes/No

Table 6.2: Process Input Entities

Based on these steps, two types of reviews are possible. The first one includes preparation, review meeting and rework (P-M-R). In the second, only preparation and rework (P-R) are performed. The available set of metrics, previously collected, allowed to characterize the entities participating in the inspection process as follows. Table 6.2 depicts input related entities. Table 6.3 depicts process structural-related entities. Each attribute has an associated base measure.

Entity	Attribute	Metric
Preparation session	Duration	Hours
Review meeting	Duration	hours
Review meeting	Review team size	Number

Table 6.3: Process Structural Entities

Based on this set of process related metrics, review rate was considered as the primary factor affecting inspection process performance and thus inspection effectiveness was modelled as a function of review rate.

$$CodeReviewEffectiveness\left[\frac{defects}{loc}\right] = f(code\_review\_rate)\left[\frac{loc}{hour}\right] \tag{6.2}$$

Code inspections effectiveness is characterized as a function of defects found by reviewers normalized by the size of code inspected. Review rate is the derived measure

that relates code size and preparation session duration in number of lines of code per hour of review performed. The decision to consider a single factor in equation 6.2 to model code inspection process was based on the following factors: firstly, available data from past inspections limited the number of variables that could be considered for modelling process performance. Collecting additional data to characterize already executed inspections did not present itself as a viable option as it required collecting information on past events. This could result in lack of accuracy of data. Secondly, data available resulted from inspections following the same inspection protocol.

Concerning process structure, inspection followed a *P-M-R* process structure described earlier. The range of values for review team size varied from 3 to 4 reviewers and review meeting duration was around 120 minutes. Conversely, review rate variability was considerable. No control was enforced by the defined process. It was up to the reviewer to inspect the code at the desired rate. This provided a good base for studying the impact of review rate on defect density. Additionally, review rate has been considered, based on empirical evidence, by several authors as an important non-negligible factor impacting the number of defects found.

In this scenario review rate was considered the decisive factor affecting process performance. The data used to build the model was obtained from code inspections performed by professional software engineers participating in a total of three projects. These were characterized as being representative of the typical software development projects at CRITICAL Software. Projects used *C* programming language and were characterized by having a typical team size and project duration.

Inspection data was generated from inspections following a *P-M-R* inspection process structure. Inspection teams varied from 3 to 4 reviewers. The reviewers spent an average of 90 minutes in the preparation session. A single inspection meeting lasted no more than 120 minutes. A total of 45 code inspections was considered to build the model, after a data set reduction to eliminate incomplete records and outliers. Information about the reviewers was also incomplete, therefore traceability to the reviewer was not considered. Prior to inspection, code was subject to static analysis using a compiler and rule checker. Reviewers were members of the development team and inspections were first time inspections.

Inspection data was collected using a spreadsheet template. All reviewers received from the inspection moderator a code sample to be reviewed. Each reviewer registered defects, typically in a printout of the code sample. In the review meeting the moderator registered the time each reviewer spent on the preparation session, the list of defects found, the final total number of defects found and code size. The inspection meeting duration was also registered.

A2.1 Analyse process performance

Having inspection data available the analysis step focused on finding an association between review rate and defect density. Additionally, we needed to build a model to estimate defect density based on review rate data. A regression analysis was used with the goal of characterizing any existing association. Figure 6.2 depicts a scatter plot of how defect density *(y-axis)* varies with inspection code rate *(x-axis)*. Since data is from real project inspections the values for defect density are absent due to confidentiality.

A basic descriptive measures of performance are provided in table 6.4. A total of 45 code inspections belonging to the baseline data were considered.

Measure	Average	Standard Deviation
Review speed ($loc/hour$)	860	533
Defect density ($defects/Kloc$)	–	18

Table 6.4: Code inspection process performance

Based on the scatter plot information, a linear association between variables was tested. Firstly, the conditions of applicability of regression analysis were validated. Using the Kolmogorov-Smirnov non-parametric test [Fie09] the normality of distributions of both variables was confirmed. Table 6.5 lists the models considered to find a possible measure a strength of association between the variables.

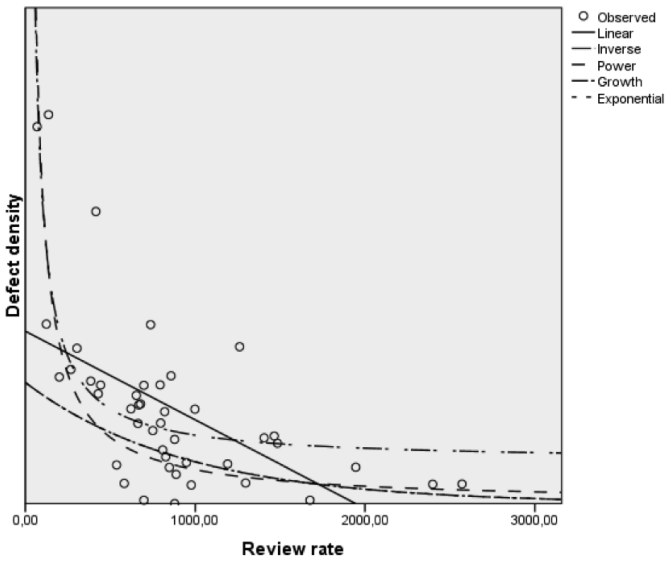

Figure 6.2: Data and linear models considered

Model	Formula	R-square	Sig
Linear	$y(t) = b_0 + b_1 t$	0.317	0.000
Inverse	$y(t) = b_0 + \frac{b_1}{t}$	0.583	0.000
Quadratic	$y(t) = b_0 + b_1 t + b_2 t^2$	0.485	0.000
Cubic	$y(t) = b_0 + b_1 t + b_2 t^2 + b_3 t^3$	0.569	0.000
Power	$y(t) = b_0 t^{b_1}$	0.093	0.044
Exponential	$y(t) = b_0 e^{b_1 t}$	0.061	0.170

Table 6.5: Fit models

The *Linear*, *Inverse*, *Quadratic*, *Cubic* and *Power* regression models are statistically significant $(p - values < 0.05)$. The inverse model has the highest R-square with a value of 0.583, a Fisher test statistic of $F(1;42) = 58.676$ for the Analysis of Variance (ANOVA) and a $p - value = 0.000 < 0.05$ (see Table 6.6).

The Inverse model explains 58.3% of the variability of the dependent variable. The data and models curve fits are depicted in the scatter plot (see Figure 6.2). With the objective to obtain a better fit with a higher *R-square* a curve fit without the constant b_0 was tested. We obtained a better fit with an improved R-square of 0.752. The inverse model without constant is significant $(p - value = 0.000 < 0.05)$ with a Fisher test statistic value of $F(1;43) = 130.218$ and with a higher R-square (see Figure 6.7). The model is given by $Y(t) = \frac{b_1}{t}$ and it is linear in terms of their parameters.

Equation	Formula	Model Summary	
Inverse	$y(t) = b_0 + \frac{b_1}{t}$	R-Square	Sig
		0.583	0.000

Table 6.6: Inverse model summary

We tested an additional non linear model based on the inverse relation: $Y(t) = b_0/t^{b1}$. The R-square obtained was 0.61 (inferior to the inverse model without constant).

Based on this information, the model with higher prediction power is the Inverse model without constant. Having an acceptable R-square the inverse model had the best predictor of process performance. Based on this information on process performance the next step was to identify and select possible improvements.

Equation	Formula	Model Summary	
Inverse (without constant)	$y(t) = \frac{b_1}{t}$	R-Square	Sig
		0.752	0.000

Table 6.7: Inverse model without constant

A3. Identify and select improvements

Considering the process model formulated in Table 6.7 and depicted in Figure 6.3, it was possible to see that a wide range of review rates were being used by reviewers. The average value was about $860\frac{LOC}{hour}$ with a standard deviation of $\sigma = 533$ and defect density declined considerably for high review rates.

Based on this evaluation a process change was considered. A new review rate was to be used for performing inspections in order to improve the number of defects found. Based on literature recommendations that argue a maximum of $200\frac{LOC}{hour}$ we used the model to estimate the expected defect density for this review rate. Comparing $200\frac{LOC}{hour}$ review speed with the project sample average defect density, if the model was accurate, reducing the review rate would provide an average of 70% increase in defect density resulting from code inspections (considering fact that data passed the normality statistical test).

Based on this evaluation, it was decided to carry out a pilot project where inspections would be performed with a controlled review rate. The pilot was to validate if defect density would improve as predicated by the model and secondly, if improvement was achieved, in which percentage did it occur.

A4.1 Implement improvement and plan pilot

The process change in this specific context was very simple to implement. The process definition was updated to have a new requirement for setting-up code inspections. A new directive recommended that code inspections should be planned to have between 200 and 250 *LOC* for each hour of inspection. Additionally, reviews should not be planned to last more than two consecutive hours.

A plan was elaborated to define the pilot approach and schedule. The plan considered the guidelines for planning of software engineering experiments defined by Goulão and Abreu [GBEA07].

Defining the requirements for the experiment

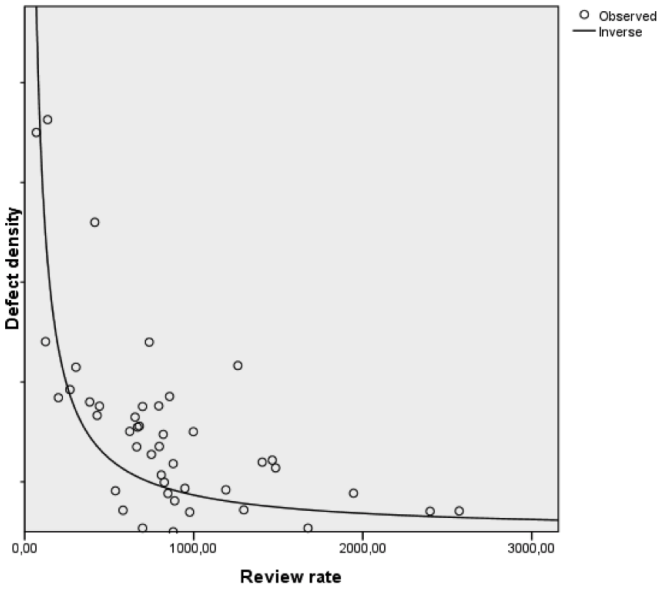

Figure 6.3: Code inspection performance model

The pilot motivation was to evaluate the impact of review speed in the ability to defect defects. In the perspective of the developers, guidance was necessary to define the right amount of code to review in a specific time frame. There was no recommendation for review speed given by the review process definition and developers felt the need for guidance on how to optimize their ability to defect defects. For project managers the concern was to meet the defined review coverage. By increasing the review speed one can spent less effort to achieve a predefined code coverage, however the ability to detect defects could be impacted.

The experiment was to be performed in the scope of real projects at CRITICAL Software. A set of projects was chosen to provide source code for the controlled code inspections. Also a set of reviewers was identified to perform inspections within the recommended review speed.

The objective of the experiment was defined as follows:

> Analyse ***the impact of inspection rate in defect detection rate***, for the purpose of ***finding an optimal inspection speed*** , with respect to ***improving the effectiveness of code inspections***, from the point of view of ***software developers*** *and* ***project managers***, in the context of a ***software process improvement initiative***.

Experiment planning

The experiment context is described considering the sources of variability for code

inspections identified in Section 6.2. The sources are used to compare the organizational context of the baseline and the context of the pilot.

For the pilot four projects were selected from a set of projects available to participate in the experiment. The selections was highly constrained by organizational factors that are mainly linked to availability of resources. Source code submitted for inspection was written in *JAVA* programming language. A poll of three reviewers was selected to perform the inspections. A total of 39 reviews were planned. The reviewers performing inspections followed the *P-R* approach (the review meeting did not occur) and in each inspection only one reviewer participated.

The limitation on the resources available led to frame the pilot in a *quasi experimental* design strategy. Quasi-experimental designs are characterised for not implementing true randomization between pre and post testing for the treatment [SCC02]. This strategy suites the case under discussion as the inspections considered for the baseline were not controlled in the scope of the improvement initiative. The challenge in quasi-experimental designs is that results are prone to internal threats to validity, mainly resulting from uncontrolled factors that can impact the outcome of the experiment, where cause and effect relationships are harder to establish.

For the quasi-experimental design scenario, the baseline setting was considered as the pre-test for the application of the treatment, in this case the absence of the control in the review speed, and the post testing scenario where the review speed or treatment was to be applied to a second group.

The differences between the baseline group and the treatment group could influence the number of defects identified as result of the inspection and by that, invalidate the pilot objective of allowing a comparison of the impact of review speed in defect density. Differences were considerable, namely, programming language of the source code under inspection, different reviewers, possible variations in initial code quality (different projects). Also different work-flows where applied and the checklist although the same was not mandatory to follow. A summary of the differences is provided in Table 6.8.

However, some of these factors have been studied in the past for their impact in the ability to detect defects. Concerning the inspection work-flow namely, the removal of the review meeting, has been argued that removing the meeting step does not impact significantly the ability to find more defects [JT98]. Also, although intuitively, a team of 3 or 4 reviewers is able to find more defects than a single reviewer (the added experience or ability of each reviewer could improve defect detection), previous studies concluded that a decrease in performance for single reviewer inspections may be negligible when compared with two and four element teams [PSTV97].

Hypothesis formulation and variables selection

Sources of variability		Baseline	Pilot
Process Structure	Work flow	P-M-R	P-R
	Number of reviewers	7	3
	Review team size	3-4	1
Techniques		Check-list based	Check-list based
Inputs	Number of Projects	3	4
	Programming language	C	JAVA

Table 6.8: Baseline and pilot context summary

The objective of the experiment was to evaluate if the review speed impacted significantly the defect density resulting from inspections. Considering the performance model, the expectation was that a change in the review speed would impact the resulting defect density. In this scenario the question raised is if a decrease in the review rate would impact significantly the defects detected. This can be framed in terms of the null hypothesis (H_{1_0}) and the alternate hypothesis (H_{1_1}).

H_{1_0} - there is no significant impact in the number of defects detected when the review speed is decreased.

H_{1_1} - a decrease in the review speed impacts the ability to detect defects.

The experiment dependent variable is *defect density*, measured in number of defects by KLOC and review speed, measured in $LOC/hour$ is the independent variable. Controlled factors of the experiment were the *reviewers* performing the review and the *projects* submitting source code for inspection.

Subjects selection

The subjects of the experiment are characterized as full time software developers, one with more than 5 years of experience and 2 junior developers (with up to 2 years of experience in developing software).

Source code subject of inspection was from different projects and projects in different domains, meaning the source code was authored by different persons and product requirements were also significantly different. The selection of software developers and projects was highly constrained by organizational factors. Ideally the context of the baseline setting would be replicated. This was infeasible mainly because there

were no available projects that fitted the profile of the project available for the baseline setting. Also, software developers that participated as reviewers in the baseline setting were also unavailable or did not fit the technological background needed for the inspections.

The sampling method used was mainly driven by convenience, considering projects and software developers availability and also project specific cost constraints. This highly constrained the context for the experiment.

Experiment Design

The experiment considered the applicability of the controlled inspection rate as the *treatment* and the *observation* the number of defect identified after performing the inspections with a controlled rate. The case study can be described as a quasi-experiment with non-equivalent groups design. The differences between the pre and post test groups are depicted in Table 6.8 which could in theory influence the ability to detect defects, namely initial code quality, programming language and reviewers ability are all factors that may impact the number of defects identified. For the purpose of this experiment we will consider that removing the review meeting from the workflow and performing single reviewer inspection rather than multiple reviewers have negligible impact in the outcome of the inspection. This assumption is mainly based on previous studies performed by [JT98] [PSTV97].

Contrarily, initial code quality and reviewers ability to detect defects are uncontrolled factors in the experiment that could influence the outcome of the inspection. Namely, source code subject for inspection in the pilot has no relation with source code used in the baseline setting (also the programming language could play a relevant factor final defect density). In a real setting controlling the number of defects present in the source code required tempering with the source code or performing additional inspections for defect control, both of which were not viable for the pilot scenario. Additionally, the set of reviewers to perform inspections in the pilot is different from the set of reviewers that performed inspections for the baseline scenario. The ability to detect defects is therefore a variable that is not controlled mainly because traceability to all reviewers participating in the baseline setting was not possible to establish due to missing information,

For the pilot, each reviewer was expected to perform a similar number of inspections of the source code from each project. This design decision was to block the impact of the combined factors in the outcome of the inspection, by assuring each reviewer add the opportunity to review code from all projects. The experiment is also classified as *online* as inspections were performed in the scope of real projects.

The scale of the experiment, namely the number of project and reviewers involved

lead us to consider the experiment *specific* instead of *general*. This means the impact on defect density might be replicable or valid in the scope of the organization only. Extrapolation to the *general* may incur in strong validity threats.

Collection process

The data resulting from the inspection was to be collected using the same approach used on the baseline context. After the inspection, spreadsheet based templates were to be used to report the number of LOC reviewed, the inspection time and a list of defects found along side with reviewer identification and source code and project identification.

