Designing a Measurement Programme for Software Development Projects

Master's Thesis

Richard Kettelerij

Universiteit van Amsterdam

Disclaimer: Due to confidentiality reasons, the *real* name of the company involved in this thesis project is replaced by the *fictitious* name of "Daniro". Other information is left intact.

Document typeset with LaTeX *in Bitstream Charter font. Bibliography formatted with* BibTeX *in Alpha/AMS style. Adobe PDF file generated with* MiKTeX pdfTeX *on 2006-08-14 20:41*

DESIGNING A MEASUREMENT PROGRAMME FOR SOFTWARE DEVELOPMENT PROJECTS

THESIS

submitted in fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

SOFTWARE ENGINEERING

by

ing. Richard Kettelerij
born in Zutphen, the Netherlands

Committee in charge:

Prof. Dr. P. Klint, *Chair*
Prof. Dr. J. van Eijck, *Vice-Chair*
Drs. H. Dekkers, *University supervisor*
Ing. D. Koopman, *Company supervisor*
Dr. J. Vinju, *Committee member*

Institution:
University of Amsterdam
Faculty of Science
1098 SM Amsterdam
The Netherlands
www.science.uva.nl

Company:
Daniro J-Technologies
Plotterweg 1
3821 BB Amersfoort
The Netherlands
(url removed)

Designing a Measurement Programme for Software Development Projects

Abstract

Software measurement is generally recognized as an effective means to understand, control, predict and improve software development projects. However, despite the attention given by the academic community, measurement is far from common practice in certain sectors of the software engineering industry. In the case of Daniro J-Technologies, measurement was found to be used for two purposes; monitoring source code quality and monitoring finances. A *measurement programme* was, however, not in place. Therefore, this research project seeks to design a process-centric measurement programme that suits the specific needs of stakeholders in development projects.

In order to do so, the present author initiated a thorough literature survey. This led amongst others to the discovery of the Goal/Question/Metric method (GQM), which formed the basis of the research approach followed during the thesis project. In light of GQM, a series of interviews were held with stakeholders at different organizational levels. This led to identify five viewpoints and four measurement goals. These goals involved respectively; understanding productivity, defects and scope. In response to these goals, questions were formulated that captured the information needs of stakeholders. In turn these questions led to the definition of a balanced set of (approximately) *thirty* software measures. These measures, together with questions and goals, were validated during an interactive presentation session and prioritized by means of a survey. Yielding an initial set of *thirteen* productivity and defect-related measures (scope measurement was excluded). Furthermore, a prototype Measurement Support System (MSS) was constructed to give stakeholders a more concise and visual representation of the information they could expect from the programme. The indicators (chart and tables) of high priority measures in this MSS were validated during personal walkthroughs. Subsequently feedback from stakeholders expressed during walkthroughs, and during the presentation was incorporated in the current programme.

The above process resulted in the initial definition of a strategic measurement programme, specifically tailored toward Daniro's Java division. Secondly, the project produced a prototype MSS for use in (future) pilot projects. The research also contributed to more subjective matters, such as increased awareness of stakeholders with respect to software process measurement.

Based on the results obtained and the literature consulted, the present author concludes that software measurement in an outsourcing organization is in some ways different from measurement programmes discussed in literature. This is mainly due to the service oriented or customer-intimate nature of outsourcing projects. As a result it is difficult to mandate a fixed set of measures. Nevertheless by concentrating on a number of core improvement areas applicable in any project, a measurement programme can still be defined. Hereby it was found that productivity is a major driver behind measurement information needs. This involves not only productivity in the sense of size vs. effort, but also measures related to schedule adherence and rework time. Although these types of process measures mainly concern managerial stakeholders, the present author believes that engineers can also benefit from this intelligence. Provided that timely feedback sessions are organized. Whether or not the measurement programme is effective in the long run, can only be determined after or during organizational *implementation*. The latter is, however, recommended as future work. As a result, it is currently not possible to verify whether the proposed measures satisfy the goals and questions stated. Nevertheless, the initial *design* of the programme brought a number of important prerequisites in place for Daniro to archive a higher level of measurement and improvement capability.

Keywords: software engineering, measurement, goal/question/metric, process improvement

I don't see how we can have software engineering in the 21st century without measurement

David N. Card[1]

[1] Fellow of the Software Productivity Consortium and Editor-In-Chief of the Journal of Systems and Software. Quote from "The Best Influences on Software Engineering" by Steve McConnell [McC00].

Preface

THIS thesis is the result of the master project carried out between January 9 - April 4 (part time) and April 10 - July 15 (full time) at Daniro J-Technologies. In short, the objective of this master project is to design a measurement programme for use in software development projects. More specific, attention is given to process-related measures that suit the specific needs of Daniro and that provide value to the stakeholders involved.

Originally, my interest in this project was attracted by the strong focus on the concept of software development processes. As such, this project allowed me to cover a wide range of software engineering aspects that I've learned during the past year. From a personal perspective however, this project has been a challenge since it was the first time I did an internship on a less technical subject like "software process measurement". Addressing this subject on a conceptual level (pure thought stuff) while working toward a practical solution wasn't easy. Nevertheless it was exciting and it helped me to expand my horizon.

This thesis would not have been possible without the help of a number of people. First I would like to thank my supervisors Dirk Koopman (Daniro) and Hans Dekkers (UvA) for their valuable feedback and guidance during the project. In addition, I would like to thank dr. ir. Rini van Solingen (LogicaCMG/Drenthe University) for his willingness to cooperate in an interview, which led to some important insights during the project. Furthermore I would like to thank all interviewees; E. Fokker, M. Willemsen, E. Dieleman, H. Jansen, R. Ligtmans, D. Koopman, M. Loggere, K. Grosskop and A. Willemse, as well as all other colleagues at Daniro for their help and support during this project. Above all I want to express my gratitude to my parents for their everlasting care and support. Finally I wish to dedicate this thesis to my younger brother Robert, whose fight for cancer over the last five years strengthened me in this accomplishment. The way you live your life is without doubt *admirable*.

Richard Kettelerij
Maurik, the Netherlands
August 14, 2006

Contents

List of Figures

Chapter 1

Introduction and Motivation

DANIRO J-Technologies is a recently formed organization unit, founded through the acquisition of Solidium B.V by the former Daniro Development Centre Java. The unit is part of Daniro System Integration and Development (SI&D), a division of Daniro that focuses on the development, integration and consolidation of enterprise information systems. In general Daniro SI&D is concerned with two types of activities; outsourcing (project implementation) and consultancy. Especially outsourcing receives a lot of attention lately, since Daniro SI&D aims to become the "national champion in software projects" by expanding its share in this market.

1.1 Context

To distinguish itself from other IT service companies on the market, Daniro SI&D is working towards the establishment of a *software factory* known as "SMART Op Maat"[1] (figure 1.1). In this context the term "software factory" refers to the establishment of a generic project approach in the broadest sense of the word. The idea is that everything from tooling to training to methods and procedures is organized in a uniform way, and applied consistently throughout the various software projects. One should not confuse this concept with the (recent) interpretation of Microsoft that "software factories are the convergence of key ideas in software product lines, component-based development and model-driven development" [Gre03]. Although these technologies can (and will) be utilized in, especially .NET related software factory projects, it is not the primary focus of "SMART Op Maat". The latter is more "a combination of suitable infrastructures, process features, and managerial guidelines" as described in the research of [Aae97].

A central role in the factory concept of Daniro is the use of a development line. This is an integrated set of platform specific tools that support project teams in their software development process. Since Daniro J-Technologies is concerned with Java/J2EE development its development line, SMART-Java, is constructed around a set of (open source) Java tools. Currently the SMART-Java development line offers services such as issue tracking, version management, automated builds and shared storage.

Another key aspect of Daniro's software factory concept is the adoption of the Rational Unified Process (RUP) as a standard project methodology. Since RUP is not meant to be used as an out-of-the-box software process, Daniro decided to tailor the process to its specific needs. This process of RUP tailoring resulted in the publication of a book called "RUP Op Maat"[2].

1.2 Problem definition

Although it is not presented that way, the software factory initiative of Daniro can be seen as a form of *software process improvement* (SPI). In essence Daniro aims to increase the maturity and quality of its software development process through the implementation of a uniform project approach. An important concept in the field of software process improvement is *software measurement*. However,

[1] `http://www.smartopmaat.nl`
[2] `http://www.rupopmaat.nl`

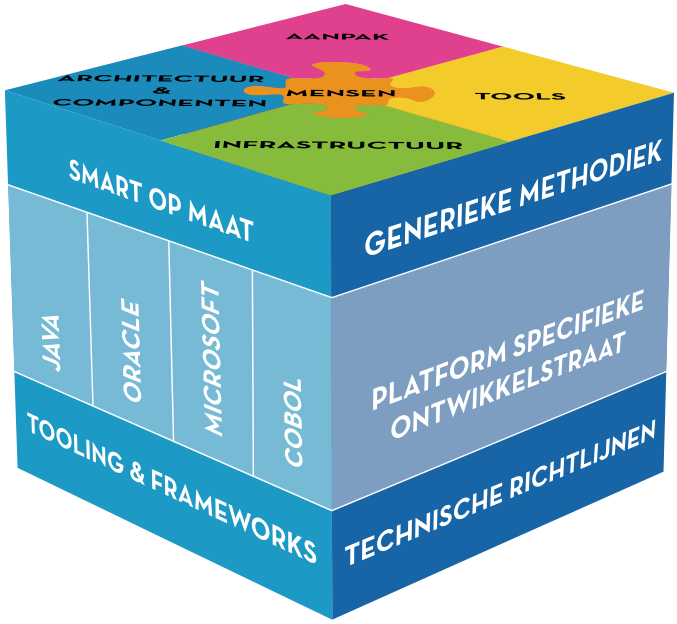


Figure 1.1: *Daniro's software factory concept*

in the case of Daniro J-Technologies this concept has not yet received explicit attention. There have been attempts to introduce software measurement in the *development* organization, but these measures focused only on the quality of source code. The current development line for example is mainly concerned with measures of source code quality through the use of static code analyzers. Furthermore measurement is used in the bidding process (i.e. Function Point Analysis), and on more managerial sides for the purpose of financial monitoring (i.e. hour-logs). Until now, however, little effort is put in a rigorously defined set of measures that focus on the entire software process, and provide value to different stakeholders involved in software development projects.