Analysis procedure

The analysis steps focused on comparing the average and variability of defect density from the pilot and the baseline setting. Descriptive statistics were used to describe and understand both central tendency and variability of the samples. Significant differences in the average of defect density would imply that hypothesis H_{1_0} was to be rejected, meaning that a decrease in review speed has a non- negligible impact in the number of defect found.

Additionally, there was the goal of measuring the correlation between the independent and dependent variable and validate if the model of performance identified for the baseline was a good predictor for the pilot results.

Instrumentation

The source code to be used in the inspections was selected from projects participating in the pilot. This was real source code resulting from implementing product specific functionalities. The inspection procedure was briefed to reviewers and clarifications on how to execute the inspection were given, in an informal meeting, to reviewers prior the inspections. The inspection steps included the use of a spreadsheet template to report on defects found during inspections. A tool was used to calculate the number of lines of code that fulfilled the requirement for LOC counting.

A4.2 Conduct Pilot

The pilot inspections followed a similar collection procedure as described in process step *A2.2 Establish performance baseline* except that the reviewers had a review rate criterion to be met and the review meeting did not occur. A total of 39 inspections

were carried out by a pool of three reviewers in a context of four different projects. Additional information was made available to the reviewers, namely requirements and design specifications.

Defects were recorded by reviewers using spreadsheet based reports and were sent directly to the author for rework.

	Project 1	Project 2	Project 3	Project 4
Reviewer 1	3	4	3	3
Reviewer 2	3	4	3	3
Reviewer 3	3	3	3	2

Table 6.9: Pilot/Experiment setting

A4.3 Evaluate pilot results

Data analysis and description of the pilot results focused on confirming the impact of decreasing the review speed in the number of defects found by comparing the average defect density between the baseline scenario and the pilot.

1) Inspection review rate impact on defect density

The scatter plot of Figure 6.4 depicts the results for defect density for the pilot inspections. The plot depicts the same relation of the performance model identified in the process step *A2.1 Analyse process performance*, relating defect density and inspection review rate.

The Pearson correlation coefficient was computed to measure variables association. With the review rate controlled no significant correlation between the variables is evident $p - value = 0.233 > 0.05$.

The average review rate changed from 860 $\frac{LOC}{hour}$ to 215$\frac{LOC}{hour}$ with a standard deviation of 46. The average value for defect density increased, reviewers were able to detect more defects. The box plot (Figure 6.5) depicts the variation in average defect density values for both scenarios, prior (1.00) and after (2.00) the controlled review rate. The normality of the dependent variable (defect density) was checked using the Shapiro-Wilk test and the sample is normality distributed $p - value = 0.279 > 0.05$ in Table 6.11.

Analysing the box plot and the value for standard deviation an increase in the variability for defect density was a fact. A *Levene* test was used to assess equality of

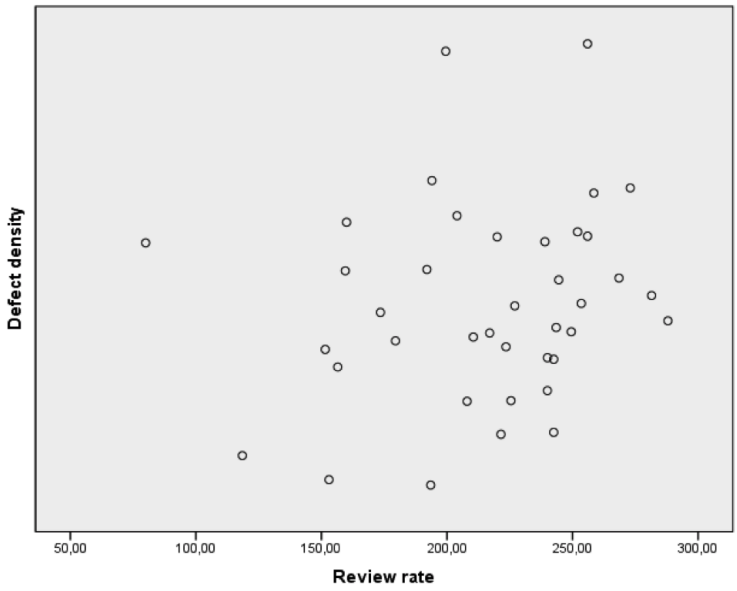

Figure 6.4: Scatter plot for new inspection setting

Measure	Average		Standard Deviation	
	before	after	before	after
Review speed ($loc/hour$)	860	215	533	46
Defect density ($defects/Kloc$)	–	–	18	25

Table 6.10: Comparison of inspection performance after the pilot

Shapiro-Wilk			
Variable	Statistic	df	Sig
Defect Density	0.996	39	.279

Table 6.11: Shapiro-Wilk statistical test for Normality

variances for both samples (Table 6.12). The test provided a $p-value = 0.033 < 0.05$ therefore, the variances are significantly different.

A statistical t-test for independent samples was used to evaluate the significance of this variation (Table 6.12) and assess if the null hypothesis (H_{1_0}) could be accepted. The test indicates that defect density increased significantly, $t(68.167) = -6.306$ and $p-value = 0.000 < 0.05$ (assuming unequal variances) therefore we reject the null hypothesis and accept the alternate hypothesis (H_{1_1}) that a decrease in the review speed

impacted the ability to detect defects.

We also checked to which extension the model delivered a reliable estimate of improvement. If the resulting defect density value for the pilot inspections was within the 95% percent range of the value predicted by the model, we would accept the model as a reasonable approximation of real process performance.

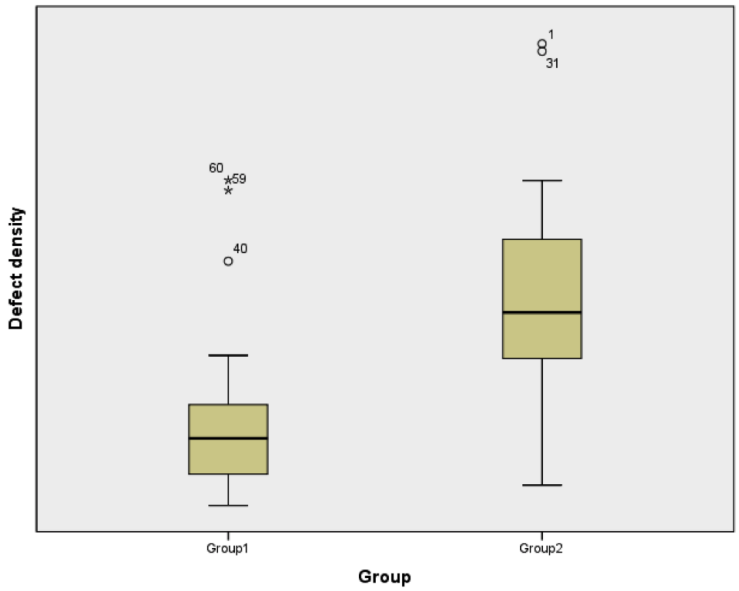

Figure 6.5: Defect density by scenario

	Levene Test		t-test for Equality		
Variable	F	Sig	t	df	Sig. (2-tailed)
Equal	4.692	0.033	-6.430	0.000	0.000
Unequal			-6.306	0.0000	0.000

Table 6.12: Test for equality of means and variances

We used the average review rate from the pilot, 215 LOC/hour and used the performance model to get the defect density estimate for that specific rate. A 95% percent confidence interval was computed for the resulting defect density. The top limit for improvement acceptable by the model implied a 165% increase in defect density. The obtained value for defect density in the pilot was an increase of 142%. Thus, the value obtained in the pilot study is within the 95% confidence interval of the estimated value by the model, leading us to conclude that the model provided an acceptable prediction of process performance.

2) Understanding the increase in process variability

To understand the increase in variability, the two controlled factors were isolated with the expectation that, at least, one of them would provide some explanation for such variability. We began by checking if variability could be explained by variation in initial code quality. We considered project as a proxy for code quality to study possible differences in initial code quality. For the second source of variation reviewers ability in detecting defects was analysed. One of the reviewers was a senior developer (R1), the others were juniors (R2 and R3). This classification translated their ability as developers and experience as reviewers. The senior status evidences both more years using the programming language and more experience in performing inspections.

To test the impact of code quality, a box plot of defect density by project is depicted in Figure 6.6. Applying an one-way *ANOVA* to assess significance in the two or more samples, the following result is obtained: $F(3,35) = 1.197$ and $p - value = 0.325 > 0.05$ (see Table 6.13). This indicates that differences in average defect density are not significant between projects.

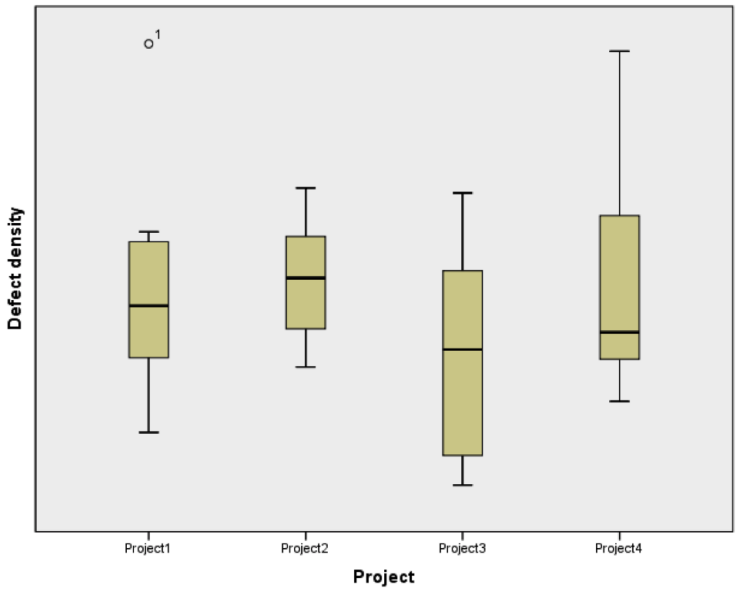

Figure 6.6: Defect density by project

The box plot showing the impact on defect density by reviewer is depicted in Figure 6.7. We applied the ANOVA test and the following result is obtained $F(2,36) = 4.620$, $p - value = 0.016 < 0.05$ (Table 6.14). This indicates the differences in the average defect density by reviewers are statistically significant.

A deeper analysis on the reviewers academic background allowed to verify that *R3* had above average performance as a student and accumulated experience as student in

Groups	Sum of Squares	df	Mean Square	F	Sig
Between	2264.417	3	754,806	1.197	0.325
Equal	22076.897	35	630.768		
Total	24341.314	38	–		

Table 6.13: ANOVA test for project means

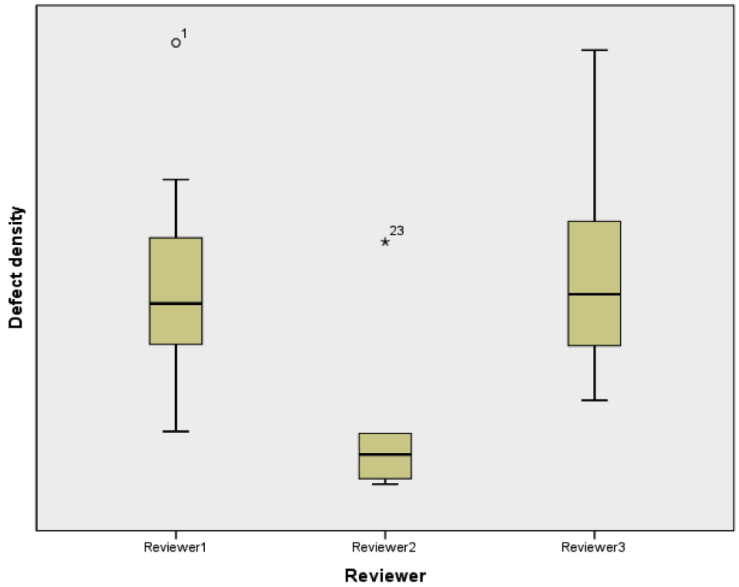

Figure 6.7: Defect density by reviewer

Groups	Sum of Squares	df	Mean Square	F	Sig
Between	4971.348	3	2485.675	4.620	0.016
Equal	19369.695	35	539.055		
Total	24341.314	38	–		

Table 6.14: ANOVA test for reviewer means

JAVA programming. This fact may justify the variation in performance of the reviewer and as result a source of significant performance variability in the pilot study.

Results packaging and interpretation

Based on the pilot results of the pilot we can argue that a decrease in the review

speed resulted in an increase (2.4x) in the number defects found by reviewers. This increase seems to be independent of possible variations in the initial code quality as different developers from different projects created the code subject of inspection and there is no significant variation between defect founds within projects. Also, it was possible to evidence an increase in variability of defect density resulting from inspections. The source of this variation was one reviewer participating in the pilot, leading us to believe the reviewer was the assignable cause for the increase in variation and not the treatment (decrease in the review speed).

Threats to validity

The main purpose of the pilot was to verify if a decrease in the review speed would increase the capability of the reviewers to defect defects. The results show that by decreasing the review speed resulted in a significant increase in defects found. However, having the pilot framed in a quasi-experiment this claim is threatened in several ways. The main thereat arises from the non-equivalence between the pre and post test groups. First, reviewers participating in both test groups were not the same and there is no way to isolate the impact of the individual capability of the reviewers in defect density. Also, the number of reviewers decreased in the post test group when compared to the pre-test group. This decrease in diversity does not help minimizing the impact of variation in reviewers capabilities. This mean that reviewers play a confounding factor in the dependent variable defect density. This is supported also by the outcome of the experiment where one reviewer had a significant different performance in detecting defects.

A second relevant threat in that code submitted for inspection by both groups was different. Code may influence the number of final defects found if there is significant differences in number of defects present, both in quantity and type. Although the impact of the variation of initial code quality is minimized by having different project and therefore different sets of code submitted for inspection, the increase in defects found may have resulted from a significant decrease in code quality in the post test group.

Other factors may also had an impact in the outcome of the experiment, namely construct validity threats. For the pilot, reviewers were aware that their reviews were part of an experimental study, when the pre test group performed inspections in an uncontrolled environment. Reviewers in the post test group may have behaved differently when compared to a completely uncontrolled environment.

Inferencing

Considering the pilot result and inherent validity threats lead us to conclude that a change in the review rate could impact positively the ability to detect defects, however reviewer ability may play a significant factor in the outcome of an inspection. Also, changes in inspections steps, namely not performing the review meeting and performing single reviewer inspections were considered to have a marginal impact in the outcome of the inspections.

Having this information and considering that one of the objectives of the pilot is to increase the confidence that an improvement will contribute positively to the overall objective of the improvement initiative, the decision was to consider that the decrease in review speed could result, with good confidence, in a decrease of projects cost of quality by allowing a more effective removal of code defects. Along with the process change, additional guidelines were identified, in line with expectations detailed in step *A1. Define problem or improvment opportunity*, namely that projects should plan an increase in effort invested in code inspections to ensure the same level of code coverage.

A5.1 Plan improvement deployment

The planning focused firstly on the tasks necessary to update the existing processes definitions affected by the improvement solution. Secondly, to plan the necessary training for the personal that performs the process and thirdly, to define a strategy to communicate the process change. Finally, criteria for evaluating the deployment phase was defined.

The update on the code inspection procedure document that included the inspection steps, was to be updated to include the new recommendation for review speed. The template used by the moderator when planning the code inspection was also to be updated to include a reminder for the new recommended review speed. Additionally, an activity at organizational level for monitoring organizational process performance needed an update to include monitoring the code inspection metrics. The update required adding new process performance baselines for monitoring and control of process performance by analysing data points resulting from project code inspections.

Training for the code inspection procedure was considered not necessary based on the small change involved and no training plan was defined. However, in the scope of the organizational process performance process, requirements for for using statistical tools were identified and that know-how on this topic was limited in the organization. As result, training was planned to address this gap.

For communication of the process change the strategy defined involved direct communication with active projects, namely, ensuring that project members responsible for quality were informed directly. Additionally, a dedicated channel for process quality

related topics was to be used to broadcast the processes changes to the organization.

Along side with the monitoring of the process, a task was planned to assess the impact of the process change. The task considered two steps, firstly, an evaluation would consider performance of the inspection process and secondly an assessment of the possible impact in the quality process performance objective of decreasing projects cost of quality. This task was planned to be performed on a yearly basis for a period of tree consecutive years.

The performance of the inspection process was planned to be evaluated by comparing defect density (measured in $defects/KLOC$) resulting from code inspections before and after the improvement. The criteria for evaluating the success of the process change was defined as obtaining a significant change in the defect density before and after the deployment (corroborating the pilot results). This evaluation was planned to be performed on a yearly basis, for a period of thee years. Secondly, an evaluation in the impact on cost of quality performance indicator should demonstrate a significant difference before and after the deployment of the improvement, meaning projects were positively impacted, by decreasing costs associated in correcting defective product during development.

A5.2 Deploy improvement

The deploy initiated by updating the identified process documents and templates as planned. The process change was communicated to running project to use the new review speed if code inspections were to be performed. The organization also was informed via a news update dedicated channel about the new recommended review speed for code inspections and the date when the change was to be put into practice. After the announcement of the process change, running and new project were to perform code inceptions taking in consideration the new recommendation for review speed.

In the scope of the organizational process performance process, information about code inspections collected at the project level was analysed at the organization level. Performance was monitored by plotting control charts of the review rate and monitoring defect density resulting from code reviews. Along with monitoring of process performance training on SPSS statistical package was provided to personal responsible for monitoring process performance.

The deployment phase lasted for 3 years until all planned deployment activities were fulfilled. At the end of each year the a formal assessment of the performance of the process was performed and the impact on the linked QPPO evaluated and reported.

A5.3 Evaluate improvement

The results described in this section refer to the last evaluation of the improvement project. Evaluation included an analysis of the changes in process performance after a 3 year period and an evaluation of the impact on the cost of quality performance goal.

1. Evaluation of process performance

The data set considered for evaluation resulted from code inspections performed by more than 10 project and more than 300 code inspections were performed. Table 6.15 summarizes the comparison between the descriptive metrics of process performance before the process change (year 2009) and after (year 2012). The average review speed changed from 860 to 206 LOC/hour and the variability also decreased from 533 and 161. The change in the procedure to decrease the review speed was effective and there was less variation on how core review were performed. However the variation on the defect density increased when compared to the 2009 baseline. Average values are not disclosed due to confidentiality.

Measure	Average (2009)	Std Deviation (2009)	Average (2012)	Std Deviation (2012)
Review speed ($loc/hour$)	860	533	206	161
Defect density ($defects/Kloc$)	–	18	–	45

Table 6.15: Code inspection process performance (2012)

An analysis was performed to compare if difference in the averages could be considered statistically significant. A statistical *t-test* for independent samples was used to evaluate the significance of this variation. The Levene test was computed to evaluate equality of variances for both samples (Table 6.16). The Levene test provided a $p-value = 0.000 < 0.05$ therefore, the variances are significantly different. The t-test than considered the unequal variances and the result indicates that the difference between averages of defect density is significant $t(160.110) = -6.051$, $p-value = 0.000 < 0.05$. The mean difference indicates there was an incrrease of 24.24 $defects/Kloc$ between 2009 and 2012 and with 95% confidence at least an variation of 16.32 $defect/Kloc$ is expectable. The box plot (Figure 6.8) depicts the distribution of defect density values in 2012 and 2009.

Secondly, an analysis was performed to evaluate the validity of the model identified

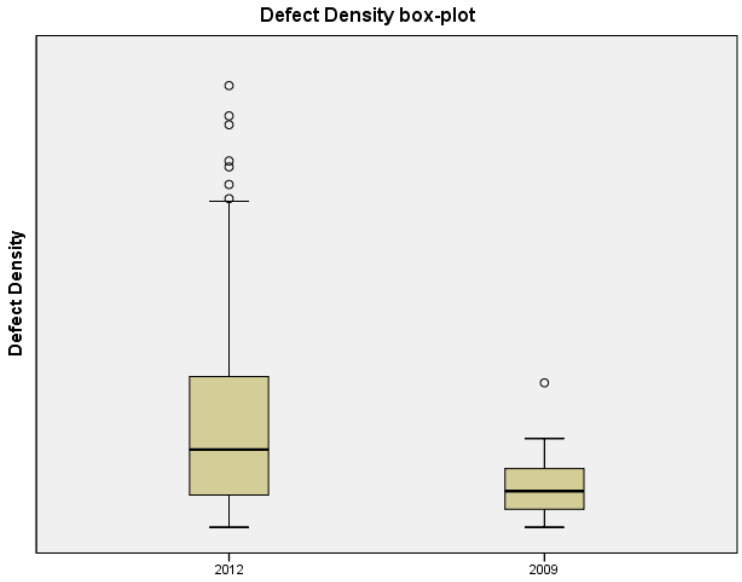

Figure 6.8: Defect density distribution in 2012 and 2009

Variable	Levene Test	
	F	Sig
defect density	25.944	0.000

Table 6.16: Test for equality of means and variances (2009 vs 2012)

t-test for Equality						
Variable	t	df	Sig. (2-tailed)	Mean Difference	95% conf	
					Lower	Upper
Defect density	-6.051	160.110	0.000	-24.24	-32.15	-16.32

Table 6.17: Test for difference of averages (2009 vs 2012)

in 2009. Figure 6.9 depicts a scatter plot of the new scenario after 3 year of data collection. The *(y-axis)* represents defect density resulting from code inspections and *(x-axis)* the review rate. Again, values for defect density are masked for confidentiality.

By analysing the scatter plot it is possible to identify that data points follow a similar pattern of the data points from 2009 model (Figure 6.2). However, the clustering of data between 0 and 500 $loc/hour$ is much higher. An attempt as made to iden-

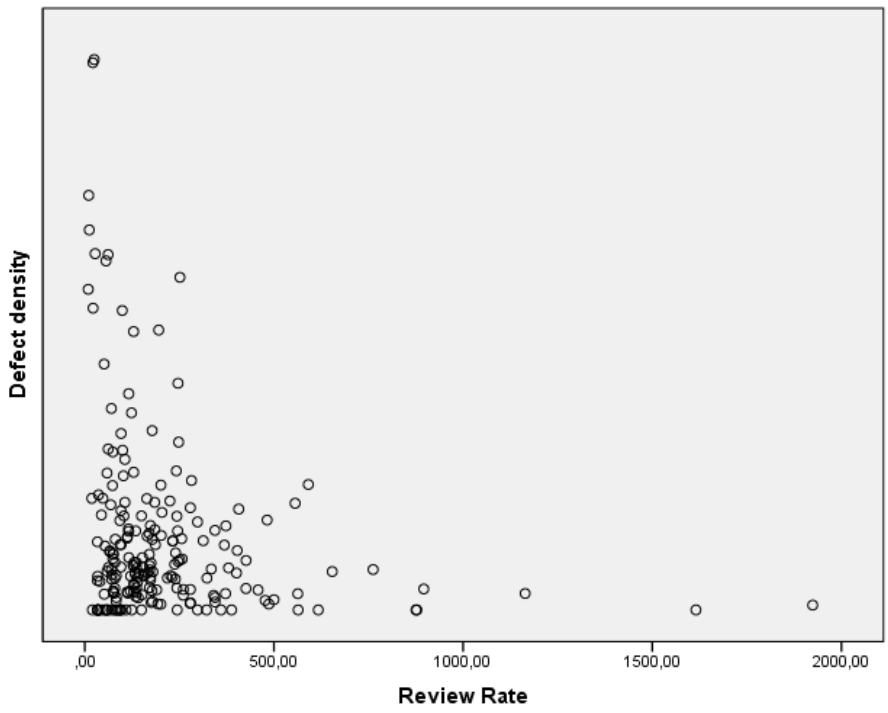

Figure 6.9: Scatter plot and regression analysis

tify the regression line that characterizes the relation between review speed and defect density. The first assumption required for a regression is that variables are normally distributed. The result of Kolmogorov-Smirnov non-parametric test on the normality of distributions of both variables was negative ($p-value = 0.000 < 0.05$). The absence of normally if confirmed by observing the histogram for defect density displayed in Figure 6.10.

The base assumption of normally of distributions is not met and therefore the fit provides no reliable mathematical relation between review speed and defect density. However for the purpose of characterize the relation between variables involved a fit was computed.

The fitting considered the *Inverse* curve with and without the constant and excluding or including inspection with zero defects result. Excluding the zero defects inspections represents that this outcome is considered a special case of a review which in this case is excluded in the model. The results are depicted in table 6.18. The highest $r-square$ is given by the inverse without constant for the data set with zeros defect inspections not considered. The $r-square$ decreases when the zero defect reviews are considered (all data), which is expectable as it increases variability.

2. Evaluation of impact in cost of quality

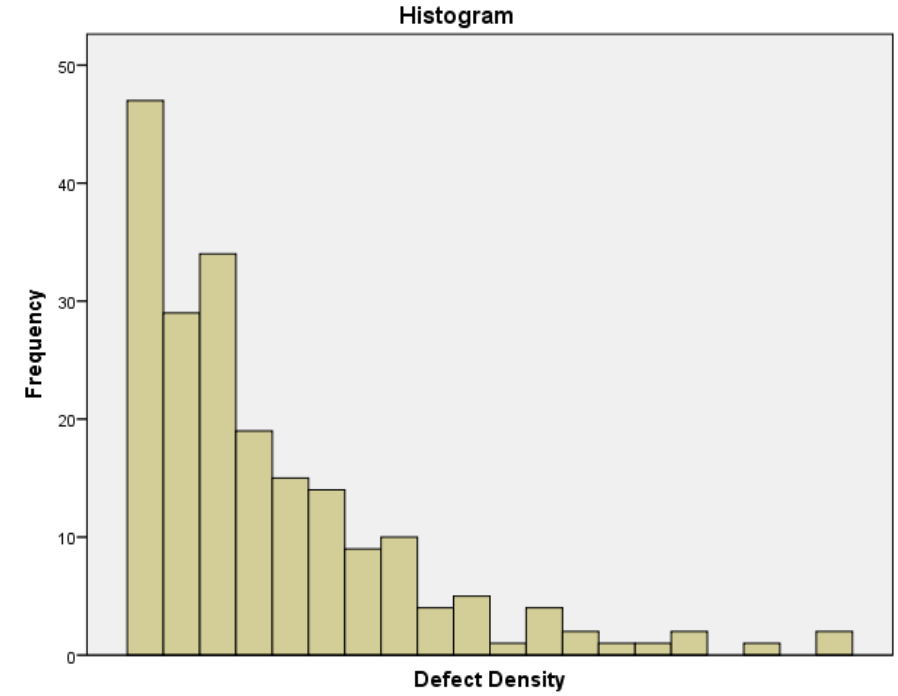

Figure 6.10: Histogram for defect density (2012)

Model	Formula	all data	zeros excluded
Inverse with constant	$y(t) = b_0 + \frac{b_1}{t}$	0.239	0.331
Inverse without constant	$y(t) = \frac{b_1}{t}$	0.466	0.550

Table 6.18: Fit models

The success of an improvement project is strongly linked with the ability to demonstrate a direct impact on the performance of the organization. CMMI-Dev stresses this fact in the scope of *OPM.SP 1.3 Identify Potential Areas for Improvement* specific practice, where a quality or process performance objective must be identified and linked to a improvement initiative. Additionally, specific practice *OPM.SP 3.3 Evaluate Improvement Effects* requires that statistical os other quantitative techniques are used to measure and analyse the progress in achieving the quality and process performance objectives.

Although it is generally accepted that in Software Engineering, finding and remov-

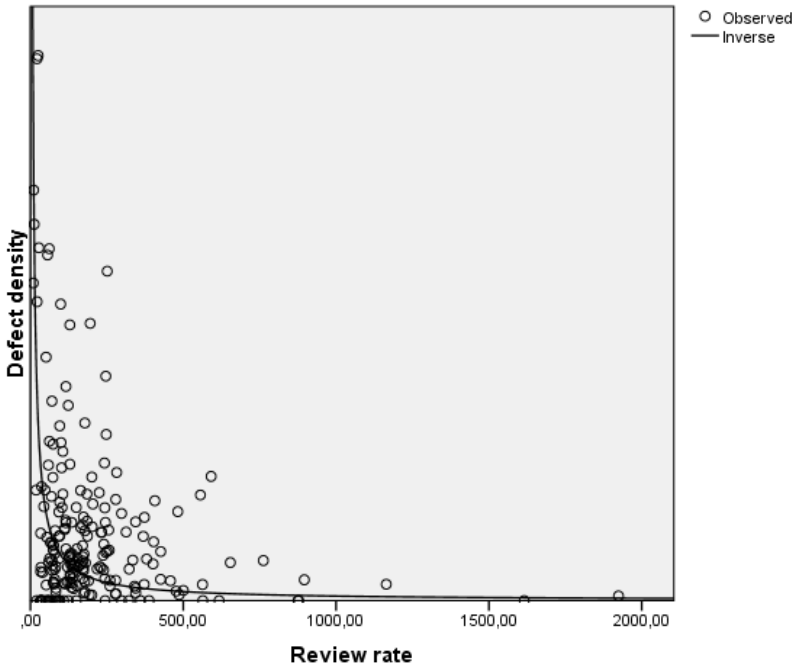

Figure 6.11: Inverse without constant for 2012 dataset

ing defects closer to the point of injection is beneficial, by allowing to decrease the effort required to remove product defects (internal failure cost), the overall cost of quality is impacted by wide range of factors.

The change in the inspection process was successful in improving the use of the appraisal effort in finding product defects. On average the number of defects resulting from code inspections improved. The second or overall goal required that a decrease in the overall cost of quality was achieved.

The evaluation towards the impact on cost of quality by decreasing the review speed of code inspections considered several metrics associated to cost of quality:

- Project cost of quality measured in hours of effort, considering the four categories of costs: preventive, appraisal, internal failure and external failure.
- The percentage of each effort category in the overall cost of quality effort.
- The percentage of cost of quality in the overall project effort (including development effort).

To measure and analyse the impact of decreasing the review speed in projects, these metrics were monitored to identify significant impacts on each of the measures against

the benchmark of the organization. Although the metric values are not provided due to confidentiality agreements this step is one of the most relevant in process improvement by assuring cause and effect of process improvement initiatives.

6.4 Analysis and discussion of results

This chapter documents a software process improvement initiative that focused on improving the performance of a code inspection process in an industrial setting. The initiative is analysed considering three distinct stages, first, identification of a process problem and understanding of process performance, second, identification and validation of the improvement solution in the scope of a software experiment, and lastly, deployment and evaluation of the solution in the organization.

A potential improvement was identified in the code inspection process that motivated the initiative. Inspections were being performed to fast and it was believed that the speed was having an impact in the effectiveness of the inspections. The analytical paradigm to process improvement requires a strong understanding of current performance before an improvement is introduced. This motivated the creation of process performance baseline to provide a clear understanding of the code inspections process. Additionally, it was possible to derive a performance model of the process. The ability to detect defects decreases as the speed of the review increases and this association was modelled by a process performance model. The data used to derive the performance model and baseline was collected from existing reports of performed code inspections. This meant that no control existed on the execution of the inspections used to characterize process performance.

For this stage there are strong requirements for standardisation of execution of the process under analysis and on measurement capabilities to establish and execute process measurement. Additionally, knowledge in statistics is required to establish performance baselines and specifically to derive process performance models.

Based on the analysis of the performance baseline and performance models an improvement was identified. The improvement impact on process performance was estimated based on the performance model and a rationale for the potential impact on higher level project quality goals was identified. The solution was tested in a controlled experiment to validate if it's effectiveness in delivering the expected improvement. From the experiment it was possible to conclude that there was a significant change on the the ability to detect defects when the review speed decreased to values around 200 $LOC/hour$ and thus a decision to deploy the improvement to the entire organization was made after considering the validity threats of the experiment. From the pilot experiment it was also possible to identify that below 300 $LOC/hour$ the review

speed does not clearly determine the ability to detect defects.

At this stage of the improvement initiative there are two relevant factors to consider. Firstly, the ability to link the expected improvement in performance to higher level quality or performance goals and secondly the ability to conduct software experiments in an industrial setting. The first factor stresses the measurement capabilities of the organization and additionally the ability to define meaningful operational high level goals of the organization. Only when these higher level goals are available and monitored it is possible to rationalize about any possible impact of process changes in high level quality or performance goals and thus drive effective process improvement initiatives. The second factor relates to the ability to plan and execute software experiments in line with the experiment design constrains so that inference is not impaired by validity threats. In an industrial setting there are strong limitations ensuring the desirable environment to validate a change to a process.

In the third stage of the improvement initiative the process improvement was deployed to the organization and data was collected on the execution of the updated inspection process. The improvement impact on performance corroborated the result from the pilot experiment. A significant increase in the ability to detect defects was achieved when the average review speed decreased to the recommended speed. Additionally, the impact of the change on higher level project goals was monitored and evaluated. In the perspective of inspection process performance, data collected from the deployment phase shows that for review speeds below 500 $LOC/hour$ the review speed impact on defect detection ability is less conclusive. A *'shotgun effect'* is present which may indicate that the determinant factor of defect density for review speeds below 500 $LOC/hour$ is other than the review speed.

Once again at this last stage the evaluation of the improvement solution highly demands on measurement capabilities. Process execution data needs to be collected systematically and additional data for monitoring higher level quality and performance objectives also needs to be available. Additionally, the deployment stage of an improvement initiative demands greatly on change management capabilities to guide a successful adoption of a process change by the organization.

6.5 Conclusion

Software process improvement initiatives in line with the analytical paradigm highly stress the capability of an organization to implement measurement programs in software engineering. Measurement is present in all stages of a software process improvement initiative. Additionally, analysis of process performance is only possible if a process is performed systematically. This implies that processes subject of improve-

ment are well established and executed by the organization. The analytical paradigm also stresses the need to have knowledge on statistics. Being able to apply statistical knowledge and tools is required in all stages of a software process improvement initiatives, namely when establishing baselines and performance models, when validating results of software experiments and when evaluating the impact of deployment of a process change.

Success of software process improvement also requires that improvement efforts are driven by higher level organizational business goals. These can take the form of quality or process performance goals. Only when improvement efforts are clearly linked to business goals there is the possibility to compare the effectiveness of different improvement solutions and validate the real impact on organizational performance.