Since measurement is generally recognized [Abr04] as a key factor in the understanding, management and improvement of software related activities, it is worthwhile to devote more attention to this subject. Therefore the idea was raised to investigate the possibilities of establishing a *measurement programme* at Daniro J-Technologies for use in software development projects.

1.2.1 Scope

Establishing a measurement programme is, like most other software process improvement initiatives, a cost and resource intensive operation. Therefore the scope of this research is limited in a number of ways. Most notably, the primary concern of this research is the *definition* of a measurement programme. Although some form of implementation is required to demonstrate the validity of the programme, there are important reasons to initially focus on definition activities. For example without proper definition the purpose of a measurement programme is unclear, and one could well be measuring the wrong things. Consequently the information gathered may not satisfy the needs of stakeholders, which could result in deceased buy-in. Furthermore without a defined programme, data collection is unstructured and more error prone. Also it is often not clear how to analyze and interpret measurement results. Overall, definition and planning is considered to be an important prerequisite for successful measurement. Since it provides the programme with a a clear focus and avoid unnecessary costs.

Furthermore this research is limited to *process*-related measures, since these measures haven't been explicitly addressed yet (at Daniro). In addition the research focuses on software *development* projects, since these type of projects account for the largest portion of all software-related projects and are therefore critical to the business. Altogether the objective of this research is to *design* a measurement programme that suits the specific needs of Daniro J-Technologies.

1.2.2 Research question

The central research question of this thesis is formulated as follows:

> *What process-related measures, with respect to organizational goals, can be defined to satisfy the information needs of stakeholders in the context of software development projects?*

In order to answer this central research question, a number of subquestions have been defined. These questions are listed below and grouped by three research topics, respectively: *Information* which concerns organizational goals and the information needs of stakeholders. *Measurement*, which deals with process-related measures that fit within the context of software development projects. *Organization*, which roughly concentrates on the context of the programme, and the costs/benefits associated with implementation.

Information

1. What purpose(s) does the measurement programme serve?
2. What goals does the organization tries to archive?
3. What stakeholders are involved in software development projects?
4. What stakeholders have an interest in the measurement programme?
5. What information do stakeholders require from the measurement programme?

Measurement

6. What kind of methods or techniques exist for defining a measurement programme?
7. What process-related measures are available in literature and at Daniro?
8. What process-related measures satisfy the information needs of stakeholders?
9. What process-related measures are feasible in the context of development projects?
10. What information about process-related measures should be specified in the programme?

Organization

11. To what extend do environmental factors influence the measurement programme?
12. What are the critical success factors in the establishment of a measurement programme?
13. What guidelines are important for measurement programme implementation?
14. What benefits are to be expected from the measurement programme?
15. Does software measurement fit within the context of Daniro's development projects?

1.3 Outline

Subsequent chapters elaborate various aspects of the research questions listed above. To start, the next chapter addresses the background and context of this research by a discussion of relevant literature. Thereafter chapter 3 discusses the approach and methodology of the research, as well as the scientific and practical relevance. Chapter 4 summarizes the process of environment characterization and goal identification. Chapter 5 describes stakeholder information needs and the constraints of the measurement programme. Chapter 6 discusses the process of measurement definition in response to these information needs. Chapter 7 describes the work on implementation aspects, such as data collection and measurement support tools. Finally, chapter 8 concludes and provides directions for future work.

Chapter 2

Background and Context

BEFORE addressing the research question of this thesis a review of existing literature is required. This contributes to a basic understanding of the process measurement field and the related concepts. Moreover, the literature study performed throughout this project resulted in a subdivision of existing work in several categories. These categories are discussed in subsequent sections of this chapter.

2.1 Software Process

ISO standard 15504[1] describes the term *software process* as "the process or set of processes used by an organization or project to plan, manage, execute, monitor, control and improve its software related activities". This definition illustrates that the process field in software engineering includes a wide range of activities. From a measurement perspective these activities can be grouped in two areas; software process improvement and software engineering management. The paragraphs below discuss the two research areas in detail.

2.1.1 Software Process Improvement

Software Process Improvement (SPI) is an extensive research area. There are many models, standards, and methods that can be used to improve the state of software engineering practice within an organization. An interesting classification of SPI related methods is given by Cannegieter in [Can03]. Cannegieter distinguishes three types of methods: *system development methods*, *quality models* and *project management methods*.

System development methods such as eXtreme Programming (XP), SCRUM, Rational Unified Process (RUP) and the Dynamic Systems Development Method (DSDM) define activities, artifacts and roles that are necessary to develop software products. If the scope of a SPI initiative is "system or application development", then organizations can use these methods to create or tailor their software development process accordingly. On the other end of spectrum there are methods such as; Software Process Improvement and Capability dEtermination (SPICE/ISO 15504), TickIT, BOOTSTRAP, the Capability Maturity Model (CMM) and its successor CMMI [Can06]. These methods serve as reference models and define requirements that an organization should meet in order to reach a particular maturity level. However, these methods do not specify *how* to implement the software process. If an organization wants to attain a certain CMM(I) level, it should implement its software process in a way that complies with the requirements of the desired maturity level. Finally an assessment can be performed to identify (and possibly certify) the maturity of the organization, as well as to propose relevant improvements. The last type of methods, identified by Cannegieter, focus on the managerial side of process improvement. These methods, such as PRINCE2, are supplementary to SPI, and not discussed in further detail. Management in the context of software engineering, however, is not a trivial matter and therefore elaborated in §2.1.2.

Apart from the classification of Cannegieter, Solingen *et al* [Sol99] divide the SPI area in two streams: top-down and bottom-up. The quality models of Cannegieter; CMM, SPICE and BOOT-

[1] `http://www.sei.cmu.edu/iso-15504/`

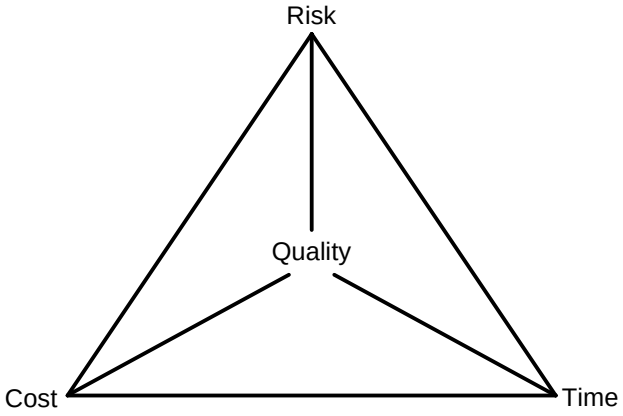


Figure 2.1: *Software Process Improvement focus areas [Sol99]*

STRAP all classify as top-down approaches, since these methods are based on assessment and benchmarking. On the contrary, bottom-up approaches such as the Goal/Question/ Metric (GQM) §2.3.1 and the Quality Improvement Paradigm (QIP) are based on the application of *measurement* as the basic guide for process improvement.

The focus of this master research will be on the measurement-based (bottom-up) stream of software process improvement. However, the actual focus of improvement is unknown at this time. In [Sol99] four main areas of software process improvement focus are identified. As illustrated in figure 2.1 these areas are: *quality*, *cost*, *risk* and *time*. Quality improvement usually starts with some kind of defect detection and defect measurement, often via inspections. Also subjective measurement such as customer satisfaction or documentation quality can be defined as improvement goals. Cost improvement is mainly concerned with a more efficient development process. Measurement in this context is often related to size, such as costs per line of code or costs per function point (§2.2.2). Improvement goals concerning risk are mainly targeted toward managing risk factors, by applying measurement to risky areas in the development process (e.g. requirements engineering). Finally, time related measurement and improvement is concerned with aspects such as productivity and time-to-market.

2.1.2 Software Engineering Management

A research area closely related to SPI, and especially important from a measurement point of view, is Software Engineering Management (SEM). The Software Engineering Body of Knowledge (SWEBOK) defines this area as "the application of management activities - planning, coordinating, *measuring*, monitoring, controlling, and reporting - to ensure that the development and maintenance of software is systematic, disciplined and quantified" [Abr04]. The notion of *project management* plays an important role in this area. In general, project management is an umbrella for many activities in software engineering (as well as other engineering disciplines). Typical activities include, amongst others; task planning, risk management, cost and resource estimation, process control and contract management. Together these activities evolve around five primary attributes namely: time, money, quality, information and organization [Gri00]. During a project it is the responsibility of the project manager to control (and possibly improve) the application of these attributes. One way to accomplish this is by taking decisions based upon quantitative data. Measurement is hereby of important [Abr04].

However it should be noted that measurement is a means to and end, not and end itself [Bas95]. The insight and experience of the project manager and his staff (particularly their understanding of social issues) are at least as important in successful project management. This relates, amongst others, to awareness of possible risks associated with measurement (mis)use. Such as; *dysfunction* (i.e. forcing people to make measures look better) and *distortion* (i.e measurement results distort people's behavior, causing them to provide less value to the organization) [Kan04].