An industrial setting is characterized by limitations in ensuring desired levels of control of the operational environment and thus conducting sound software experiments is challenging. Resources are most of the times limited, whether by opportunity or by number. Opportunity issues occur when the desired resources are not available, e.g., the need to have a project using a specific programming language, having a experienced software developers available or having enough projects available for performing a specific software life-cycle phase within a desired time window. Limitation in number is when the desired number of subjects are not available or is not possible or feasible to have the minimal number of repetitions for a specific experiment. As result software engineering experiments in an industrial setting are most of the times limited by threats to validity that limit the soundness of possible claims resulting from engineering experiments. This is highly relevant if the goal is to derive solid software engineering knowledge. However, the focus of software process improvement is in fact improving the process and experiments provide a mechanism to decrease the risk of introducing changes that may fail to have the desired impact on performance.

Software experiments, even if limited by some validity threats, are preferable to the scenario where no experimentation is considered, however there is a cost associated to experimentation. In the end a decision to perform an experiment is one of a trade off between risk reduction and cost.

Chapter 7

Modelling Large Software Process Systems

Contents

Improvement models typically prescribe a wide set of practices and the decision to adopt such models typically results in an Organizational Standard Set of Processes that are difficult to manage. Abstraction is a technique used in Computer Science to manage complexity of systems. In this chapter a meta-model is proposed to support the definition of process architectures along with a set of transition rules, to refine architectures into low level, ready to enact, process specifications by allowing the transition between two levels of abstraction. A demonstration of the use of the meta-model is documented for creating a process architecture along with the transition to low-level process specifications.

7.1 Process modelling

Software development organizations adopt improvement models generally to improve overall performance or to address specific business goals. A recent trend is the simultaneous adoption of more than one improvement model [SKMM08c]. The goal is to obtain the cumulative added value of each model onto a single environment. Adopted models do not only consider software engineering related practices but also practices related to non-engineering disciplines, like governance, quality assurance, project and process management.

A result of adopting several improvement models is an increase in the number of best practices performed by the organization that inevitably result is an increase in the number of processes and activities performed, *e.g.*, if one considers a full implementation of CMMI-Dev for Development [CK06] or ISO12207 [II08] combined with Information Technology Infrastructure Library the number of expected processes is considerable.

As a system of processes grows in size, properly managing process dependencies becomes a necessity. If these dependencies are not documented properly it becomes difficult to grasp the overall work flow of the system of processes, jeopardizing effectiveness and efficiency of the overall system. The system literally becomes too complex as result of the number of dependencies between entities in the system. Additionally, as system evolves by adding new processes or as result of process evolution it can easily result in process inefficiencies if process dependencies are not managed properly. Managing a complex system of processes requires an approach that not only considers modelling low level process details but also modelling at higher levels of abstraction.

Modelling large process systems is a fairly recent subject. Gruhn and Wellen address the need to model processes at different levels of abstraction to deal with complex process models [GW01]. Barreto *et al.* [BDRM10] also proposed a model to facilitate reuse in defining low level process definitions. The benefits of having a process model are several, aside providing a representation of activities to be performed in the organization. A process model facilitates process guidance and supports enforcing and partial automation of the software development process [CJ99].

Research in software process modelling has focused on defining PML (Process Modelling Languages) to support the implementation of process centred engineering environments [CJ99]. According to Osterweil [Ost87] and Humphrey [Hum88] developing a process should consider several phases of development where *process design* and *process implementation* should be considered as process modelling/engineering phases.

Process implementation involves the specification of low-level process details to

allow a process to be enacted. Process design involves developing a process architecture with the necessary process abstractions to allow, consistently relating and incorporating process elements, supporting reuse enhancement and tailoring of processes [HK89] [WH92]. Each of these phases deals with process modelling at different levels of abstraction.

An extensive study by Brendaou *et.al* [BJGB10] concluded that several PMLs were proposed to support process implementation but had limited adoption due to their complexity and inflexibility. However, recent proposals have gained increasing interest not only by the research community but also by industry, namely SPEM (Software and Systems Process Engineering Meta-model), Business Process Modelling Notation [BPM09] and Little-JIL [Wis06].

Research in process design has focused on defining process abstractions to support the definition requirements for process architectures. Requirements for modelling processes at higher levels of abstraction (Design Phase) differ from modelling at a lower level (Implementation phase). It is likely that each phase requires different modelling languages but maintaining formal dependencies between specifications is desirable for process the overall modelling exercise. To date approaches for documenting process models at higher leave of abstraction do not specify how high level modelling entities are mapped to low level process definitions.

Research motivation and goals

CRITICAL Software S.A. has a multi-model OSSP (Organisational Set of Standard Processes) that is compliant with several improvement models like CMMI-Dev, ISO9001, Allied Quality Assurance Publications (AQAP) among others. Adopting multiple models was a strategic decision to address business opportunities and improve performance. The motivation to develop a process architecture came out firstly to address the need to deal with the increasing size of the OSSP. CMMI-Dev v1.2 defines 175 specific practices and ISO12207 a total of 124 Activities and 404 Tasks. Orchestration and management of a system of such magnitude is a challenge and an *ad-hoc* approach to manage such complexity is not effective. As the OSSP grows in the number of processes as improvement models are adopted and evolve over time becomes difficult to orchestrate the set of best practices adopted. This fact motivated the need to develop a birds eye view of the OSSP in order to manage the increasing complexity os the OSSP in line with the concerns describes by Gruhn and Wellen [GW01].

We address the issue of modelling complex systems of processes and explore the possibility of high level process specifications be used to develop low level process implementation where semantic constructs used in high level design are used to support the design of low level process definitions systematically, or:

Can constructs used to develop high level process specifications (design phase) be used to inform the development of low level process descriptions (implementation phase) and thus maintain traceability between process modelling specifications?

This chapter documents and discusses the application of a process modelling approach that allows to define processes architectures followed by a refinement to actual process implementations. A process design meta-model is defined to develop a process architecture and a set of rules are proposed to guide the transition between design and implementation abstraction layers.

The meta-model to support the design phase is based on an extension of Unified Modeling Language (UML) 2.0 Component Diagrams and for the implementation phase Little-JIL process constructs are used to define low level process implementations. The method is based on a set of transition rules that guide a systematic transition from design to implementation phases by mapping UML components constructs to Little-JIL process constructs.

7.2 Related work

Osterweil and Humphrey relate engineering of software processes to Software and Systems Engineering [Ost87][WH92]. They argue that process development should include phases of requirements specification, architecture and design, and then process implementations providing ready to execute work instructions to software developers. Albeit a theoretical analogy to systems and software engineering was identified, research in the field of software process engineering has mainly focused on the process implementation phase and a clear insight of what process design should consider is a recent research subject.

Several authors have developed approaches to support software process modelling at higher levels of abstraction. Basically, structuring concepts that organize process related concepts are proposed to support process implementation. Zhao *et.al.* [ZCL05] argue the application of agent-based technology to support software process modelling. They consider a software processes as a collaboration of groups of process agents that are responsible for managing software development activities. Modelling is executed by combining process agent profiles that hold information related to activities, participants and artefacts involved in the process. The modelling perspective is one of collaboration between process entities.

The concept of *Process Landscape* is described by Gruhn and Wellen and focuses on modelling complex processes at high levels of abstraction [GW00].The underlying

context is organizations carrying out parts of a process with different levels of autonomy and distributed among different locations. A process landscape considers process clusters as an abstraction to group processes at the same level of detail. The landscape predominant view is of logical dependencies between clusters of processes and interfacing is defined as a first class modelling entity. However, it does not consider control flow or behaviour information between clusters. Along with the concept of process landscape, a set of attributes for describing distribution properties is proposed. These are considered useful to analyse complexity and efficiency of distributed processes.

A recent trend in software process modelling are SPLs (Software Process Lines) which are motivated by the need to have processes more aligned with development needs [MVW06]. It targets highly heterogeneous project environments where development contexts may demand different development practices. In software process lines modelling process commonalities and variability assumes central role to support process reuse and adaptation. The concept is comparable to the well-established concept of Software Product Lines. In this line of research Barreto *et al.* [BDRM10] proposed a model of process related concepts to be applied in the context of developing a SPL. They define and relate concepts like process architecture, process element, process component, process feature, among others, to capture process related information. Commonality and variability is modelled at the process feature level and at the architectural component level. Features are associated to process components with predefined variation types to decide if a component is present or not in an enactable process. Abstract process components are used to represent alternatives to component implementation. Concrete components are associated to abstract components specifying which alternatives can be adopted.

Münch [MVW06] describes the concept of *Process Line Architecture* has a process structure to model commonality and variability in a collection of processes . The emphasis of the work is on describing how to create a set of processes by comparing alternative (variability) work-flow elements to core process elements (commonalities). Alternative work-flows are mapped to features diagrams that help with the development of project-specific processes.

SPEM version 2.0 is a Software and Systems Process Engineering Meta-model and defines structural concepts for software and system processes development. Aside being a process modelling language to be used in the implementation phase, defining basic modelling process concepts like Activities, Tasks, Work Product, among others, it provides a conceptual framework to support process definition based on the following scopes: libraries of reusable method content, systematic development and management and growth of development processes, configuration of a process framework customized for project needs and process enactment, which can be considered design supporting elements. It suggests a clear distinction between reusable content and

processes definitions. Processes should be defined by reusing methods from reusable method libraries. Variability is supported at implementation phase by defining different types of variability mechanisms.

The perspective in high level process modelling approaches is that process abstractions are used to manage coherent sets of process related modelling entities, ignoring low level modelling details. Higher order process elements like process architectures and process components are defined and expected to be used along with basic or primary process modelling elements. Recently, expressing process commonality and variability is considered relevant to align process implementation with development needs.

None of the approaches presented describes clearly how their higher order concepts are linked to low level process modelling entities. This research work addresses this concern by describing a software design process based on a transition mechanism between process design and process implementation specifications. A conceptual model is used to support the creation of process design specification and Little-JIL process programming language is used to specify process implementation specifications. A set of transition rules is applied in the transition from design to implementation phases.

7.2.1 Process design meta-model

Humphrey defined process architecture as:

'A process architecture should provide a supporting infrastructure relating process abstractions to support the activity of process engineering. It should facilitate consistently relating and incorporating process elements, support reuse, enhancement and tailoring of processes.' [WH92]

An architecture is highly influenced by stakeholders of the system that the architecture is expected to yield. Each stakeholder has different concerns that influence how they envision the system though its architecture. Our perspective in supporting the development of process architecture is the one of a process engineer. The process architecture shall facilitate the isolation and abstraction of detail of self-contained development practices. The architecture should support the process engineer in a more effective development and maintenance of complex systems of process.

In line with Humphrey definition we developed a process design model as an extension of UML 2.0 meta-model Component Diagrams that provides a set of process constructs to be used for defining a process architectures. The model uses *process components* as the main entity for developing an architecture. Process components hide low level process details to focus on the interaction between components. The

model proposed is in line with previous research where components are entities as an abstraction of abstract low level process details. *Process components* are similar to Grumh and Wellen *clusters* that organize logically functional properties desired for the system of processes or similar to *ProcessComponents* proposed by Barreto *et.al.* [BDRM10].

The novelty of the meta-model is the reuse of UML 2.0 Component Diagrams that allow to specify *interfaces* between process components. Interfacing assumes central relevance in our proposal as it provides a mechanism to model process components interactions. Interfacing also allows declaring desired/required process interactions assuming that an interface is a declaration of a need for information flow between architectural components.

Figure 7.1 depicts the class diagram for the process design meta-model. Highlighted classes represent the extension of standard UML 2.0 component classes. The elements of the model are described next.

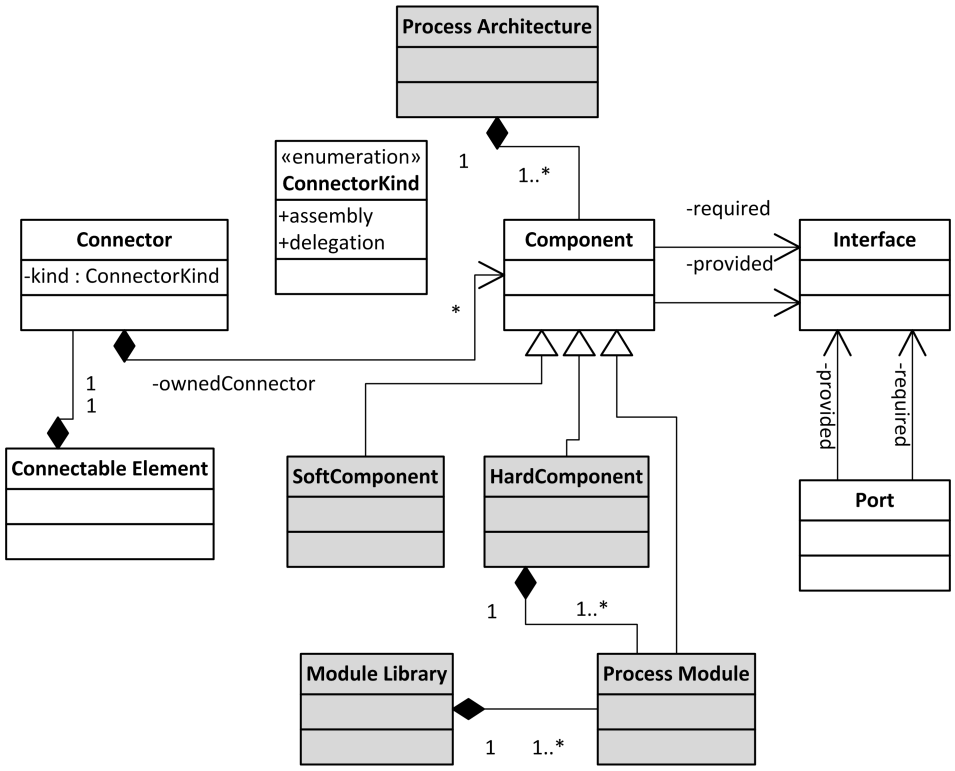

Figure 7.1: Process Design Meta-model

Module Library and Process Modules

A Module Library aims to provide a repository of Process Modules. A Process Module is conceptually analogous to SPEM Method Libraries of reusable Method Content. The goal is to support management of textual descriptions of practices used in software development. We adopt the perspective of SPEM specification where software processes should be defined, mostly, but not only, by combining existing practices from a method repository. Process modules shall be documented by process engineers or process performers as news development practices are adopted or evolved as changes are required. The goal is to have a catalogue of development practices performed in the organization applicable to different development scenarios that are self-contained and thus be managed independently. In line with Humphrey definition, a module library is expected to assist process reuse in the process development process.

Process Architecture

A *Process Architecture* is composed of *Components*. A distinction is made between *SoftComponent* and *HardComponent*. Soft components are used to declare architectural entities with no semantics defined e.g., required functionality is yet to be defined to satisfy identified process needs (e.g., satisfy an expected quality practice or goal) or its implementation is assured by external parties, e.g., outsourced functionally. A hard component has its semantics defined by combining existing process modules from a module library. Hard and soft components group logically related process modules by means of *Ports* and *Interfaces*.

In defining process a architecture an *Interface* is a representation of a desired interaction between two process components. The *Port* acts as a design entity that facilitates the specification of component interfacing. Interfacing can be characterized as information flowing between two components when these are part of a process architecture. The detail of the interaction is specified when process components are refined and implemented by means of a low level process modelling language.

A *Port* represents an interaction point between a component and its environment or between the functionality realized by the parts of that component (represented as a contract by means of that port). A component is encapsulated at a port, therefore ports allow specifying the component disregarding of it's environment.

An *Interface* specifies the contract a port represents for a component or what the component somehow (if no port is considered) is expected to realize in order to provide additional functionality. Connectors may be of two types: delegation or assembly. A delegation connector links a port of a component to the parts of that component that realize functionality that component represents. That port is the representation of a contract for that functionality the component must internally realize. It may be the case

that in between that port and those parts of the component some ports are traversed. In order for operation requests and events to be handled by parts of a component, ports and delegation connectors shall be modelled. An assembly connector links components and represents the provisioning of services of one of those components to the other, which requires them.

7.2.2 Transition rules

Along side a design model a set of transition rules are defined to assist the process engineer in refining a process architecture to an actual process implementation. The transition rules supports a systematic translation of interfaces between process components. This is possible by using Little-JIL process constructs that support the translation of an *interface* declared between process components of an architectural specification to low level process coordination constructs that describe how the interface is implemented. Both the model and the transition rules are summarized next along with a process design approach that integrates both the design meta-model and the transition rules.

A transition between process development phases requires that outcomes from one phase are used in delivering outcomes of the succeeding phase. A process architecture is expected as an outcome of the design phase. The implementation phase shall deliver a low level detail process definition by refinement of process modules from a process architecture.

PML for the implementation phase

A major concern in was which PML is most suited to implement process modules. A Process Module should support process definitions at the lowest level of process detail. Defining the semantics of a Process Module requires the use of a PML to define methods capable of describing most common elements of process definitions. The concept of a Process Module is similar to the SPEM MCP (Method Content Package) proposed by the SPEM specification. We follow most of the requirements set by the specification for the definitions of methods, namely: be reusable in the context of process instances definition; provide step-by-step explanation on doing units of work, have a clear purpose and a defined goal, and be independent of when in a project life cycle should it be used.

SPEM MCP provides adequate concepts for defining methods, however it does not include behavioural semantics and information on how these steps are coordinated or organized is not specified. In the perspective of refining a process architecture, where process components and their interfaces assume central relevance, having behavioural semantics built-in in process definitions should increase understandability of process

definitions. Little-JIL is an agent coordination process programming language that focuses on modelling behaviour of processes. A comparison between SPEM MCP process modelling elements and Little-JIL process programming language constructs is depicted in Table 7.1 (on page 190). The table compares the semantic constructs of Little-JIL and MCP with the purpose of identifying which provides the most suited set of modelling constructs for process modelling in the context of this research.

Little-JIL [Wis06] is defined as an agent coordination, visual based, with a formal graphical syntax process programming language. It implements sharp separation of concerns by separating internal specification of how agents carry out their work and their coordination. Little-JIL focuses on a three part strategy to address complexity and ease of process programming, namely:

1. **Coordination** - process requirements related to coordination of activities and agents

2. **Abstraction** - defines appropriate abstractions and language constructs capturing important coordination aspects of software processes,

3. **Visual** - provides visual and rigorous representations to aid adoption and understandability.

The basic semantic construct of Little-JIL is the *Step* (see figure 7.2). A step represents a unit of work and connected to each other through the interface badge. Each subset is a refinement of the parent step unit of work, when no more refinement is desired *Leaf* steps should contain the ready to execute instructions to execute the unit of work. Along with the refinement, information flow is coded visually using parameters passing. Coordination is coded using *badges* for control flow, *pre-requisite*, *post-requisite*, *continuation* and *exception*. A Little-JIL process program resembles a work breakdown structure where each step is associated to an agent. However, it does not specify a data model for agent, parameter and resources description or provides information on how to carry out basic units of work called *Leaf Steps*. This information must be maintained aside the visual representation of the process. Additional semantic construct are provided by the language and detailed information can be found in [Wis06].

Three Little-JIL semantic constructs are particularly useful in supporting the transition from process architecture to actual process implementations. *Reactions*, *messages* and *channels* are described next along with the explanation how they support the transition.

Reactions provide a mechanism to respond to the arrival of messages to Little-JIL steps. Often, performing a specific set of practices outside of the scope of what is

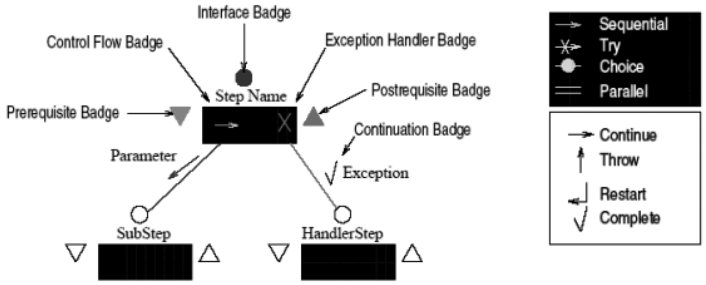

Figure 7.2: Little-JIL Step Process Construct

being executed is required. Reactions allow combining pairs of process modules by attaching the root step of a second module to any step in the hierarchy of steps of a first module. *Reactions* along with *Messages* allow a very precise identification of what is occurring between two separate sets of steps. Both the originating step and the event that triggers the reaction are identified along with the new set of steps (process tree) expected to be executed when the event occurs. This is possible as Little-JIL processes are specified as a decomposition hierarchy of steps.

Reactions and *Messages* can be used to implement the concept of *Interface* in Figure 7.1. When an Interface is declared between two components it indicates that interaction between two process modules is expected. When refining an interface declared between two components at the architectural level to a Little-JIL process implementation, a message can be used to declare an event associated to the desired collaboration between process modules. The *message* is declared on a Little-JIL process step (actually on the arc that links the step to its parent) that requests the interaction between steps. Additionally, a *reaction* is declared in the parent step to signal that the step should initiate the reaction associated to the message declared.

A Little-JIL *Channel* provides a communication mechanism not tied to the hierarchical structure of Little-JIL. It allows data-centric synchronization and support communication of potentially parallel or independent threads of execution. *Channels* are used in the context of components interfaces to allow communication between two independent Little-JIL processes, e.g., outputs of a first step are used as input to a second process step. *Channels* provide mechanisms to combine process modules with minimal change to their structure, only the participating steps in the interface are subject to changes to realize the desired interface between modules.

Messages, *Channels* and *Reactions* support the transition from an architecture to a implementation level and modelling behaviour is assured when a process is implemented as a refinement from an architectural specification. Also, the hierarchical structure of Little-JIL process programs facilitates module integration. The hierarchy of

steps entails different levels of detail which may be used to establish links between independent processes at the desired level of process detail. In our perspective, the ability to specify behaviour and the hierarchical structure of little-JIL programs provide significant advantages when compared to other process descriptions languages. The use of these Little-JIL constructs in the refinements of a process architecture to a process implementation is summarized in the following transition rules:

- **Rule 1** - A process module is implemented defining a Little-JIL process program. When a process module is connected through an interface to another process module the origin and target steps of both process programs are identified.
- **Rule 2** - A *reaction* arc is declared between the parent step of the originating step and the root step of the connecting process tree.
- **Rule 3** - A message identifies the event associated with the expected collaboration between process modules. The message is declared on the process step (actually on the arc that links the step to its parent) that requests the collaboration and in the reaction arc linking both process trees.
- **Rule 4** - A channel is declared in the parent step of the originating step to allow the communication of parameters between process trees. Parameters to be communicated are identified and the flow is declared in the arcs of communicating steps by binding operations available in Little-JIL.

Based on these transition rules an example is described next, where a process component from process architecture is implemented using Little-JIL.

7.3 Software process design and implementation demonstration case

CRITICAL has an OSSP compliant with several improvement models and one requirement of CMMI-Dev at level 3 is that processes need to be documented. CRITICAL organisational processes are aligned with ISO15504 [ISO04a] structure. Each process is defined by using a template that considers definition of *activities*, each including *inputs*, *outputs*, *roles*, *purpose* and detailed description of *steps* to be executed. Additionally, events are identified that provide information related to when the activity is to be performed during the process life cycle. Finally, tailoring guidelines are provided to be applicable when needed and this information is rather informal.

A shortcoming in this process modelling approach is related to orchestration or behavioural modelling. Processes are defined with ready to execute instructions but

information related to when activities need to be performed in the overall development scenario is not explicit.

Basically, activities are coordinated through event related information that is mostly associated to project life cycle milestones. This basic coordination information does not provide full description of activities interdependencies and as the system grows in size this becomes an issue to understand activities interdependencies.

This limitation e aggravated by the size of CRITICAL OSSP that is above 187 formally defined activities belonging to several processes. Understanding of the interdependencies of these activities becomes vital to assess the effectiveness of the system as a hole. CRITICAL has a large set of low level process descriptions but the information about their orchestration is limited. The following subsection describes an example of developing a process architecture that originates detailed process implementation descriptions.

7.3.1 Developing a process architecture

Using the process design meta-model and the set of transition rules, a process design approach is depicted in Figure 7.3. It consists of 3 tasks that are expected to deliver a *Module Library*, a *Process Architecture* and finally an *OSSP implementation*. The tasks *Develop a Module Library* and *Develop a Process Architecture* are independent and can be executed concurrently and provide outputs that are input to the task responsible for delivering the OSSP implementation.

Task: Develop a Module Library

The purpose for having a Module Library is to provide a catalogue of reusable development practices. Under an area of concern, practices are defined to handle situational development needs, e.g., measurement and analysis is necessary at different project life cycle phases. Another example is project management practices with recurrent practices during the software development life-cycle.

The Module Library is created by analysing existing organizational processes and activities. Typically a process engineer in the role of a process owner is responsible for defining related process modules. Defining categories to group activities definitions related to same areas of concern is desirable to organize logically the repository. In this example CMMI-Dev process areas were used to help group related process modules. The Module Library shall maintain detailed descriptions related to core Little-JIL semantic constructs, like *Leaf Steps*, *Resources* and *Parameters* to complement visual process coordination information.

A process design approach based in creating a process module library considers

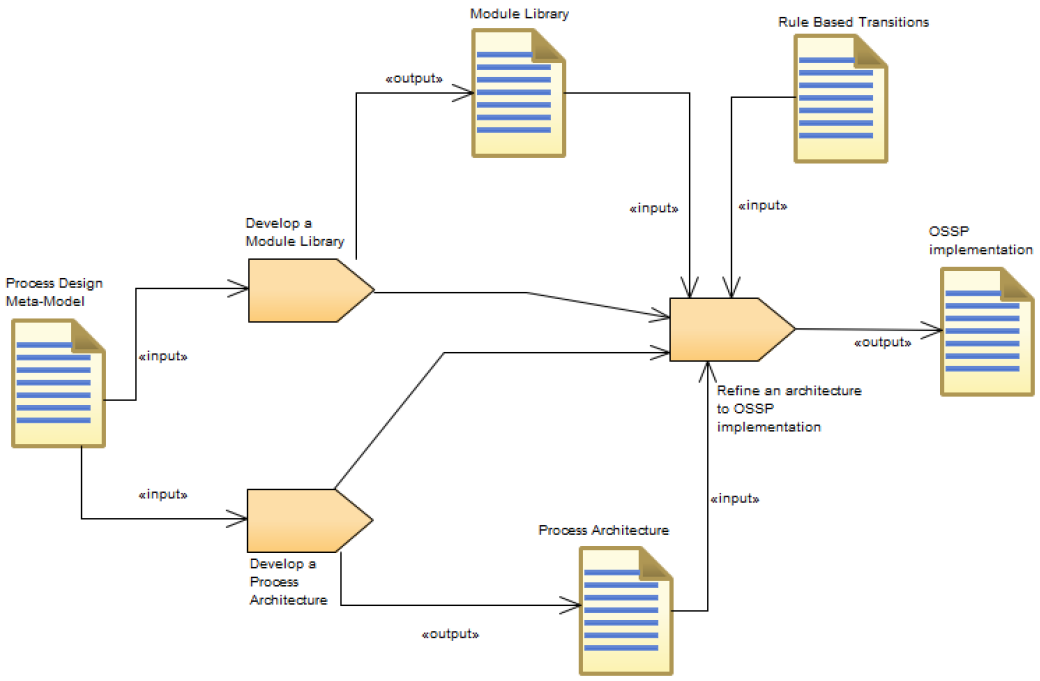

Figure 7.3: Process Design Process

development practices as a collection of process modules which are descriptive of specific situational development practices. As an example the measurement and analysis process is specified as a collection of process modules that defines precisely how measurements and analysis practices should be performed, e.g., modules for measurement and analysis for code reviews practices and for project management practices are defined independently.

Using Little-JIL in detriment of custom process definitions templates implies that activities are now codified as process programs and coordination information about activities becomes now explicit. Processes formerly defined using the template are generally codified in more than one process module. The decision to break in more than one process modules depends on the detail and context of use of a process.

A good example is the measurement process previously defined as a set of loosely uncoordinated activities now gives origin to a set of process modules that can be interfaced with other process modules when desired.

Figure 7.4 depicts this vision, where process modules of a Module Library are organized by areas of concern aligned with CMMI-Dev process areas. For each area, several modules are defined to address specific development needs, namely: *Code Review Measurement*, *Configuration Management for General Documentation*, *Documentation Management Guidelines*, *ADA Programming Guidelines and Code Reviews* are examples of practices carried out in a project context. The driver to define process

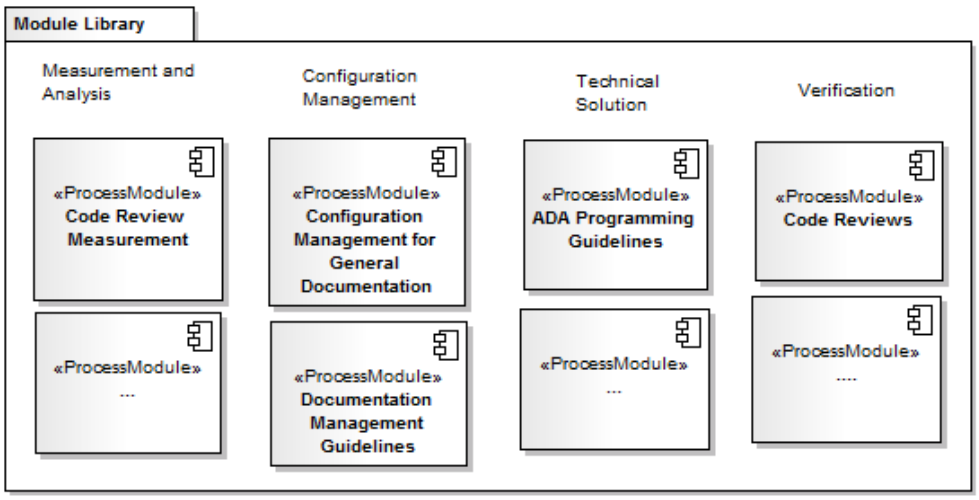

Figure 7.4: Module Library aligned with CMMI-Dev Process Categories

modules is to capture information by documenting how practices are being performed by projects in specific situations. This practice is aligned with situational method engineering principles applied to information systems [BKKW07]. Situational method engineering has been reported as suitable for software engineering, namely to support software process improvement [HS06].

Figure 7.6 depicts an example of a process module defined using Little-JIL. It shows how the step hierarchical decomposition of Little-JIL is used to describe a Code Review process module (information related to agents, resources and parameters is omitted). Additionally, describing modules using Little-JIL allows having behavioural information present in module descriptions. This is possible by using the hierarchical *Step* refinement and e.g., the Sequencing ($\rightarrow$) semantic construct for step coordination.

Descriptions related to *Leaf Steps*, *Agents*, *Step*, *Parameters* and *Resources* are not detailed in the visual representation of Little-JIL. Instances of these classes should be kept in the Module Library. In the class diagram of Figure 7.5 the type of dependencies between Little-JIL constructs are represented. A *Step* is associated to *Agents*, *Parameters* and *Resources*. Additionally, *Leaf Steps* are a special kind of *Step* and hold detailed descriptions on how agents should perform the non-devisable units of work. Figure 7.6, *Choose Moderator* and *Select Reviewer* are considered leaf steps.

Task: Develop a process architecture

An architecture can be developed by using a bottom-up or top-down approach. A top down approach considers analysis of commonalities and variability of a domain to derive process architecture. A step by step description of a top-down example is given by [Was06]. A bottom-up approach to process modelling is based on identifying

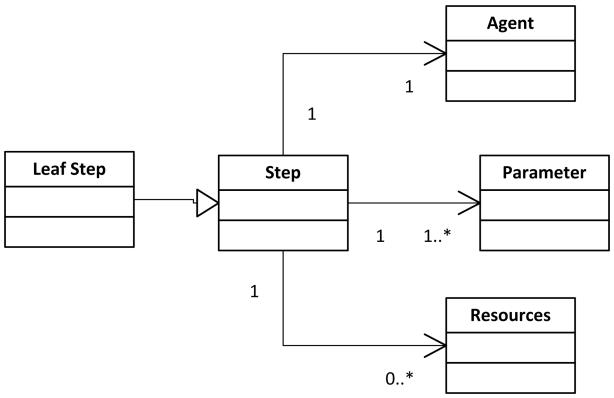

Figure 7.5: Class Diagram relating Step related entities

organizational practices that are common to different software development needs and then define alternative work-flows to address these specific development needs. In this view, process modules from a Module Library are combined to develop high order process entities like core process components which may be further extended with other specific process components to address specific development needs.

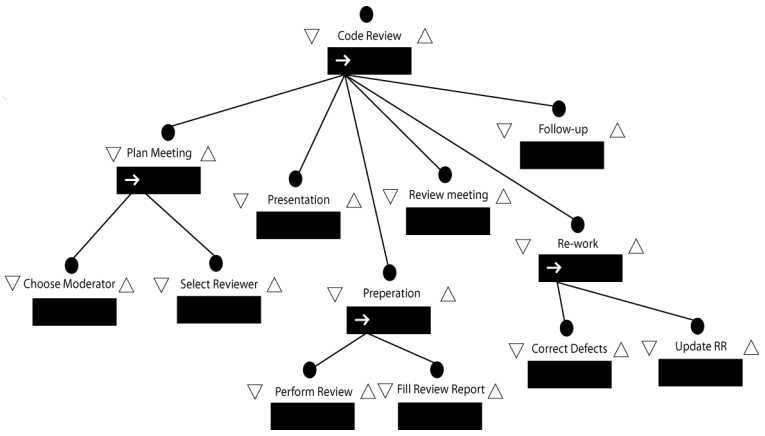

Figure 7.6: Process Module definition using Little-JIL

Developing a process architecture bottom-up requires a rationale to guide how process modules are combined to form process components e.g., a rationale may be derived from high level process goals or development needs e.g. comply with CMMI-Dev goals. With the purpose of detailing how an architectural process component is implemented using Little-JIL, an example scenario of organizational practices is described next.