2.2 Software Measurement

Software measurement can be loosely defined as "the process of defining, collecting and analyzing data on the software development process and its products in order to understand and control the process and its products, and to supply meaningful information to improve that process and its products". [Sol99]. As the definition indicates there are several reasons for conducting software measurement. The first reason is to *understand* the product, process or resource in question. This may lead to the establishment of a baseline for future comparisons. Once basic understanding is reached, measurement information can be used to *control* a particular product, process or resource. This involves performing corrective and perfective actions. Thereafter an analysis of measurement data, can help to identify opportunities and inefficiencies in products, processes and resources in order facilitate *improvement* actions. Finally as part of improvement, measurement information can be applied to *predict* the development of products, processes and resources over time [Par96].

The (widely accepted) classification in *product-*, *process-* and *resource*-oriented measures is suggested by Fenton and Neil [Fen00]. In general, product measures describe the (quality) characteristics of the product under development. Example measures are: size, complexity, performance and the level of test coverage. Process measures on the other hand are used to characterize software development and maintenance processes. Typical process measures include: development time, effort, cost and the number of requirement changes. Finally, resource measures describe the characteristics of the project or organization under consideration. Resource measures relate to: productivity, cost, schedule and maturity. The classification of Fenton and Neil provides valuable insight in the measures research field, and gives a handful reference to focus on a particular type of measures (*process* in this case). Nevertheless, the three types cannot be considered in isolation. There's a large overlap, which is illustrated by use of the Metrel (Metric Relationship) rules [Woo01]. The Metrel rules state that: For any valid *product* measure, its derivative with respect to time is a valid *process* measure. Subsequently, for any valid *process* measure, its derivative with respect to time is a valid measure for the *organization*[2]. In example:

> The number of defects in a system is a useful and valid *product measure*. Then the rate of insertion of defects into code per phase (the error proneness of the methodology) is a valid *process measures*. The rate of removal of defects per inspection (the test efficiency) is also a valid measure of the *process*. And the rate of improvement in test efficiency, over a series of projects, is a valid *organization measure* [Woo01].

In the field of software measurement there are a number of important (research) areas that need additional explanation. An introduction to two of these areas, *measurement theory* and *functional size measurement*, is provided in the remaining sections of this paragraph.

2.2.1 Measurement Theory

In general *measurement* is defined as "the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules" [Fen94]. In this context an *attribute* is seen as a measurable property of an entity. For example, "size", "complexity" and "testability" are attributes of entity "source code". Underlying these basic definitions is a principle known as the *representational theory of measurement*. This is an extensive subject concerned with the mapping of relations between entities and attributes. A fundamental concept in this theory is the notion of an *empirical relational system*. This system represents the entities in the real world and the empirical knowledge of the entities' attributes. Intuitive understanding of attributes gives rise to relations between entities. For instance, if one is interested in "document length" the relations "is longer than" (document A is longer than document B) can be formulated. In order to measure an attribute represented by an empirical relation system, one needs to map the entities and relations from the empirical system to a *numerical relational system* (formal world), with respect to the *representational condition*. This condition implies that for every relation defined in the empirical system, there is a equivalent relation defined

[2] In this context the term *organization* is similar to *resource*

on the measures of those entities in the numerical system. The link between both worlds is expressed through measures and scales. Hereby distinction is made in two types of measures; *direct* and *indirect*. A direct measure, i.e. "document length", is not dependent upon a measure of any other attribute. However, indirect measures such as "productivity" do involve measures of other attributes. Note that the term *measure*[3] is explicitly used in this thesis to differentiate from the definition of measurement. Use of the term *metric* is avoided since there is no general/unambiguous definition known in the context of software engineering [Gar06; Off97; Par96].

In measurement theory, scale types determine the kind of statements that can be made about measurement data. A scale type of a particular measure is determined by the *admissible transformations* that can be made to the scale of that measure. That is, meaningful statements about a value in one scale should also apply to values of the same measure in another scale. Scale types commonly used in software measurement are [Par96]:

- *Nominal*: Objects with the same scale value are equal on some attribute. For example; programming languages (Java, C#, Cobol, Pascal).
- *Ordinal*: Objects with a higher scale value have more of some attribute. For example; a quality rating (+, ++, +++).
- *Interval*: A certain distance along the scale means the same thing, regardless of position. For example; the difference between 5 and 7 is the same as the difference between 10 and 12.
- *Ratio*: Equal to interval scales, except that there is a true zero point. For example; a building with a height of 100 meters is twice as large as a building of 50 meters.
- *Absolute*: Used when there is only one possible way to measure an attribute.

Principles from measurement theory, such as the representation of scale types, are controversial in software engineering research (see the debate between [Kit95], [Mor97] and [Kit97]). For example, some scientists argue that properties that imply or exclude measurement scales in the definition of a measure cannot be used since scales change, depending on the questions ask during data analysis [Kit95]. Others disagree, and claim that without such properties one abstracts away all relevant measurement structure and limits the ability to say anything of interest [Mor97]. However, overall most scientist agree that attributes in software engineering, e.g. "correctness", are not yet sufficiently understood to be certain about a particular type of measurement scale [Fen94].

2.2.2 Functional Size Measurement

Quantifying size is one of the most basic activities in software measurement [Gra94]. A frequently used size measure is "lines of code" (LOC). Although LOC can be useful in many, especially coding related, situations it has a number of drawbacks [Low90]. First of all the number of LOC depends on the implementation language and coding style used. Secondly, LOC cannot be estimated *a priori* to implementation. Only when software construction is (partially) finished it becomes possible to faithfully count the number of lines in a software product. In response to these issues many organizations rely on *functional* size measurement for their products/projects. In short, functional size measurement is aimed at measuring the size of the software product from the perspective of what gets delivered to the (end)user.

Function Points

The idea of measuring software size in terms of functionality as opposed to physical components such as LOC, was first put forward by Albrecht in '79. Albrecht introduced the concept of "function points" and the accompanying Function Point Analysis (FPA) method [Low90]. FPA is a structured technique to measure software size by quantifying *functionality* in the form of *function points*, based on requirements and design. The technique breaks the system into smaller components in a way that these can be analyzed and understood. Function point counts can be applied to development projects, maintenance projects, and existing applications. There are five major components of

[3] The assignment of numbers or symbols to entities, in order to characterize a specific attribute (derived from [Gar06])

FPA which capture the functionality of the application. These are: External Inputs (EI), External Outputs (EO), External Inquiries (EQ), Internal Logical Files (ILF) and External Interface Files (EIF). The first three are treated as user transactions, and the last two are referred to as logical data collections.

At first, FPA was only used to measure productivity after the completion of development and maintenance activities. However, soon it became clear that FPA could also be used to support the software process in its inception, since the required data can be available early in the project. This aspect made FPA a widely adopted technique for software engineering tasks such as cost and resource estimation. This development puts heavy responsibility on the use of function points. Since accurate cost estimation is vitally important in software process improvement [Miz98] and especially interesting in terms of process measurement. To conclude, a typical[4] function point analysis consists out of the following steps [Low90]:

- Determine the type of function point count (users, purpose, dependencies)
- Identify and rate user transaction- and logical data collection functions, to calculate their contribution to the Unadjusted Function Point count (UFP)
- Determine the Value Adjustment Factor (VAF) by using general system characteristics
- Finally, calculate the adjusted function point count

Use Case Points

In addition to FPA, many modern development projects working on object oriented software apply the concept of Use Case Point (UCP) analysis [Car05] to estimate project costs. UCP is in many ways the same as FPA, since use cases consist of goals and scenarios that provide functionality to a business domain. Therefore these specifications can be used to provide insight into an applications complexity. To derive size and effort estimates, one needs to examine the actors and scenarios of the use case. In a nutshell, use case points are determined by the complexity of the actor, the number of transactions in the scenario, the estimated technical weight of the implementation and the experience of the organization. UCP was originally invented at Rational Software in '93, but over the years multiple modifications were made to the original method. Unfortunately until now no official standard on UCP analysis exists. This makes it hard to compare UCP results of different projects (and organizations) with each other.

2.3 Measurement Methods

Over time various methods have been proposed to implement software measurement programmes in development organizations. Well known are; the Goal/Question/Metric (GQM), Practical Systems and Software Measurement (PSM) and Statistical Process Control (SPC). A popular instance of the latter method is Six Sigma. In general however, SPC based methods are dependent on a well-defined and repeatable software processes (say CMM-4). This is in most organizations (including Daniro) not the case. Therefore only the two most practical and promising methods (§3.2), GQM and PSM are discussed in further detail.

2.3.1 Goal Oriented Measurement

Goal Oriented Measurement can be described as "the definition of a measurement programme based on explicit and precisely defined goals that state how measurement will be used" [Bri96]. The most widely known method for applying goal-oriented measurement is the Goal/Question/Metric (GQM) method [Sol99]. The principle behind GQM is that measurement should be goal-oriented. Therefore organizations have to define their measurement goals based upon corporate goals. Subsequently, in order to improve their process, organizations need to transform these goals into activities that can be measured during the execution of projects. These actions take place in a top-down fashion. As illustrated in figure 2.2; goals are refined into question that in turn translate to measures. The opposite is true for the analysis and interpretation steps. By measuring

[4]There are (minor) differences between the available function point guidelines. However, the two largest function point users groups, IFPUG and NESMA, are working together for almost sixteen years now to eliminate these differences.