Organizational scenario

One of the recognized approaches to delivering quality software is to perform code reviews. They allow finding defects near the point of injection, making their removal cheaper. Usually, from a code review, a report is expected with a list of defects identified. Also, according to good practices of managing document configurations, storing the document in a version control system is expected to assure traceability on document changes. Additionally, for monitoring and supporting process improvement measurements related to code reviews are expected.

Figure 7.7 depicts two process components built from process modules present in the Module Library of Figure 7.4. The process component are a example of an initial architecture that resulted from interpreting the organizational scenario described. The process is then iterative as new components are added to the architecture as new development practices and requirements are considered. Detailing the process of eliciting an architecture is not the subject of this research work, we are focusing rather in the narrow issue of supporting the transition from architectural to implementation phases.

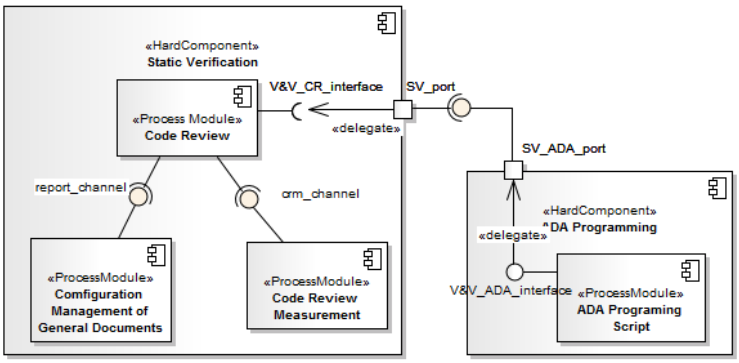

Figure 7.7: Process Component Diagram

7.3.2 Refinement of a process architecture

The architecture will be refined reusing process modules present in the Module Library of Figure 7.4. The *Code Review* process module will be combined to a *Configuration Management of General Documents* process module. Code Review process module (see Figure 7.6) has a step named *Fill Review Report* that delivers a report that is expected to be subject to configuration management practices. Based on this requirement an interface between *Code Review* and *Configuration Management of General Documents* process modules is defined by declaring the `report_channel` (see Figure 7.7). The respective process modules implementation are depicted in Figure 7.6 and Figure 7.8 respectively.

Figure 7.9 depicts, partially, both *Code Review* and *Configuration Management of General Documents* process trees. The focus is now on the implementation of the

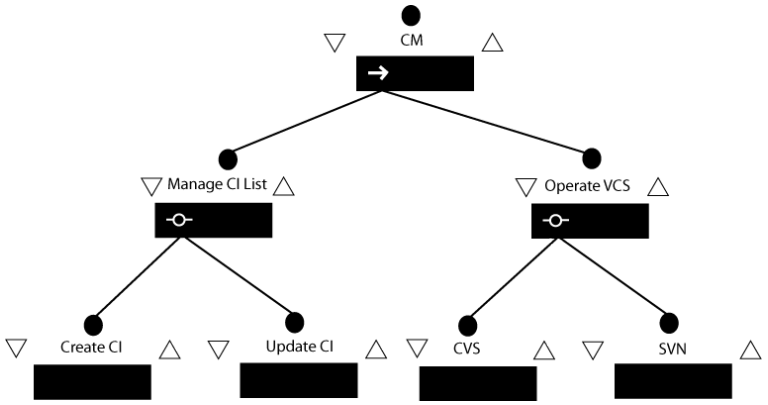

Figure 7.8: Configuration Management for General Documents module defined using Little-JIL

interface `report_channel` applying the transitions rules described in the previous section:

- Applying rule 1: communicating steps of both *Code Review* (see Figure 7.7) and *Configuration Management of General Documents* (see Figure 7.8) process modules are identified. The function requested to the *Code Review* process module is one of configuration management for the *review report* that is an output of the *Fill Review Report* leaf step. The function of configuration management is to be provided by the *CM* root step.

- Applying rule 2: a reaction arc is declared connecting both process trees. The reaction is declared in the parent step *(Preparation)* of the *Fill Review Report* step and connects to the *CM root* step of *Configuration Management of General Documents* process module. The reaction arc is signaled by the lightning badge in the *Preparation* step.

- Applying rule 3: a message (`report_msg: StepfinishedEvent`) is declared on the arc of *Fill Review Report* step. When the step has finished a message is triggered. In the reaction arc `report_msg` is declared to identify that the *Preparation* step reacts when `report_msg` is received in the *Preparation* step.

- Applying rule 4: a channel is declared in the *Preparation* step to allow communication of parameters at lower hierarchical steps. A `report_channel` is declared in both arcs between *Preparation* step to *CM* and *Fill Review Report* steps, to communicate the report done document. *Fill Review Report* step writes the parameter into the channel to be read by *CM* step. This assures that report is passed between process steps. A parameter binding is also declared to state that

report done document should be document parameter input to *CM* step. This assures changes are not required below the root step and the module is reused as is below the root step.

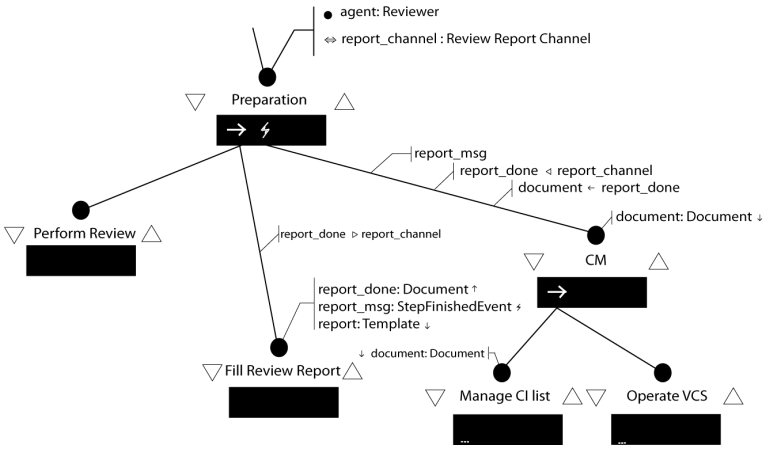

Figure 7.9: Interface report_channel for Static Verification HardComponent implemented using Little-JIL

7.4 Analysis and discussion of results

A first goal was set to develop an architecture in order to have a birds eye view of the OSSP. Although previous research has demonstrated how to develop such an architecture, the approach presented in this research work provides a mechanism to have a direct map of architectural process entities to low level process specifications. At an architectural level interfacing becomes a first class modelling entity. The approach presented allows a systematic translation of interfaces present at the architectural level to low level process definitions making easier process engineer task of process modelling.

Relative to the orchestration concern, the use of Little-JIL allowed coordination information to become explicit not only within process module definitions but also across modules when combining process modules. Coordination information improves the understandability of process definitions both for process performs and process engineers in the role of processes owners. It is now explicit the information related to how and when processes are communication with each and when.

The applicability of the design approach at CRITICAL in creating Module Library revealed that defining process modules is easier by developing first an architectural

blueprint of performed practices in the organization. The development of an architecture should be iterative allowing to identify in each iteration which process modules should be present in a Module Library.

A sub-set of existing process definitions were described as process modules that are reused in defining new and updated process life-cycles. This design approach can be applied in modelling complex process systems, not only for software as Little-JIL is a general purpose modelling language and the set of transition rules remains applicable.

The results are limited in the extent that only a situational example was given and it is not representative of all type of interactions in a system of processes. The approach for design a process architecture and associated process description is highly dependent of a software tool that can support the exercise of design and implementation of processes.This is because Little-JIL is mainly a visual description language and the software is not yet mature to support a large scale use. Also the perception of improved level of information was not validated empirically.

7.5 Conclusion

Adopting multiple best practices models into one single environment results in a high number of quality requirements and goals to be satisfied. This often leads to an increase in the number of activities and processes to be performed in the organization, leading to large and complex systems of processes that become hard to analyse and manage effectively. In this research work a process design approach is proposed and a example is given on how to develop a process architecture to better manage the increasing size of a system of processes. Additionally, a description is given on how an architecture can be refined to a process implementation using Little-JIL process programming language. An example how to apply the approach is given based on a software development scenario derived from process definitions from a Portuguese software house.

A process architecture aims to provide a perspective on how a system of processes is structured, where the predominant view is one of functional dependencies between process components. Process components provide the necessary abstraction of low-level process details and allow focusing on the dependencies among process components. The abstraction layer resulting from a process architecture improves communication among process stakeholders and process owners. Additionally, it provides a mean of manifestation of early design decisions.

A process architecture built upon process modules allows having a systemic view of the development process. It allows process owners or designers to focus on situational development needs and assure that links to other process modules are not

broken by omission or by neglecting information that is being communicated between modules. At first hand, having a process architecture based on process modules will facilitate definition and maintenance of project specific development processes. A possible use case is the one of a project manager, that can define project specific processes by adding, dropping or replacing process modules that better adjust to project quality needs. Attitudinally is is also a form to communicate the development process to the project team.

		Little-JIL Process Constructs													
	Language Constructs	**Steps**	**Modules**	**Sequencing**	**Step Interface**	**Resources**	**Parameters**	**Channels**	**Cardinality**	**Requisites**	**Exceptions**	**Handlers**	**Deadlines**	**Messages**	**Reactions**
SPEM - Method Content Package	Default Responsibility Assignment	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	Default Task Definition Parameter	–	–	–	–	–	X	–	–	–	–	–	–	–	–
	Default Task Definition Performer	–	–	–	–	X	–	–	–	–	–	–	–	–	–
	Method Content Element	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	Optionality Kind	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	Qualifications	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	Role definition	–	–	–	–	X	–	–	–	–	–	–	–	–	–
	Step	X	–	X	–	–	–	–	–	–	–	–	–	–	–
	Task Definition	X	–	–	–	–	–	–	–	–	–	–	–	–	–
	Tool Definition	–	–	–	–	X	–	–	–	–	–	–	–	–	–
	Work Product Definition	–	–	–	–	X	–	–	–	–	–	–	–	–	–
	Work Product Definition Relationship	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Table 7.1: SPEM MCP and Little-JIL semantic constructs comparison

Chapter 8

Conclusions

Contents

This chapter provides a summary of the research efforts, contribution and limitations of this dissertation. In retrospective we also discuss possible research topics and the applicability of our proposals.

8.1 Synthesis of research efforts

This dissertation is framed by an organisational context characterized by the adoption of multiple improvement models as a strategy to improve software processes. Research on software process improvement in multi-model environment is a fairly recent topic and research contributions address challenges related to selection and integration of multiple models into a single organisational environment.

Additionally, given the nature of this dissertation, that involved a close collaboration with industry, an experimental case study was conducted that relates to the enactment of a software process improvement process by documenting an iteration of process improvement in line with the highest standards of maturity in the industry.

As described in Chapter 2 there are two paradigms for software process improvement, the benchmark or best practices based approach and the analytical of principle based approach. The benchmarked based approach as gained recognition from the industry and improvement models have been adopted with considerable success, although issues exist in this approach, namely, the suitability of the improvement models for small organisations and validity of claims of actual benefits resulting from models adoption. Multi-model environments are created when multiple improvement models are adopted into a single organisational environment and the most frequent combinations result from combining benchmarked based models, e.g., CMMI-Dev and ISO standards. Combinations of benchmarked based approaches and principle based are also present e.g., CMMI-Dev and Six Sigma. Research challenges in multi-model environments are summarised in the context of an harmonisation framework defined by Siviy [SKMM08c]. The fact that a great number of improvement models models exist across several disciplines makes selection of improvement models a challenge to organizations adopting improvement models. Also, when several improvement models are selected their integration for a joint implementation becomes a challenge.

In Chapter 3 we addressed the issue of improvement models selection by extending a taxonomy to characterise improvement models. One of the approaches to better understand improvement models to inform adoption decisions is to compare their content. Several methods exist to compare improvement models. Our contribution evolve one of this methods by proposing size and complexity metrics to analyse architectural properties of improvement models and derive a method for their computation. We applied the method to derive a quantitative understanding of ISO standards, namely, ISO12207, ISO9001 and ISO15528 metrics for size and complexity when compared to CMMI-Dev as a base reference model.

Research Contribution 1: A proposal for computing size and complexity metrics in the scope of a taxonomy for improvement frameworks and a method to derive their

computations. The contribution addresses the research goal 1 defined in Chapter 1.

In Chapter 4 we addressed the challenge of compliance in multi-model environments. When multiple improvement models are adopted, specifically benchmarked based models, ensuring compliance of organizational practices with prescribed practices or goals from adopted models is often a necessity. Whether the requirement is set by normative bodies (external compliance) or by the model itself (internal compliance) organizations in multi-model environments face the challenge of making sure that all adopted practices or goals are met. To our knowledge there is no relevant work in this topic of ensuring compliance to multi-model environments.

Our proposal is based on a set of constructs and a model that organizes information to manage compliance in multi-model environments. The approach consists of a set of abstractions that aim to isolate existing structural and terminology differences of improvement models. When the model is instantiated, the resulting information system takes advantage of similarities and differences of adopted models. By making explicit these differences and similarities the resulting information system can be used to optimize the effort required to ensure compliance in multiple improvement models.

A scenario of a multi-model environment was used to demonstrate the applicability of the model. The scenario was defined by considering publicly available mappings that identify similarities of ISO standards with CMMI-Dev. The scenario allowed to demonstrate how information can be managed when improvement models share a portion of their content, meaning they prescribe similar goals or practices. An example of a multi-model process assessment was described to demonstrate how in a single exercise, one can assess compliance with multiple improvement models.

Research Contribution 2: A set of constructs and a model for managing compliance in multi-model environments. The contribution addresses research goal 2 defined in Chapter 1.

In Chapter 5 and Chapter 6 we address the challenge of continuously improving the software process. Organizational strategies for software process improvement can consider both benchmarked and principle based approaches. Benchmarked based approaches provide a reference base of good practices and analytical approaches typically provide the work flow for continuously improving the software process. The focus is often improving the set of organizational practices that are adopted based on benchmarked based models.

The goal of operating at CMMI-Dev maturity level 5 is that organizations *'..continually improves its processes based on a quantitative understanding of its business objectives and performance needs'*. This goal is shared by most analytical based ap-

proaches for software process improvement. Operating at level 5, meaning a continuously optimizing organization, is claimed to be the stage when the benefits of software process improvement are most significant. The fact is that the details of what involves operating a maturity level 5 is yet a practice that few organizations are capable of.

Our motivation was to shed a light on what involves meeting requirements of an optimizing organizations in line with CMMI-Dev level 5 view of an optimizing organization. With that goal in mind we provide a case study of an improvement initiative in line with requirements set by CMMI-Dev Maturity Level 5 Optimizing staged representation. We detail how an improvement process was instantiated with the end goal of improving an inspection process.

Research Contribution 3: We describe a process improvement approach to guide organizations in operating at optimizing level in line with requirements set by CMMI-Dev Optimizing staged representation. The contribution addresses the research goal 3 defined in Chapter 1.

In Chapter 7 we address the challenge that organizations face when their system of processes becomes too complex with a considerable number of activities and interdependencies between activities. Benchmarked based models tend to be extensive in the number of practices prescribed. It is fairly easy for organizations to end up with considerable number of activities on their organizational set of standard processes. In the context of adopting multiple improvement models this can be aggravated. As the system of processes increases in size it becomes increasingly difficult to design and manage the overall process, even if detailed descriptions of processes are maintained. Planning at higher levels of abstraction becomes a necessity to manage complexity. There are few approaches for designing process architectures and the traceability to low level process descriptions is not explicitly addressed by existing approaches.

In line with this challenge, we propose a meta-model to design software process architectures that, through a set of transitions rules, allows to refine high level process constructs into low level process constructs. We demonstrate it's applicability by defining an example of a process architecture specification and using the transition rules to refine the architecture specification into low level process descriptions using Little-JIL as the low level process modelling language.

Research Contribution 4: A meta-model for defining process architectures and a set of transition rules to support the refinement of high level process specifications into low level process specifications, assuring traceability between high level and low level specifications. The contribution addresses the research goal 4 defined in Chapter 1.

8.2 Reliability and validity

In line with design research methodology described in Chapter 1 we fulfilled to different extents the sequencing of expected activities. The activities directly related to reliability and validation of the research work are *Activity 4. Demonstration*, *Activity 5. Evaluation* and *Activity 6. Communication*.

Activity 4. Demonstration

Our proposed solutions were applied in solving an instance of the problem to the possible extension of the problem. Our goal was to ensure that through logical reasoning and by using methods and models proposed, we were successful in solving the problems addressed. These demonstrations formed the core of the content submitted for peer reviews that resulted in successful publications in international conferences.

Activity 5. Evaluation

We are aware that we are short in evaluation activities that are desirable to supplement the demonstration of proposed solutions. Additional validation efforts, e.g., gathering empirical evidence through rigorous research methods is desirable for a sound contribution to the software engineering body of knowledge. Unfortunately, it was not possible develop such validation efforts due to the magnitude of time and resources required.

Activity 6. Communication

As result of our research efforts a total of 6 research papers were successfully accepted in international peer reviewed conferences.