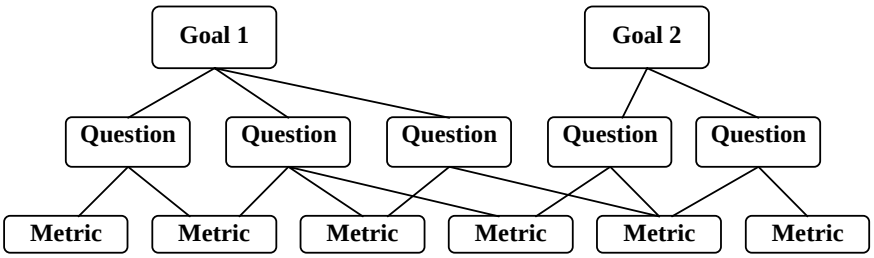


Figure 2.2: *The Goal/Question/Metric (working both top-down and bottom-up) [Bas94]*

attributes, questions can be answered that in turn led to identify whether or not goals are reached. To facilitate these actions, the GQM method contains four phases that are as listed below. These phases can be executed sequentially, however it is also possible to incorporate the GQM in the six step Quality Improvement Paradigm (QIP) [Sol99; Lat98; Bri96]; resulting in an integrated software process improvement and measurement method.

1. *Planning* phase, in which a project for measurement is selected, defined, characterized and planned. This phase results in a project plan.

2. *Definition* phase, in which the measurement programme is defined (goal, questions, measures, hypotheses, ...) and documented (in GQM-, measurement- and analysis plans).

3. *Data collection* phase, where the actual gathering of (raw) measurement data takes place.

4. *Interpretation* phase, the collected data is processed into measurement results that provide answers to the defined questions. After this phase goal attainment can be evaluated.

Goal/Question/Metric evolution

The GQM method described above reflects to a large extend the original GQM approach, invented by Basili *et al* (summarized in [Bas94]). Originally, the GQM approach was developed to evaluate defects in projects of NASA' Software Engineering Laboratory (SEL) [Bas95]. Over time, however, the method gained in popularity and since then numerous extensions have been made. A few notable GQM extensions, relevant in the context of this research, are highlighted below.

In '96 Park *et al.* [Par96] studied the GQM method at the Software Engineering Institute (SEI). As part of their research, they extended the method with an extra aspect known as an *indicator*: a visual representation (e.g. chart or table) of data which helps to answer specific questions. Park *et al.* state "our experience is that sketches of pictures and displays helps significantly in identifying and defining appropriate measures". Furthermore, Park *et al.* paid explicit attention to the role of *mental models* in the GQM definition process. In short the addition of, amongst others, indicators and mental models resulted in the establishment of the *Goal/Question/Indicator/Metric* method. This GQ(I)M method consists out of the following ten steps:

1. Identify your business goals.
2. Identify what you want to know or learn.
3. Identify your subgoals.
4. Identify the entities and attributes related to your subgoals.
5. Formalize your measurement goals.
6. Identify quantifiable questions and the related indicators that you will use to help you achieve your measurement goals.
7. Identify the data elements that you will collect to construct the indicators that help answer your questions.
8. Define the measures to be used, and make these definitions operational.
9. Identify the actions that you will take to implement the measures.
10. Prepare a plan for implementing the measures.

In '97 a European research initiative known as PERFECT [Bir97] was started. The goal of this initiative directed by Fraunhofer IESE, was to assist in measurement-based improvement of software processes. In this light a number of techniques, methods, and tools were developed. Hereby, attention was also given to the GQM method. A notable extension resulting from this project is the distinction between *project*-level and *strategic*-level measurement. Until then GQM was mostly used for defining goals, questions and measure *within* a single project. The PERFECT booklet, however, summarizes the process of applying GQM to *organizational* goals and issues in *multiple* projects.

In '99 Solingen and Berghout [Sol99] published the first book on GQM. This book was largely based on master projects performed at Schlumberger RPS (involving various aspects of GQM), and partially based on Solingen its Ph.D project [Sol00]. Apart from existing work on GQM, the book emphases the importance of providing *feedback* to stakeholders with respect to measurement data collected in software projects. The primary instrument used for this purpose are *feedback sessions*. During these sessions measurement data is analyzed, presented and interpreted by project members. The latter helps to increase the overall learning effect of measurement programmes. Furthermore, the book discusses a few other extensions to GQM; such as *process modeling* [Bri96].

2.3.2 Practical Software and Systems Measurement

Practical Software and Systems Measurement [Jon03b; Jon03c] or PSM in short, is a measurement methodology initiated and sponsored by the United States Department of Defense (DoD). PSM served as the reference model for ISO measurement standard 15939, which in turn was used as the basis for the Measurement and Analysis Key Process Area (KPA) of CMMI (§2.1.1). Important detail of the method is that PSM is "information driven" instead of goal-driven, like GQM. Implementing a measurement programme using PSM includes, defining organization and project *information* needs and then *selecting* measures that supply information relative to those needs. These measures can be selected from several categories: schedule and progress, resources and cost, product size and stability, product quality, process performance, technology effectiveness and customer satisfaction. In contrast to GQM, PSM is specifically designed to meet *management* information needs.

Although there are other (important) differences, the PSM and GQM method are in many ways the same when viewed from a purely technical perspective. For instance PSM evolves like GQM around four comparable phases, as illustrated in figure 2.3. Furthermore the success factors for PSM are similar to those of GQM.

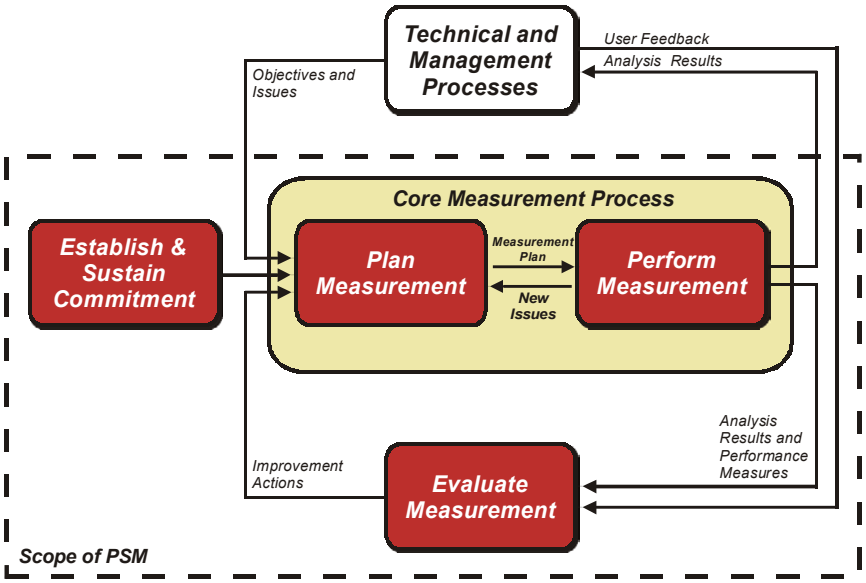


Figure 2.3: *Four phases of the PSM measurement process [Jon03c]*

2.4 Success Factors

Although software measurement has been a topic of research for many years, establishing a measurement programme is still considered a challenging undertaking [Bri96]. Few organizations completely succeed in their measurement attempts (some reports suggest a failure rate up to 80% [Wie97; Das92]), which is disturbing since measurement programmes are often cost intensive. Typically measurement accounts for 2% till 4% [Hal97; Bas95] percent of the total project costs. However, these numbers should be taken with care since the *context* (organization culture, maturity) and the *size* (number of measures, frequency of collection) greatly determines the overall costs of a measurement programme. Furthermore the benefits gained from a measurement programme are difficult (altough not impossible [Sol99]) to quantify.

Nevertheless, since software measurement is not a trivial matter, numerous publications appeared to discuss the critical success factors in the establishment of measurement programmes. The exact factors differ per publication, however there are a number of similarities to detect. Therefore the present author decided to assemble a *consensus list* of success factors by means of a literature survey. Sources reviewed are: Bassman *et al* [Bas95], Hall and Fenton [Hal97], Jones [Jon03b; Jon03c], Niessink and Van Vliet [Nie01], Park *et al* [Par96], Solingen [Sol00, §7.2.3] and Berghout [Sol99], Rifkin and Cox [Rif91], and Wiegers [Wie97; Wie99]. The list of success factors is presented below. Note that phrases printed in bold are *consensus* factors, defined by the present author. Related success factors found in literature are cited *below* the bold text.

1. **Begin small and extend the programme as you go**
   - Start small (with common goals and issues) [Wie99; Rif91; Par96; Jon03b; Bas95]
   - Incremental implementation [Nie01; Hal97]
   - Focus initially on project level analysis [Jon03c]
2. **Motivate managers and gain commitment**
   - Commitment from project managers secured [Nie01; Hal97]
   - Motivate managers [Rif91]
   - Demonstrate commitment [Jon03b]
3. **Measurement must be goal-oriented**[5]
   - Goal-oriented approach [Hal97]
   - Measurement should be goal-oriented to ensure a limited but relevant set of measures [Sol99; Sol00]
   - Well-planned metrics framework [Nie01]
   - Design a consistent measurement process that is linked to organizational goals and objectives [Par96]
   - Use issues and objectives to drive the measurement requirements [Jon03c]
   - Understand the goals [Bas95]
4. **Provide training to people affected by the programme**
   - Educate and train [Rif91]
   - Provision of training for practitioners [Nie01; Hal97]
   - Provide adequate training [Jon03b]
   - Create a short training class [Wie97]
5. **Involve multiple stakeholders in programme definition and implementation**
   - Involve all stakeholder [Rif91]
   - Involvement of developers during implementation [Nie01]
   - Developer participation [Hal97]
   - Acquire quality models of the people involved [Sol99; Sol00]
6. **Define the measurement programme precisely**
   - Define data items and procedures [Wie99]
   - Design a consistent measurement process that includes rigorous definitions [Par96]
   - Use a rigorously defined set [Rif91]

[5] The U.S. DoD Data and Analysis Center for Software (DACS) assembled a list of success factors *specifically* relevant to GQM-based measurement programmes. These factors have been derived from various sources, among others [Sol99; Lat98]. The list is retrievable via `http://www.goldpractices.com/practices/gqm/index.php#detail`

- Define and collect measures based on the technical and management processes [Jon03c]

7. **Update and refine the measurement programme continuously**
   - Take an evolutionary approach (and plan to throw one away) [Rif91]
   - Constantly improving the measurement program [Nie01]
   - Design a consistent measurement process that continuously evolves [Par96]
8. **Provide regular feedback to people involved in the programme**
   - Organize feedback sessions and let the data be interpreted by the people involved [Sol99]
   - Provide feedback to practitioners [Hal97]
   - Feedback to developers [Nie01]
   - Share the data [Wie99]
   - Provide regular feedback to your teams about the data they have collected [Wie99]
9. **Automate where possible, but recognize limitations**
   - Automate collection and reporting [Rif91]
   - Use automated data collection tools [Hal97; Nie01]
   - Do not expect to fully automate data collection. [Bas95]
   - Do not limit data collection to tools. [Sol99]

Obviously this is *not* a complete list, one can think of many more factors critical to a successful measurement programme (to illustrate; the present author found a total of $\pm$ 85 success factors in the reviewed literature). Nevertheless, the above list gives a comprehensive overview of success factors considered to be important by *multiple* scientists. Subsequently the list provides an answer to research question nr. 12; "what are the important success factors in establishing a measurement programme", as posed in paragraph 1.2.2.

Chapter 3

Research Method and Approach

BASED on the problem definition stated in chapter 1, and the information gathered during the literature survey (chapter 2), a suitable research method and approach was chosen. The purpose hereof is to provide both practical as well as scientific value to the field of software process measurement.

3.1 Methodology

The methodology used in this project makes up a combination of *exploratory* and *constructive* research. Since it is currently unknown whether it is actually possible to establish an effective measurement programme for use in (all) Daniro' software development projects, an exploratory approach is desirable. As a consequence no hypotheses are formulated in advance. On the other hand this research is constructive in the sense that work is carried out toward the establishment of a practical solution; a measurement programme "design". However, since time and resources are limited no empirical data of this construct can be gathered. Validation is therefore primarily *qualitative* in nature.

3.2 Approach

The GQM method, as discussed in §2.3.1, is used as a basis for the design (i.e. planning/definition) of the measurement programme. The reason for selecting GQM is threefold: First GQM is explicitly focused on *measurement*-based process improvement, as opposed to SPI methods such as CMMI (§2.1.1). Second GQM is specifically targeted at software development activities, unlike for example Six Sigma (§2.3). Third GQM is *not* focused on a single type of stakeholder, like PSM (§2.3.2).

Since GQM is selected as the method of choice, the phasing of this research project is set-up accordingly. Roughly speaking, the research starts with environment characterization and goal definition by means of a series of interviews. Based on interview outcomes, goals are defined and prioritized. Thereafter, questions are formulated and validated in an interactive presentation session. In parallel, effort is put in studying and defining measures that provide value to the questions specified. Subsequently measures are prioritized by means of a survey. After that, measurement collection, analysis and interpretation activities are addressed. Therefore a prototype Measurement Support System (MSS) [Sol99] is build and validated by means of personal walkthroughs. Finally, near the end of the project, concrete recommendations are given and future work is discussed.

As can be seen from figure 3.1 this research project follows a type of *funnel* approach. That is, the initial scope starts wide since it is unknown what interests the measurement programme should serve. However, as the project progresses the scope narrows until eventually a balanced set of measures is defined. After implementation aspects (introduction, data collection and interpretation) and measurement validation have been addressed, the scope is widened again and a conclusion is provided. Although a literature survey is performed at the beginning of this project, literature research flows down through the entire research. That is, literature is used to *gather* information as well as *validate* observations. Important detail of this approach is the level of attention for measurement *success factors* (§2.4). By explicitly addressing a number of vital factors

during programme definition, the present author aims to increase the likelihood of measurement (implementation) success. This decision is in line with work of Offen *et al.* [Off97], who uses a set of success factors to bootstrap their M^3P measurement framework (which also incorporates GQM).

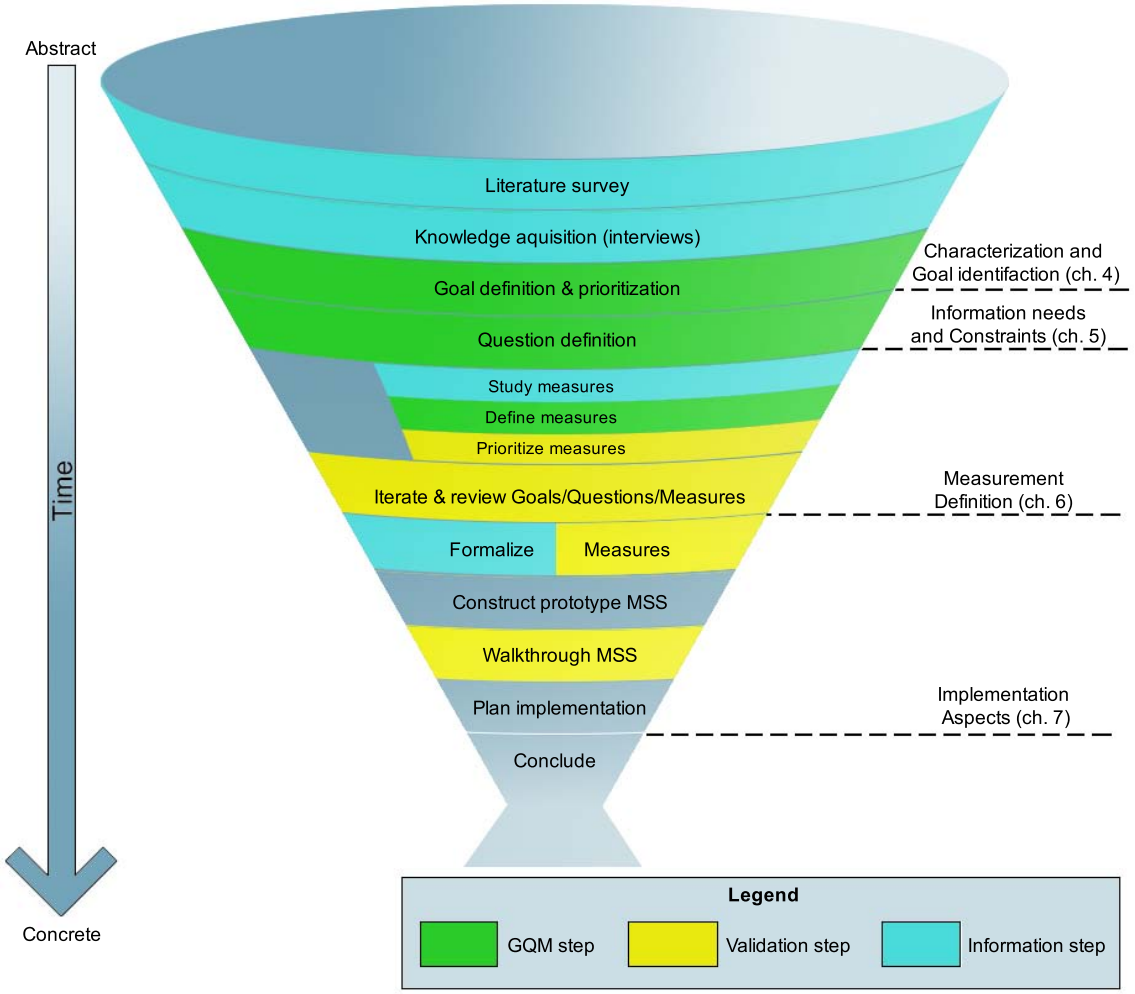


Figure 3.1: *Funnel approach; overview of thesis phases and activities*

Note that figure 3.1 only illustrates the general *sequence*, *scope* and *purpose* of activities in the thesis project. The amount of time invested in activities and the duration of phases isn't displayed.

3.3 Contribution

The actual *design* of the measurement programme accounts for the most concrete *practical* contribution of this research project. That is, the list of goals, questions and measures together with rationales, formal descriptions, and implementation directions. This also includes the prototype Measurement Support System; constructed for the purpose of measurement programme validation and use in (future) pilot projects. A less concrete but more *social* contribution, lies in the involvement of stakeholders during the measurement definition process. This involvement creates goodwill and can be seen as the first steps toward the establishment of a "measurement culture" [Wie99]. From a more *scientific* point of view, the relevance of this research is demonstrated through the use of GQM in an ICT service/outsourcing organization (Daniro). The literature research by the present author found little to no experiences of GQM and/or measurement programme use in these types of organizations.

Chapter 4

Characterization and Goal Identification

ONE of the first steps undertaken in the design of the measurement programme is environment characterization and goal identification. This chapter elaborates the process followed and the results obtained.

4.1 Project environment

In order to understand the goals and interests of stakeholders it's helpful to know about the way projects are organized at Daniro. In a nutshell, the project manager is responsible for controlling resources and budget. The amount of budget and resources allocated to a specific project is, however, decided by organization management during internal steering group meetings. As illustrated in figure 4.1 there are two steering groups. The internal steering group is headed by the Director of Operations, while the external steering group is headed by customer representatives (project owner, project leader, ...). The contract manager forms the link between the two groups. Usually a project group is divided in small teams that contribute various expertises (e.g. software construction, testing, functional design) to the project. These teams are often headed by a team leader; an expert in the field with broad responsibilities. Note; even though the Project Office is placed above projects, it is actually a staff department that *assists* project managers in controlling a project.