1. André L. Ferreira, Ricardo J. Machado, Mark C. Paulk. **An Approach to Software Process Design and Implementation Using Transition Rules**. Stefan Biffl, Mika Koivuluoma, Pekka Abrahamsson, Markku Oivo (Eds.), Proceedings of the 37th EUROMICRO Conference on Software Engineering and Advanced Applications - SEAA'2011, Track on Software Process and Product Improvement (SPPI), Session on Process Design and Implementation, Oulu, Finland, September, 2011, pp. 330-333, IEEE Computer Society Press, Los Alamitos, California, U.S.A., [ISBN: 978-0-7695-4488-5].

2. André L. Ferreira, Ricardo J. Machado, Mark C. Paulk. **Supporting Audits and Assessments in Multi-Model Environments**. Danilo Caivano, Markku Oivo, Maria Teresa

Baldassarre, Giuseppe Visaggio (Eds.), Product-Focused Software Process, pp. 73-87, LNCS Series vol. 6759, Springer-Verlag, Berlin Heidelberg, Germany, June, 2011, [ISBN: 978-3-642-21843-9]. (Proceedings of the 12th International Conference on Product Focused Software Development and Process Improvement - PROFES'2011, Torre Canne, Italy, June, 2011).

3. André L. Ferreira, Ricardo J. Machado, Mark C. Paulk. **Quantitative Analysis of Best Practices Models in the Software Domain**. Jun Han, Tran Dan Thu (Eds.), Proceedings of the 17th Asia-Pacific Software Engineering Conference - APSEC'2010, Session on Software Process, Sydney, Australia, December, 2010, pp. 433-442, IEEE Computer Society Press, Los Alamitos, California, U.S.A., [ISBN: 978-0-7695-4266-9].

4. André L. Ferreira, Ricardo J. Machado, Lino Costa, José G. Silva, Rui F. Batista, Mark C. Paulk. **An Approach to Improving Software Inspections Performance**. Proceedings of the 26th IEEE International Conference on Software Maintenance - ICSM'2010, Session on Industry: Software Process, Timisoara, Romania, September, 2010, IEEE Press, Piscataway, New Jersey, U.S.A., [ISBN: 978-1-4244-8628-1].

5. André L. Ferreira, Ricardo J. Machado, Mark C. Paulk. **Size and Complexity Attributes for Multi-model Software Process Improvement Framework Taxonomy**. M.R.V. Chaudron (Ed.), Proceedings of the 36th EUROMICRO Conference on Software Engineering and Advanced Applications - SEAA'2010, Track on Software Process and Product Improvement (SPPI), Lille, France, September, 2010, pp. 306-309, IEEE Computer Society Press, Los Alamitos, California, U.S.A., [ISBN: 978-0-7695-4170-9].

6. André Ferreira, Ricardo J. Machado. **Software Process Improvement in Multimodel Environments**. Proceedings of the 4th International Conference on Software Engineering Advances - ICSEA'2009, Session on SEDES'2009 Workshop, September, 2009, pp. 512-517, IEEE Computer Society Press, Los Alamitos, California, U.S.A., [ISBN: 978-0-7695-3777-15].

8.3 Recommendations for further research

This research work proposed several solutions to deal with specific challenges of software process improvement in multi-model environments. As stated previously proposed solutions require further validation and in line with this shortcoming a tool to support the implementation of the solutions presented in Chapter 3 and Chapter 4 is being developed.

A proof of concept tool - called **MOPP** - to compute automatically the metrics of size and complexity described in Chapter 3 when a map between two models is provided is being developed. The goal is to automate the calculation of the metrics

when comparisons of models are made available and thus collect further information to validate the usefulness of the metrics.

In what concerns Chapter 5 and Chapter 6 contributions, we argue that integration of benchmark based approaches and analytical approaches for software process improvement requires that industry is receptive to perform such empirical studies and that academia alone cannot deliver significant insights to this topic alone. Analytical approaches are closely related to requirements set by the Optimizing CMMI-Dev Optimizing maturity level. Operating at optimizing maturity is a challenge to organizations as it stresses the ability to find the right balance between investment in measurement and experimentation to improve the software process consistently. Thus, methods on how to drive an optimizing organization require further research.

In the scope of Chapter 7 research work, a proof of concept was described of how to link high level order process specifications with low level process descriptions. The proposal developed used Little-JIL process modelling language as it provides behavioural constructs to model low level process description. However, Little-JIL is far from being a widely adopted modelling language and thus tool support to develop Little-JIL process specification are limited and Little-JIL depends heavily on visual representation to be effective.

For effective process modelling, complexity can only be tackled by linking two abstraction layers. A low level layer, composed of detailed process description with built-in orchestration constructs and a high level abstraction layer that focus on functional and non-functional process specifications. One possibility to explore the use to Business Process Modelling Notation (BPMN) in place of Little-JIL. The reason is that BPMN has wider user base and the set of tools supporting the notation is significant.

Bibliography

[20005] *Introduction to ITIL.* Best practice. Stationery Office, 2005. **Cited** on page 43.

[707] ISO/IEC JTC 1/SC 7. ISO/IEC systems and software engineering - measurement process. 2007. **Cited** on page 53.

[Bas85] Victor R. Basili. Quantitative evaluation of software engineering methodology. In *Proceedings of the First Pan Pacific Computer Conference*, Melbourne, Australia, September 1985. **Cited** on pages 16, 26, 27, 28 and 102.

[Bas89] Victor R. Basili. Software development: a paradigm for the future. In *Computer Software and Applications Conference, 1989. COMPSAC 89., Proceedings of the 13th Annual International*, pages 471–485, 1989. **Cited** on pages 4 and 18.

[Bas93] Victor R. Basili. The experience factory and its relationship to other improvement paradigms. In Ian Sommerville and Manfred Paul, editors, *Software Engineering — ESEC '93*, volume 717 of *Lecture Notes in Computer Science*, pages 68–83. Springer Berlin Heidelberg, 1993. **Cited** on page 26.

[BCP+10] Maria T. Baldassarre, D. Caivano, Francisco J. Pino, M. Piattini, and G. Visaggio. A strategy for painless harmonization of quality standards: A real case. In M. Ali Babar, Matias Vierimaa, and Markku Oivo, editors, *Product-Focused Software Process Improvement*, volume 6156 of *Lecture Notes in Computer Science*, pages 395–408. Springer Berlin Heidelberg, 2010. **Cited** on page 77.

[BCPP12] Maria T. Baldassarre, D Caivano, Francisco J. Pino, and M Piattini. Harmonization of ISO/IEC 9001:2000 and CMMI-DEV: from a theoretical comparison to a real case application - springer. *Software Quality \ldots*, 2012. **Cited** on page 74.

[BCR02a] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. *Experience Factory*. John Wiley Sons, Inc., 2002. **Cited** on pages xvii, 28, 29 and 102.

[BCR02b] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. *Experience Factory*. John Wiley and Sons, Inc., 2002. **Cited** on page 28.

[BDRM10] A. Barreto, E. Duarte, A. Rocha, and L. Murta. Supporting the definition of software processes at consulting organizations via software process lines. page 15–24, 2010. **Cited** on pages 170, 173 and 175.

[BHL+07] V. Basili, J. Heidrich, M. Lindvall, J. Munch, M. Regardie, and A. Trendowicz. GQM+ strategies – aligning business strategies with software measurement. *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on*, page 488–490, August 2007. **Cited** on page 121.

[Bie04] R. Biehl. Six sigma for software. *Software, IEEE*, 21(2):68–70, 2004. **Cited** on page 31.

[BJGB10] R. Bendraou, J. Jézéquel, M. Gervais, and X. Blanc. A comparison of six UML-Based languages for software process modeling. *Software Engineering, IEEE Transactions on*, 36(5):662–675, 2010. **Cited** on page 171.

[BKKW07] Tobias Bucher, Mario Klesse, Stephan Kurpjuweit, and Robert Winter. Situational method engineering. In Jolita Ralyté, Sjaak Brinkkemper, and Brian Henderson-Sellers, editors, *Situational Method Engineering: Fundamentals and Experiences*, volume 244 of *IFIP — The International Federation for Information Processing*, pages 33–48. Springer US, 2007. **Cited** on page 183.

[BLR+10] V. Basili, M. Lindvall, M. Regardie, C. Seaman, J. Heidrich, J. Munch, D. Rombach, and A. Trendowicz. Linking software development and business strategy through measurement. *Computer*, 43(4):57–65, 2010. **Cited** on page 121.

[BPM09] Business Process Modeling Notation (BPMN) Version 1.2. Technical report, January 2009. **Cited** on page 171.

[BPPV09a] M. Baldassarre, M. Piattini, F. Pino, and G. Visaggio. Comparing ISO/IEC 12207 and CMMI-DEV: towards a mapping of ISO/IEC

15504-7. In *Software Quality, 2009. WOSQ '09. ICSE Workshop on*, page 59–64, 2009. **Cited** on pages 47 and 74.

[BPPV09b] Maria T. Baldassarre, M. Piattini, Francisco .J. Pino, and G. Visaggio. Comparing iso/iec 12207 and cmmi-dev: Towards a mapping of iso/iec 15504-7. In *Software Quality, 2009. WOSQ '09. ICSE Workshop on*, pages 59–64, May 2009. **Cited** on page 77.

[BR88] V. Basili and H. Rombach. The TAME project: towards improvement-oriented software environments. *Software Engineering, IEEE Transactions on*, 14(6):758–773, 1988. **Cited** on pages 28 and 102.

[Cam12] Michael Campo. Why cmmi maturity level 5? *CrossTalk - The Journal of Defense Software Engineering*, 2012. **Cited** on page 95.

[Car91] David Card. Understanding process improvement. *Ieee Software*, 8(4):102–103, 1991. **Cited** on pages 3 and 18.

[Car04] David Card. Research directions in software process improvement. *Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International*, page 238 vol.1, August 2004. **Cited** on pages 4 and 19.

[CF02] H. Conradi and A. Fuggetta. Improving software process improvement. *Ieee Software*, 19(4):92–99, July 2002. **Cited** on pages 2, 4 and 19.

[Cha05] R. N. Charette. Why software fails [software failure]. *IEEE Spectr.*, 42(9):42–49, September 2005. **Cited** on page 2.

[CJ99] Reidar Conradi and M.Letizia Jaccheri. Process modelling languages. In Jean-Claude Derniame, BadaraAli Kaba, and David Wastell, editors, *Software Process: Principles, Methodology, and Technology*, volume 1500 of *Lecture Notes in Computer Science*, pages 27–52. Springer Berlin Heidelberg, 1999. **Cited** on page 170.

[CK06] M. Chissis and M. Konrad. CMMI for development, version 1.2. *Addison-Wesley*, 2006. **Cited** on pages 18 and 170.

[CKS11] M. Chrissis, M. Konrad, and S. Shrum. *CMMI for Development: Guidelines for Process Integration and Product Improvement ; [CMMI-DEV, Version 1.3]*. SEI Series in Software Engineering. Addison Wesley Professional, 2011. **Cited** on pages xvii, xviii, 3, 22, 24 and 98.

[Cro79] P. Crosby. Quality is free: the art of making quality certain. *McGraw-Hill*, 1979. **Cited** on pages 16 and 32.

[dACBP03] J. de Almeida, J. Camargo, B. Basseto, and S. Paz. Best practices in code inspection for safety-critical software. *Software, IEEE*, 20(3):56–63, 2003. **Cited** on page 138.

[DD98] Onur Demirors and Elif Demirors. Software process improvement in a small organization: Difficulties and suggestions. In Volker Gruhn, editor, *Software Process Technology*, page 1–12. Springer Berlin / Heidelberg, 1998. **Cited** on page 20.

[Dem00a] W. Deming. *The New Economics: For Industry, Government, Education*. Massachusetts Institute of Technology, Center for advanced engineering study, 2000. **Cited** on pages xvii, 27 and 99.

[Dem00b] W. Deming. Out of the crisis: quality productivity and competitive position. *Cambridge University Press*, 2000. **Cited** on pages 16, 27 and 32.

[DS97] M Diaz and J Sligo. How software process improvement helped motorola. *Software, IEEE*, 14(5):75–81, September 1997. **Cited** on pages 19 and 20.

[DS07] C. Denger and F. Shull. A practical approach for quality-driven inspections. *Software, IEEE*, 24(2):79–86, 2007. **Cited** on page 139.

[Dyb05] T Dyba. An empirical investigation of the key factors for success in software process improvement. *Software Engineering, IEEE Transactions on*, 31(5):410–424, 2005. **Cited** on page 21.

[ED07] Christof Ebert and Reiner Dumke. *Software Measurement: Establish — Extract — Evaluate — Execute*, chapter Introducing a Measurement Program, pages 109–164. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. **Cited** on page 121.

[Fag76] M. Fagan. Design and code inspections to reduce errors in program development. *IBM Systems Journal 15*, page 182–211, 1976. **Cited** on pages 134, 135 and 136.

[Fag86] M. Fagan. Advances in software inspections. *IEEE Trans. Softw. Eng.*, 12(7):744–751, 1986. **Cited** on pages 135 and 138.

[FCB00] W. Florac, A. Carleton, and J. Barnard. Statistical process control: analyzing space shuttle onboard software process. *Software, IEEE*, 17(4):97–106, 2000. **Cited** on page 2.

[Fei83] A. Feigenbaum. Total quality control. *McGraw-Hill*, 1983. **Cited** on page 32.

[Fie09] Andy Field. *Discovering statistics using SPSS*. 2009. **Cited** on page 145.

[Flo93] R.L. Flood. *Beyond TQM*. John Wiley & Sons, 1993. **Cited** on page 31.

[FO99] B. Fitzgerald and T. O'Kane. A longitudinal study of software process improvement. *Software, IEEE*, 16(3):37–45, May 1999. **Cited** on pages 19, 20 and 135.

[FS98] J. Ferguson and S. Sheard. Leveraging your CMM efforts for IEEE/EIA 12207. *Software, IEEE*, 15(5):23–28, 1998. **Cited** on pages 4 and 46.

[FSM14] D. Falessi, M. Shaw, and K. Mullen. Achieving and maintaining cmmi maturity level 5 in a small organization. *Software, IEEE*, 31(5):80–86, Sept 2014. **Cited** on page 95.

[GBEA07] M. Goulao and F. Brito E Abreu. Modeling the experimental software engineering process. In *Quality of Information and Communications Technology, 2007. QUATIC 2007. 6th International Conference on the*, pages 77–90, Sept 2007. **Cited** on pages 123, 124 and 147.

[GW00] Volker Gruhn and Ursula Wellen. Structuring complex software processes by ´process landscaping´. In Reidar Conradi, editor, *Software Process Technology*, volume 1780 of *Lecture Notes in Computer Science*, pages 138–149. Springer Berlin Heidelberg, 2000. **Cited** on page 172.

[GW01] Volker Gruhn and Ursula Wellen. *Unifying Petri Nets: Advances in Petri Nets*, chapter Process Landscaping: Modelling Distributed Processes and Proving Properties of Distributed Process Models, pages 103–125. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001. **Cited** on pages 170 and 171.

[Hal96] Thomas J. Haley. Software process improvement at raytheon. *IEEE Software*, 13(6):33–41, 1996. **Cited** on pages 2 and 19.

[Hat08] L Hatton. Testing the value of checklists in code inspections. *Software, IEEE*, 25(4):82–88, 2008. **Cited** on pages 134 and 138.

[HC01] Christian Halvorsen and Reidar Conradi. *A Taxonomy to Compare SPI Frameworks*. 2001. **Cited** on pages 36, 37, 38, 46 and 47.

[HC10] A. Hevner and S. Chatterjee. *Design Research in Information Systems: Theory and Practice*. Integrated Series in Information Systems. Springer, 2010. **Cited** on pages 9 and 11.

[Hit03] Derek Hitchins. *Advanced Systems: Thinking, Engineering, and Management*. 2003. **Cited** on page 51.

[HK89] Watts Humphrey and Marc Kellner. Software process modeling: principles of entity process models. *Proceedings of the 11th international conference on Software engineering*, 1989. **Cited** on pages 8 and 171.

[HKS12] D.E. Harter, C.F. Kemerer, and S.A. Slaughter. Does software process improvement reduce the severity of defects? a longitudinal field study. *Software Engineering, IEEE Transactions on*, 38(4):810–827, 2012. **Cited** on page 19.

[HMPR04] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS Q.*, 28(1):75–105, 2004. **Cited** on page 10.

[Hol95] Robert Holeman. The software process improvement game. In Rosalind Ibrahim, editor, *Software Engineering Education*, page 259–261. Springer Berlin / Heidelberg, 1995. **Cited** on page 19.

[HP09] Keith Heston and William Phifer. The multiple quality models paradox: how much best practice is just enough? *Software Process: Improvement and Practice*, 2009. **Cited** on pages 38, 46 and 47.

[HS06] B. Henderson-Sellers. SPI - a role for method engineering. In *Software Engineering and Advanced Applications, 2006. SEAA '06. 32nd EUROMICRO Conference on*, page 4–5, 2006. **Cited** on page 183.

[Hum88] W. Humphrey. Characterizing the software process: a maturity framework. *Software, IEEE*, 5(2):73–79, March 1988. **Cited** on page 170.

[Hum89] W. Humphrey. Managing the software process. *Addison-Wesley Professional*, 1989. **Cited** on pages 3 and 16.