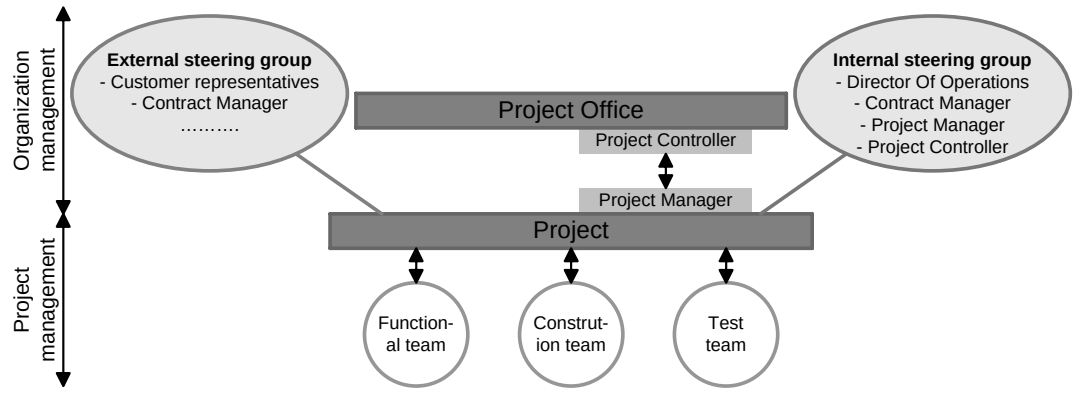


Figure 4.1: *Organization of projects (visualized in concordance with Business Unit Manager)*

4.1.1 Progress reporting

Apart from all kinds of project specific documentation and reports (mandated by methodologies such as "RUP Op Maat" or PRINCE2, §1.1), project managers are required to submit periodic *progress reports* to Daniro's Project Office. There are two types of progress reports: *quantitative* reports containing financial data (including forecasts) and *qualitative* reports that explain the (subjective) status of the project in terms of risks and milestones. The quantitative reports contain

a list of project members with their respective hour rates. These rates are combined with logs from the hour registration system to calculate cost expenditures and visualize financial trends. Based on these reports, steering group meetings are organized and decisions with respect to project progress and costs are made.

4.2 Knowledge acquisition

In order to define the *focus* of the measurement programma, knowledge needed to be acquired with respect to the current state of practice, improvement opportunities and business goals underlying the project organization (figure 4.1). Therefore a series of interviews were held with people at different organizational levels who play a key role (or represent a group that plays a key role) in the successful establishment of a measurement programme. After conducting a number of informal interviews, a series of semi-structured interviews was started. In total, nine semi-structured interviews with different types of stakeholders were carried out. The interviewees included; the Director of Operations, the Director of Innovation, two Project managers, one Project controller, one Business Unit/Contract Manager, one Test leader, one Functional lead and a Senior Software Engineer.

In order to cope with different stakeholder disciplines, the interview questions were split in two sections; a generic part and a stakeholder specific part. This offered the advantage of addressing stakeholders with relevant questions while preserving the option to compare interview outcomes. An overview of the interview questions is provided in appendix A. This list also includes the standard "seven questions" [Lat98; Sol99] for goal identification. To allow interviewees to prepare themselves, all question lists were sent in advance. Furthermore to avoid missing relevant information the interview conversations were, if appropriate, registered on a memo-recorder.

4.3 Goals

An in-depth study into the (commercial) factors that drive software development projects is beyond the scope of this thesis. Nevertheless, in order to define a measurement programme, a basic understanding of the business goals that Daniro aims to archive with respect to development projects is required. The interview with, primarily, the Director of Operations helped to identify the following:

B1. Be competitive in the enterprise Java (J2EE) project market
B2. Deliver software systems of sufficiently high quality

Although these goals are fairly general, a number of (market) factors are of particular importance. For instance *costs*, which is one of the primary factors when it comes to competing on the project market. Moreover, an increase in productivity is seen as *the* most important factor in lowering overall project costs. However, the relation with customers is also given a high priory. Especially since Daniro tries to acquire the majority of its profit from a limited group of (approximately forty) clients. As a result *quality* is a factor that requires attention in development projects. This refers to both product quality (i.e. conformance to specification) as well as customer satisfaction (i.e. possibility of gaining future contracts).

The (high-level) business goals above give a general indication about possible measurement areas at Daniro. However, in order to define a starting point for the measurement programme, it is important that the organization first selects an area in which it wants to improve. The interview results indicated the following (informal) improvement goals:

I1. Baseline productivity
I2. Minimize defects
I3. Manage project scope

The improvement goals mentioned above are the foundation of the measurement programme. However from a measurement perspective, the goals are too abstract. Therefore it is necessary to translate these improvement goals into measurable goals with help of the GQM goal template [Sol99]. This template helps to describe the purpose, perspective and context of goals in a clear and understandable way. The present author concludes, however, that in this case the standard GQM

goal template (of [Sol99] or [Par96]) isn't expressive enough to capture all necessary information. Since GQM-style goals already represent a *consensus* between the goals of different stakeholders, information about individual stakeholder concerns is lost. Explicitly documenting these concerns and their origin is important, because one objective of this measurement programme is to provide value to different kinds of stakeholders during a development project. Therefore an extension to the goal template of GQM [Sol99] was added in line with work of Sommerville *et al* [Som99]. This extension is elaborated in §4.3.1, after which measurement goals are discussed in §4.3.2.

4.3.1 Viewpoints

The research of Sommerville *et al.* [Som99] is focused on finding process inconsistencies in software development organizations. Therefore an approach to process analysis is used that helps to discover different *perspectives* on the software process. The latter is used to find inconsistencies and stimulate process improvement. As part of the approach, Sommerville *et al.* introduced the notion of "process viewpoints". These viewpoints contain information about particular software process areas, as expressed by different stakeholders.

Process Viewpoint: Organization management	
Concerns:	Productivity, Repeatability (learning capability), Customer satisfaction, Costs
Focus:	- Improve cost and resource estimates for future projects - Increase productivity in order to reduce costs (and to eventually win more bids) - Promote a standard approach toward software project execution (factory idea)
Sources:	Managing Director, Business Unit Manager, Project Planner/Controller

Process Viewpoint: Project management	
Concerns:	Productivity, Readiness, Costs
Focus:	- Control project time and resources - Understand the level of project readiness/completeness - Understand fulfillment of non-functional requirements (ie. quality attributes)
Sources:	Project Manager(s), Project Planner/Controller

Process Viewpoint: Functional design	
Concerns:	Customer expectations, Repeatability
Focus:	- Understand customer requirements - Understand (subjective) correctness of requirements throughout the project
Sources:	Functional Leader

Process Viewpoint: Construction	
Concerns:	Quality, Productivity
Focus:	- Monitor the quality of software development activities - Understand the amount of time required to complete certain tasks
Sources:	(Sr.) Software Developer

Process Viewpoint: Testing	
Concerns:	Customer validation, Defect analysis
Focus:	- Improve customer involvement in test activities - Improve communication with customer w.r.t the amount of test effort - Understand artifact changes
Sources:	Test Leader

Figure 4.2: *Process viewpoints*

In the context of this measurement programme the viewpoints of Sommerville *et al* were used to capture stakeholder perspectives on software process areas *in need of measurement*. Since this slightly differs from the original approach a modification was made to the viewpoint templates (of Sommerville *et al*). For example, the process descriptions (in the form of process models) are excluded from the viewpoints, since these are to much of a burden at this stage of the research. Actually software process modeling [Cur92] was not applied at all during goal identification and environment characterization. Although the benefits of process models are well known [Cur92; Bri96], the initial scope of the programme (at the very beginning) of this research was simply to wide to effectively model all processes. Therefore it was suggested to postpone process modeling

until more information about process areas, relevant to the measurement programme, are known. Due to time constraints, however, process modeling falls outside the scope of this thesis project and is thus recommended as future work (§8.3.2). So far the use of viewpoints, as illustrated in figure 4.2, is assumed to provide enough insight in the organizational environment.

Interpretation

As can be seen from the first two viewpoints, stakeholders are mainly interested in progress and cost information when it comes to measuring productivity. However, this information serves two distinct purposes. On an organizational level (i.e. across projects) stakeholders will use this information to increase the learning capability of the organization. This includes more accurate project estimates and support for the identification of improvement opportunities. On the individual project level, productivity information will mainly be used for monitoring progress and managing project members. Thus in terms of §2.2 the organization is focused on *predict* and *improve* goals, while project management stakeholders are focused on *control* goals. However in both cases a solid *understanding* of productivity is required before addressing the control, improve and predict goals [Sol99]. As a result it is decided to concentrate the initial measurement programme on understanding productivity at the *project level*. Nevertheless, considering the goal of baselining productivity, attention will also be paid to project measures that facilitate (future) cross project comparisons. That is, measures suitable for project portfolio *benchmarking* [Max01]. This decision, to focus initially on project level measurement, is in line with principles of [Jon03c, chap. 3]. The latter also applies to defect and scope measurement.

4.3.2 Measurement goals

Based on the improvement goals listed above and the concerns captured in viewpoints, a definitive set of *measurement goals* was formulated. Questions and measures discussed in subsequent chapters are derived from *these* goals. The goals are illustrated in figure 4.3, and formatted according to the GQM template of [Sol99]. Furthermore references are made to relevant process viewpoints (fig. 4.2).