[Hum93] W. Humphrey. Introduction to software process improvement. 1993. **Cited** on pages 2 and 3.

[IEC00] ISO IEC. 9001 quality management systems - requirements. 2000. **Cited** on page 22.

[IEE04] IEEE. Ieee 1028 - standard for software reviews and audits. Technical report, 2004. **Cited** on pages 81 and 82.

[IEE12] IEEE. Ieee standard for configuration management in systems and software engineering. *IEEE Std 828-2012 (Revision of IEEE Std 828-2005)*, pages 1–71, 2012. **Cited** on page 33.

[II08] ISO/IEC and IEEE. Iso/iec 12207 - systems and software engineering - software life cycle processes. 2008. **Cited** on pages 3, 18, 23 and 170.

[Ins15] CMMI Institute. Cmmi institute published appraisal results. https://sas.cmmiinstitute.com/pars/pars.aspx, March 2015. **Cited** on page 95.

[ISO04a] ISO/IEC. Iso/iec 15504 software process improvement and capability determination model (spice). 2004. **Cited** on pages xvii, 18, 22, 23, 82 and 180.

[ISO04b] ISO/IEC. Software engineering — process assessment — part 2: Performing an assessment. 2004. **Cited** on page 82.

[ISO06] ISO/IEC. 15504-5 information technology — process assessment — part 5: An exemplar process assessment model. 2006. **Cited** on pages 83 and 87.

[JG99] J. Juran and A. Godfrey. Juran's quality control HandBook. *McGraw-Hill*, 1999. **Cited** on page 32.

[Jon96] C. Jones. Our worst current development practices. *Software, IEEE*, 13(2):102–104, 1996. **Cited** on page 2.

[JP05] A. Jedlitschka and D. Pfahl. Reporting guidelines for controlled experiments in software engineering. In *Empirical Software Engineering, 2005. 2005 International Symposium on*, pages 10 pp.–, Nov 2005. **Cited** on page 123.

[JT98] Philip Johnson and Danu Tjahjono. Does every inspection really need a meeting? *Empirical Software Engineering*, 3(1):9–35, 1998. **Cited** on pages 136, 149 and 151.

[KKC00] Kazman, Klein, and Clements. ATAM:Method for architecture evaluation. *CMU/SEI-2000-TR-004*, 2000. **Cited** on page 43.

[KKR07] E Kantorowitz, T Kuflik, and A Raginsky. Estimating the required code inspection team size. In *Software-Science, Technology & Engineering, 2007. SwSTE 2007. IEEE International Conference on*, page 104–115, 2007. **Cited** on page 137.

[KM93] J. Knight and E. Myers. An improved inspection technique. *Commun. ACM*, 36(11):51–61, 1993. **Cited** on page 136.

[Kot96] J.P. Kotter. *Leading Change*. Harvard Business School Press, 1996. **Cited** on page 126.

[KP09] C. Kemerer and M. Paulk. The impact of design and code reviews on software quality: An empirical study based on PSP data. *Software Engineering, IEEE Transactions on*, 35(4):534–550, 2009. **Cited** on pages 135 and 138.

[KPB99] P. Kuvaja, J. Palo, and A. Bicego. TAPISTRY—A software process improvement approach tailored for small enterprises. *Software Qual J*, 1999. **Cited** on pages xvii, 25 and 26.

[Lai98] O. Laitenberger. Studying the effects of code inspection and structural testing on software quality. *Software Reliability Engineering, 1998. Proceedings. The Ninth International Symposium on*, page 237–246, 1998. **Cited** on page 134.

[McF96] B. McFeeley. Ideal: User's guide for software process improvement. *CMU/SEI-96-HB-001*, 1996. **Cited** on pages xvii, 28, 30 and 107.

[Mil07] P. Miller. An SEI process improvement path to software quality. *Quality of Information and Communications Technology, 2007. QUATIC 2007. 6th International Conference on the*, page 12–20, August 2007. **Cited** on page 2.

[ML09] M. Mantyla and C. Lassenius. What types of defects are really discovered in code reviews? *Software Engineering, IEEE Transactions on*, 35(3):430–448, 2009. **Cited** on page 134.

[Moo99] J.W. Moore. An integrated collection of software engineering standards. *Software, IEEE*, 16(6):51–57, 1999. **Cited** on pages xvii, 33, 38 and 39.

[MS03] B. Mutafelija and H. Stronberg. Systematic process improvement using ISO 9001:2000 and CMMI. *Artech House*, 2003. **Cited** on pages 37, 74 and 77.

[MS09] Boris Mutafelija and Harvey Stromberg. ISO 9001:2000 to CMMI v1.2 map. *http://www.sei.cmu.edu/cmmi/casestudies/mappings/cmmi12-iso.cfm (Accessed March 2010)*, 2009. **Cited** on pages xvii, 47, 60, 61, 62, 88 and 89.

[MSS09a] Boris Mutafelija, Harvey Stromberg, and SEI. ISO 12207:2008 to CMMI v1.2 map. Technical report, 2009. **Cited** on pages 47, 60, 61, 88 and 89.

[MSS09b] Boris Mutafelija, Harvey Stromberg, and SEI. ISO 15288:2008 to CMMI-DEV v1.2 map. Technical report, 2009. **Cited** on pages 47, 60 and 61.

[Mun95] John C. Munson. Software measurement: Problems and practice. *Annals of Software Engineering*, 1(1):255–285, 1995. **Cited** on page 4.

[Mut06] B. Mutafelija. Architecting standard processes with SWEBOK and CMMI. *SEPG Conference*, 2006. **Cited** on page 41.

[MVW06] Jürgen Münch, Matias Vierimaa, and Hironori Washizaki. *Building Software Process Line Architectures from Bottom Up*, volume 4034. 2006. **Cited** on page 173.

[NV10] Gopalakrishman Nair and Suma V. Impact analysis of the inspection process for effective defect management in software development. *Software Quality Professional*, 12(2), 2010. **Cited** on page 139.

[Ost87] Leon Osterweil. Software processes are software too. In *Proceedings*, page 2–13, 1987. **Cited** on pages 170 and 172.

[Pau98] Mark Paulk. Using the software CMM in small organizations. *Software Engineering Institute*, page 13, 1998. **Cited** on pages 20 and 22.

[Pau08] Mark Paulk. A taxonomy for improvement frameworks. *Fourth World Congress for Software Quality*, 2008. **Cited** on pages 4, 7, 33, 38, 46, 47, 48 and 71.

[PBP+10] Francisco J. Pino, Maria T. Baldassarre, Mario Piattini, Giuseppe Visaggio, and Danilo Caivano. Mapping software acquisition practices from iso 12207 and cmmi. In LeszekA. Maciaszek, César González-Pérez, and Stefan Jablonski, editors, *Evaluation of Novel Approaches to Software Engineering*, volume 69 of *Communications in Computer and Information Science*, pages 234–247. Springer Berlin Heidelberg, 2010. **Cited** on pages 77 and 79.

[PBPV09] Francisco Pino, Maria Baldassarre, Mario Piattini, and Giuseppe Visaggio. Harmonizing maturity levels from CMMI-DEV and ISO/IEC 15504. *Software Process: Improvement and Practice*, 9999(9999):n/a, 2009. **Cited** on pages 37 and 77.

[PCCW93] M. Paulk, B. Curtis, M. Chrisis, and C Weber. Capability maturity model for software, version 1.1. *CMU/SEI-93-TR-024*, 1993. **Cited** on pages 3, 17 and 18.

[PK08] John Morley Jeannine Siviy Patrick Kirwan, Lisa Marino. Process architecture in a multimodel environment. Technical report, 2008. **Cited** on page 57.

[PL03] D. Parnas and M. Lawford. Inspection's role in software quality assurance. *Software, IEEE*, 20(4):16–20, 2003. **Cited** on page 134.

[PPW+02] D. Perry, A. Porter, M. Wade, L. Votta, and J. Perpich. Reducing inspection interval in large-scale software development. *Software Engineering, IEEE Transactions on*, 28(7):695–705, 2002. **Cited** on page 134.

[PSTV95] A. Porter, H. Siy, C. Toman, and L. Votta. An experiment to assess the cost-benefits of code inspections in large scale software development. *SIGSOFT Softw. Eng. Notes*, 20(4):92–103, 1995. **Cited** on page 137.

[PSTV97] A. Porter, H. Siy, C. Toman, and L. Votta. An experiment to assess the cost-benefits of code inspections in large scale software development. *Software Engineering, IEEE Transactions on*, 23(6):329–346, 1997. **Cited** on pages 137, 138, 149 and 151.

[PTRC07] Ken Peffers, Tuure Tuunanen, Marcus Rothenberger, and Samir Chatterjee. A design science research methodology for information systems research. *J. Manage. Inf. Syst.*, 24(3):45–77, dec 2007. **Cited** on page 9.

[PV97] A. Porter and L. Votta. What makes inspections work? *Software, IEEE*, 14(6):99–102, 1997. **Cited** on pages 135, 136 and 139.

[PW85] David Parnas and David Weiss. Active design reviews: principles and practices. *Proceedings of the 8th international conference on Software engineering*, 1985. **Cited** on page 136.

[Pyz03] T. Pyzdek. The six sigma handbook. *McGraw-Hill*, 2003. **Cited** on pages 30 and 74.

[Rec12] J. Recker. *Scientific Research in Information Systems: A Beginner's Guide*. Progress in IS. Springer, 2012. **Cited** on pages xvii, 8, 9 and 10.

[Rem05] J. Remillard. Source code review systems. *Software, IEEE*, 22(1):74–77, 2005. **Cited** on page 139.

[RROC85] R.A. Radice, N.K. Roth, A.C. O'Hara, and W.A. Ciarfella. A programming process architecture. *IBM Systems Journal*, 24(2):79–90, 1985. **Cited** on page 118.

[SB97] A. Sweeney and D. Bustard. Software process improvement: making it happen in practice. *Software Quality Journal*, 6:265–274, 1997. **Cited** on page 19.

[SCC02] W.R. Shadish, T.D. Cook, and D.T. Campbell. *Experimental and Quasi-experimental Designs for Generalized Causal Inference*. Houghton Mifflin, 2002. **Cited** on page 149.

[SD86] W.A. Shewhart and W.E. Deming. *Statistical Method from the Viewpoint of Quality Control*. Dover Books on Mathematics Series. Dover, 1986. **Cited** on pages 3, 18 and 27.

[Ses12] Girish Seshagiri. High maturity pays off: It is hard to believe unless you do it. *CrossTalk - The Journal of Defense Software Engineering*, 2012. **Cited** on page 95.

[SF04] J. Siviy and E. Forrester. Accelerating CMMI adoption using six sigma. *CMMI Technology Conference and User Group - http://www.dtic.mil/ndia/2004cmmi/2004cmmi.html - Access in November 2008*, 2004. **Cited** on pages xvii, 39, 40 and 42.

[She01] S.A. Sheard. Evolution of the frameworks quagmire. *Computer*, 34(7):96–98, Jul 2001. **Cited** on page 33.

[SK08a] J. Siviy and P. Kirwan. Multimodel improvement in practice. 2008. **Cited** on page 46.

[SK08b] J. Siviy and P. Kirwan. SEI approach to harmonization. *SEI Webinar Series*, 2008. **Cited** on pages xvii, 33, 34, 35 and 43.

[SKK94] S. Simila, P. Kuvaja, and L. Krzanik. BOOTSTRAP: a software process assessment and improvement methodology. *Software Engineering Conference, 1994. Proceedings., 1994 First Asia-Pacific*, page 183–196, 1994. **Cited** on page 25.

[SKMM08a] J. Siviy, P. Kirwan, L. Marino, and J. Morley. Process architecture in a multimodel environment. *http://www.sei.cmu.edu/process/research/prime.cfm (Accessed March 2010)*, 2008. **Cited** on page 42.

[SKMM08b] J. Siviy, P. Kirwan, L. Marino, and J. Morley. Strategic technology selection and classification in multimodel environments. *http://www.sei.cmu.edu/process/research/prime.cfm (Accessed March 2010)*, 2008. **Cited** on pages 6, 47 and 74.

[SKMM08c] J. Siviy, P. Kirwan, J. Morley, and L. Marino. Maximizing your process improvement ROI through harmonization. *http://www.sei.cmu.edu/process/research/prime.cfm (Accessed March 2010)*, 2008. **Cited** on pages 4, 6, 76, 77, 170 and 192.

[SM06] N. Srivastava and S. Murthy. Harvesting CMMI benefits – the six sigma sickle. *SEPG Conference*, 2006. **Cited** on page 40.

[SMM08] Kirwan Pat Siviy, Jeannine, Lisa Marino, and John Morley. The value of harmonizing multiple improvement technologies: A process improvement professiona's view. *SEI*, 2008. **Cited** on pages 32, 33, 36, 44, 46 and 78.

[SMT92] G Schneider, Johnny Martin, and W Tsai. An experimental study of fault detection in user requirements documents. *ACM Trans. Softw. Eng. Methodol.*, 1(2):188–204, 1992. **Cited** on page 136.

[SPS08] J. Siviy, M. Penn, and R. Stoddard. *CMMI and Six Sigma Partners in Process Improvement*. 2008. **Cited** on page 35.

[SV01] H. Siy and L. Votta. Does the modern code inspection have value? In *Software Maintenance, 2001. Proceedings. IEEE International Conference on*, page 281–289, 2001. **Cited** on page 134.

[Tea10] CMMI Product Team. CMMI for development, version 1.3. Technical report, Carnegie Mellon, 2010. **Cited** on pages xvii and 60.

[Tea11] SCAMPI Upgrade Team. Standard cmmi appraisal method for process improvement (scampi) a, version 1.3: Method definition document. Technical report, Software Engineering Institute, 2011. **Cited** on pages 82 and 83.

[Tin96] M. Tingey. Comparing ISO 9000, malcolm baldrige, and the SEI CMM for software: A reference and selection guide. 1996. **Cited** on pages 36, 46 and 47.

[TM94] M. Thomas and F. McGarry. Top-down vs. bottom-up process improvement. *Software, IEEE*, 11(4):12–13, July 1994. **Cited** on page 18.

[UGI+11] Michael Unterkalmsteiner, Tony Gorschek, A K M Moinul Islam, Chow Kian Cheng, Rahadian Bayu Permadi, and Robert Feldt. Evaluation and measurement of software process improvement - a systematic literature review. *IEEE Transactions on Software Engineering*, PP(99):1, 2011. **Cited** on page 20.

[VK07] V. Vaishnavi and W. Kuechler. *Design Science Research Methods and Patterns: Innovating Information and Communication Technology*. Taylor & Francis, 2007. **Cited** on pages 8 and 9.

[VR03] P. Vitharana and K. Ramamurthy. Computer-mediated group support, anonymity, and the software inspection process: an empirical investigation. *Software Engineering, IEEE Transactions on*, 29(2):167–180, 2003. **Cited** on pages 138 and 139.

[Was06] Hironori Washizaki. Building software process line architectures from bottom up. In Jürgen Münch and Matias Vierimaa, editors, *Product-Focused Software Process Improvement*, volume 4034 of *Lecture Notes in Computer Science*, pages 415–421. Springer Berlin Heidelberg, 2006. **Cited** on page 183.

[Wel93] E. Weller. Lessons from three years of inspection data [software development]. *Software, IEEE*, 10(5):38–45, 1993. **Cited** on pages 135 and 138.

[Wes08] L. Westfall. *The Certified Software Quality Engineer Handbook*. ASQ Quality Press, 2008. **Cited** on page 140.

[Wes09] Linda Westfall. *The Certified Software Quality Engineer Handbook*. 2009. **Cited** on pages 136, 137 and 142.

[WH92] Peter Watts Humphrey. Software process development and enactment: Concepts and definitions. Technical report, 1992. **Cited** on pages 171, 172 and 174.

[Wis06] Alexander Wise. Little-JIL 1.5 language report. Technical Report UM-CS-2006-51, 2006. **Cited** on pages 171 and 178.

[WRH+12] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering*. Computer Science. Springer Berlin Heidelberg, 2012. **Cited** on page 122.

[Xu07] P. Xu. Software process tailoring: An empirical investigation. *Journal of Management Information Systems*, 2007. **Cited** on page 18.

[YYL+04] Chanwoo Yoo, Junho Yoon, Byungjeong Lee, Chongwon Lee, Jinyoung Lee, Seunghun Hyun, and Chisu Wu. An integrated model of ISO 9001:2000 and CMMI for ISO registered organizations. *Software Engineering Conference, 2004. 11th Asia-Pacific*, page 150–157, 2004. **Cited** on page 74.

[Zah98] Sami Zahran. *Software Process Improvement: Practical Guidelines for Business Success*. Addison Wesley Publishing Company Incorporated, 1998. **Cited** on page 17.

[Zah99] Sami Zahran. Software process improvement: practical guidelines for business success. *Journal of Software Maintenance: Research and Practice*, 11(4):285–291, 1999. **Cited** on page 16.

[ZCL05] Xinpei Zhao, Keith Chan, and Mingshu Li. Applying agent technology to software process modeling and process-centered software engineering environment. In *Proceedings*, page 1529–1533, 2005. **Cited** on page 172.