Productivity		
MG1.	*Analyze:*	software development process
	For the purpose of:	understanding project execution
	With respect to:	productivity
	From the viewpoint of:	project management
	In the context of:	enterprise Java development projects
MG2.	*Analyze:*	software development process
	For the purpose of:	baselining project execution
	With respect to:	productivity
	From the viewpoint of:	organization management
	In the context of:	enterprise Java development projects
Scope		
MG3.	*Analyze:*	software development process
	For the purpose of:	understanding project scope
	With respect to:	requirements and change requests
	From the viewpoint of:	functional design, project management
	In the context of:	enterprise Java development projects
Defects		
MG4.	*Analyze:*	software development process
	For the purpose of:	understanding
	With respect to:	defect reporting
	From the viewpoint of:	project management, construction, testing
	In the context of:	enterprise Java development projects

Figure 4.3: *GQM measurement goals*

As can be seen there are two goals concerned with productivity. This is done to differentiate between the concerns of *organization-* (baseline/future benchmarking) and *project* (understand) management stakeholders.

4.4 Concluding remarks

This chapter discussed the process of environment characterization and goal identification. The primary instrument used for acquiring knowledge, about the organization and associated business goals, was semi-structured interviews. These interviews resulted in the identification of three improvement goals and five stakeholder viewpoints. In turn this led to the definition of four (GQM) measurement goals. These goals address productivity, defects and scope management in software development projects. Although stakeholders were eager about the improvement and control aspects of measurement programmes, it was found that there is currently insufficient data available to initiate measurement for these purposes. The measurement goals therefore aim at *understanding* productivity, defects and scope.

Considering these goals, the conclusion can be drawn that the programme is focused on fairly "traditional" areas in software engineering. For example, measuring productivity and defects are already recommended practice in early measurement programmes, like [Gra94] (based on research by Grady and Caswell at HP in '87). However, this doesn't make measurement in these areas less important. Productivity measurement, for instance, is vital to almost any organization and still considered a challenging undertaking [Max01]. So to speak, there are two levels of (productivity) *measurement usage* to discern: Project (management) stakeholders aim to use measurement data *in*side a specific project. While organization management stakeholders plan on utilizing measurement data *across* projects. However, it is decided to focus initially on *project level* measurement and analysis, while *facilitating* future organizational use (by careful definition of measures).

Additionally, stakeholders confirmed that the measurement goals in this chapter apply to (nearly) *all* software development projects. As a result this research targets the establishment of a *strategic* measurement programme (§2.3.1).

Chapter 5

Information needs and Constraints

THE previous chapter described the focus of the measurement programme through the process of viewpoint discovery and measurement goal identification. This chapter elaborates the definition of questions in response to these goals.

However, it is important to note that formulating questions (chapter 5) and defining measures (chapter 6) are not completely distinct activities. There is a strong relation between both tasks, since it was noticed that stakeholders have very divergent ideas about information related to specific goals. For example, depending on the exact area of interest, stakeholders sometimes express rather abstract and unstructured *information needs*. On the other hand stakeholder are also temped to express concrete and detail *measures*, while reasoning about their needs. Therefore questions and measures were, for a large part, formulated in parallel. Nevertheless, for the sake of readability, the two activities are discussed in separate chapters and references are made where applicable.

5.1 Terminology

Before discussing the questions it is important to establish a common terminology. For instance there are quite a number of different terms used in literature to express a measure of effort. To avoid confusion between words like; budgeted, actual, estimated, expected, remaining and spent effort the meaning of these terms are explained in figure 5.1. Note that the same terminology is also applicable in the context of project "duration"[1].

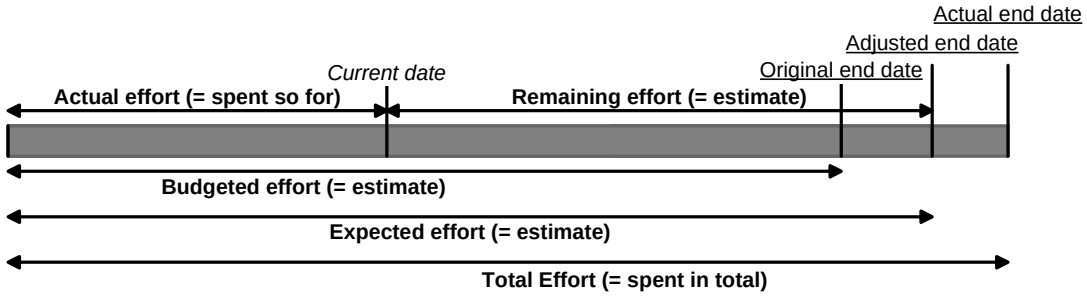


Figure 5.1: *Effort terminology*

Apart from the terminology used for effort and duration measurement, there is often confusion about the definition of a "defect". To avoid such confusion in this thesis, the terminology of [Dia97] is roughly followed. That is "a problem detected in the same phase it was introduced is defined as an *error*, a problem that escapes detection in the phase it was introduced is defined as a *defect*". Figure 5.2 illustrates this terminology.

[1] In this thesis the term *duration* is equal to *cycle time*

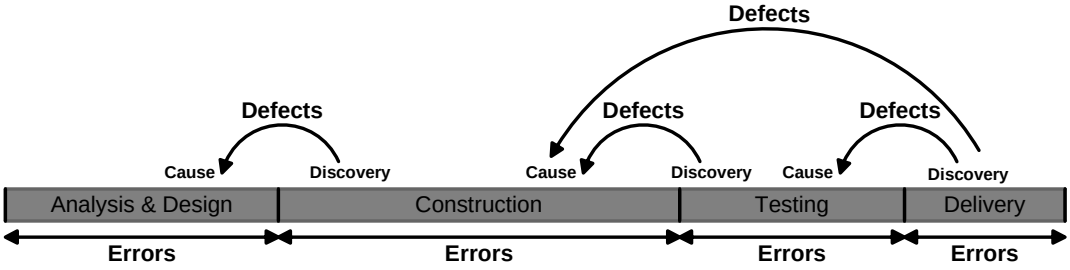


Figure 5.2: *Defect terminology*

5.2 Strategy

In addition to establishing a common terminology, a "strategy" for the definition of questions (and associated measures) is required. Specifically because there are a number of constraints attached to this measurement programme. For instance, in order to ensure its effectiveness and attainability, the measurement programme should start small and follow and incremental approach (§2.4).

Goal prioritization

The first step undertaken in this direction is the prioritization of measurement goals. In concordance with stakeholders priorities were assigned as followed[2]: First comes productivity or more specific the understanding of productivity on the project level (MG1). Understanding/baselining productivity on an organizational level is put second (MG2). This goal is directly followed by the understanding of defect data (MG3). Finally, understanding of requirements scope (MG4) is assigned the lowest priority. Scoping is therefore excluded from *initial* programme implementation and discussed in appendix B. Moreover, the process of questions and measures prioritization is discussed in §6.3.

Measurement simplicity

In addition to goal prioritization, a constraint was put on the number of questions associated with each goal. A "fixed limit" is however undesired. Therefore it was decided to include only questions that relate to important entities of a specific goal. For example, a goal concerned with "productivity" mainly includes questions related to schedule and effort (of entity *planning*). Questions related to other entities (such as tooling and project methodology) are not included. Since these are, by belief of stakeholders and the present author, less prominent when reasoning about a factor such as "productivity". This approach is in line with work of Park *et al.* [Par96], who are using "Entity-Question Lists" during their goal definition process. Although this specific method was not applied during the research, the principle of Park *et al*, (i.e. framing questions in response to entities) was roughly followed.

Limiting the number of questions also affects the number of measures, because one should derive appropriate measures (§2.4). So to speak, *more* measures are not necessarily better since it results in extra data collection, analysis and interpretation work. Therefore only measures that explicitly satisfy stakeholder questions are included [Bas95]. Furthermore in order to provide a high level of information value, it was decided to combine measures where appropriate. For example; a direct measure such as the budgeted duration of a certain iteration, doesn't tell much about whether or not the project is on schedule. Only in relation to the actual duration (past calender time) and the expected duration (estimated calender time to completion), one can decide if the project is running as anticipated. Nevertheless measures should always follow the KISS principle[3]. That is, if simple measures suffice to satisfy certain questions then there is no need to define more complex measures.

[2]Note that the goals in paragraph 4.3 are already placed in order of their priority
[3]Acronym for *Keep It Simple, Stupid* (SOURCE: FOLDOC)

Process changes

Another constraint to this measurement programme, is that the development process (in projects) should not change solely for the sake of measurement implementation. That is, measures should address the software process currently in place. Prematurely changing a specific part of the software process (e.g. unit testing activities) in order to facilitate measurement (e.g. life span of defects) is considered bad practice. Of course measurement will require additional effort and (some) changes in the process, but this should be mainly supplemental to the existing work. This constraint is put forward because it is believed that premature changes to the development process will complicate (initial) measurement introduction. After all, it requires changes on two fronts; the current *work process* and the *measurement process*. This constraint only applies to initial programme implementation. Once measurement data is available (that is, *understanding* is reached §2.2) the software process can be changed accordingly.

5.3 Information needs

Based on the information from the interviews, the concerns captured in viewpoints (§4.3) and the strategy described above, a set of questions for each of the goal types was devised. The purpose was to formulate *questions* that reflect the information needs of various stakeholders, with respect to the measurement goals stated. As opposed to the information in viewpoints these questions are a *consensus* between the specific information needs of the individual stakeholders. However, formulating these questions isn't straightforward [Sol99]. It took quite a number of iterations before the questions were at the "right" level of abstraction. For example a question like "what is the estimated versus actual calender time for this iteration?" is too detailed because it already defines the measure, instead of the information required. Therefore this question cannot be used to assess whether or not a specific goal is (being) attained. A question such as "is the project on schedule?" does (even though it is somewhat general) a better job in representing a stakeholder information need. The definitive set of questions are discussed in the remaining sections of this paragraph. Associating measures are presented in chapter 6.

5.3.1 Productivity questions

From an economical perspective, productivity is often defined as "the amount of output produced per unit of input". In software development input might be the number of man-hours and output might be a certain amount of code [Dia97]. Although this sounds a bit simplistic, productivity is an aspect that is difficult to measure. Primarily because the level of productivity in a software development project is determined by many factors. For example; personnel experience, project size, tooling, methodology, reuse, application domain and technical complexity all affect the productivity within a development organization. An extensive survey of more factors influencing productivity is presented in [Sca95].

Considerations

As previously stated (in chapter 4), stakeholders are focused on cost and progress information when it comes to productivity. The questions expressed by stakeholders are discussed below.

Duration

Considering costs and progress, the extend to which software projects adhere to *schedule* was one of the first information needs expressed by stakeholders during interviews (Q1). This refers to overall project *duration* (§5.1) and the duration of individual phases. This information is especially relevant to fixed-date projects, where schedule overruns may result in decreased customer satisfaction. However, other types of projects also benefit from this information, since a proper understanding of duration may aid in more accurate task planning. The latter is also interesting on the organizational level for making better duration estimates on future projects. Thus, in short, stakeholders are interested in the distribution of time in the project as well as the (in)accuracy of duration estimates. Information on matters such as the influence of new techniques on cycle time, or the origin of schedule overruns is currently not a top priority.

Effort

Apart from schedule, stakeholders are interesting in knowing whether or not the project is within *budget* limits (Q2). The term budget refers in this instance to two separate attributes, namely: costs (euro's) and effort (man-hours). Although cost management is an important factor in commercial environments (as indicate in chapter 4), stakeholder questions did not target this specific attribute. This is mainly due to the (active) financial monitoring of Daniro's Project Office (§4.1.1). However, this doesn't mean stakeholders don't care about costs. Effort (in man-hours) is used for tracking both *costs* and *progress* (i.e. how far are we? how much work is there left?). Costs refers in this instance to the *budgeted* amount of effort assigned to the project in relation to the *actual* amount of effort spent on the project. With help of this information project managers can track whether they are within (labor) cost limits. Furthermore by tracking the *expected* amount of effort (§5.1), project managers can look ahead and decide if the budget suffice for the time remaining. The latter information can also be used to track project progress. Since it gives an indication about the amount of tasks completed and the amount of work left to do.

Size

Even though progress can be measured by monitoring effort expenditures, the amount of effort says little about the functionality implemented. Tasks can be completed on schedule (time) and withing budget (effort) but still result in little functionality being realized. Considering the fact that customers pay for functionality, it is important to keep track of *functional size* over time (Q3). The problem with size, however, is that there is no single measurement method available (§2.2.2). At Daniro, Function Points and Use Case Points are often used for estimating software projects. However, this isn't *always* the case, projects are also measured using other (simple) counting methods (# screens, # use cases, ...) or by expert judgment. These different sizing methods make it harder to compare progress (i.e. the functionality completed) across projects. Nevertheless, this isn't currently a top issue since the programme is mainly focused on project-level measurement (§4.4). Therefore it is decided to maintain the current situation and determine the exact measurement method individually, per project. A second difficulty in tracking functionality over time is the demand for *periodic* data collection. When simple counting methods are used periodic size measurement is easy. However, in projects depending on function point analysis, periodic measurement is costly and cumbersome because an educated (i.e. certified) function point analyst is required. An alternative might be the use of "backfiring" [Jon95] to convert Lines of Code in Function Points. Unfortunately the backfiring technique only gives a rough indication at best. Despite these issues, stakeholders were positive about periodic size measurement. Therefore, question (Q3) concerning this information need is included in the programme. Nevertheless, the *granularity* (§6.2.1) of the measure(s) formulated in response to this question must be considered carefully.

Overall performance

Until now three aspects of productivity have been considered, respectively: duration (schedule adherence/calender time), effort (cost/progress) and size (progress). These aspects reflect stakeholder information needs with respect to the productivity in a project, and need to be included in the programme. However, the dependencies between these aspect haven't been considered. Therefore it is suggested to include one or more measures that focus on the *relation* between duration, effort and size. This in order to determine the *performance* of the project as a whole (Q5). The latter is particularly useful to organization management as indicated in §4.4.

Staffing

The staffing on the project (Q4) also relates to this question, since it is assumed to have a major influence on productivity. Especially at Daniro where the staffing on a project is heavily subjective to change, because people get assigned to other projects or companies (consultancy). Therefore monitoring how the project team is assembled over time, assists managers in making well-considered decisions with respect to the amount of people working on a project. Since one logic response to low productivity (assumed to be identified by the measures described above), is to increase the number of people participating. However, Brooks' Law [Bro95] shows this is most often not the right decision to make. Brooks states that "adding people to a late software project only makes it

later" because intercommunication increases and it takes people time to learn a new project. Therefore, considering the goal of baselining productivity, it is decided to include staffing measures in order to understand when an "optimal" number of people for a particular project is reached.

Decisions

With respect to the information needs discussed, the following set of questions was proposed:

Q1. Is the project on schedule?
Q2. Is the project within budget?
Q3. How much of the intended functionality is implemented?
Q4. How is the project team assembled?
Q5. What's our overall performance?

5.3.2 Defect questions

Another wish of stakeholders was to gain more insight in the "quality" of software development projects (B2). Quality however, is an even more wider and ambiguous term than "productivity". Therefore decisions had to be made with respect to quality areas in need of measurement. Furthermore measurement of software quality is usually aimed at the *product* level (with measures of source code complexity, coupling, ...). Since these type of measures are outside the scope of this research, effort is put in investigating measures of process quality. Hereby the notion of *defects*, and the development of defects over time, is of particular importance. Although software quality involves more then defects alone, *tracking defects* is considered a (cost) effective way of monitoring process quality throughout a software project [Jon96].

Considerations

During interviews it became clear that defect-related information is currently not actively used by stakeholders on levels of project or organization management. Nevertheless, most managerial stakeholders expressed their interest in this kind of information. The stakeholder information needs concerning defects are, however, diverse. This is understandable since defect data can be used for a wide range of purposes. For example, defects are known to be used for measuring [Kan04]; overall product quality, effectiveness and thoroughness of testing, product reliability, readiness for release, customer satisfaction and project status.

However, it isn't possible to address every single information need concerning defect detection. Therefore decision have to be made with respect to the most frequently arising issues (discussed above) while leaving out "nice to know" issues.

Quality perspective

Stakeholders indicated that *thoroughness of testing* and *readiness for release* are particularly important. Understanding the thoroughness of testing relates to both system testing (e.g. performed by a test team) as well as customer testing (e.g. acceptance and functional testing). Stakeholders indicated that the latter is notorious for being overlooked. That is, customers do not always sufficiently test a certain release. This results in a small number of initial defects that suddenly increase near the end of the project, when defects are more costly to fix. In addition, the readiness for release is important to measure since it affects project planning. Which in turn influences schedule performance (§5.3.1), thus *productivity* (§5.3.2). Furthermore, readiness for release is vital from a financial point of view, since Daniro provides a *guarantee period* to customers. This is a fixed period of time after project delivery, during which defects are repaired free of charge. Only defects that by reasonable assumption couldn't have been identified during acceptance tests are covered by this guarantee period. Therefore, it is key to fix defect early on in the development process.

As one may already notice the two purposes *thoroughness of testing* and *readiness for release* (as well as other defect *measurement purposes*) are very much intertwined. So to speak, both depend on a measure of defect count. Thus a decrease in defect count could indicate the readiness for release (since the product is becoming stable) but it could just as well indicate a fall in the thoroughness of testing. In short, the purposes aren't mutually exclusive.

Productivity perspective

Even though defects are often used for quality purposes, understanding quality development throughout the project as discussed above is not the *only* concern of stakeholders at Daniro. Productivity appeared to be an important driver behind defect measurement. After all, effort invested in defects is *rework* that (in an ideal situation) could have been discovered earlier or avoided altogether [Boe01].

Decisions

To summarize, defect information serves both a productivity (B1) as well as quality (B2) goal. Therefore, in the case of Daniro, it is decided to focus measurement on two objectives: 1) tracking effort invested in rework and 2) tracking defect count. These two objectives are largely intertwined. However, the first objective is directly related to understanding *effort expenditures* over time (Q8). The *time* at which defect are found (Q7) and the *origin* of defects (Q9) are also useful in this matter. This allows stakeholder to analyze the possible causes of defect (and thus effort) increases, and helps in making decisions with respect to risky phases that require extra attention. The same holds true for the second objective of tracking *defect count* (Q6). Which is required for understanding release readiness and thoroughness of testing. The *impact* on the project (Q10), in terms of defect severity and priority, is vital for the correct interpretation of defect count. The last question (Q11) is concerned with most of the aspects mentioned above, to facilitate organizational measurement. Altogether, the following questions are formulated:

Q6. How many defects are found in this project?
Q7. When are defects being found?
Q8. What effort is required to fix defects?
Q9. What causes the defects in this project?
Q10. What is the impact of defects on the project?
Q11. How effective is the defect solving process?

5.4 Concluding remarks

Based on the questions discussed, the conclusion is drawn that productivity is *the* most important driver behind stakeholder information needs. The productivity goal (defined in chapter 4) was not only assigned the highest priority, it also appeared to be a major driver behind information needs related to *defects* (and *scope*, see appendix B). However it should be noted that only a small number of questions directly relate to productivity, in the economical sense of size/effort. Factors *depending* on productivity (schedule adherence, costs) and factors *influencing* productivity (readiness for release, staffing) account for the majority of productivity-related information needs.

Overall, one may conclude that the information needs discussed in this chapter, mainly reflect the *interests of managers*. The reason behind this observation is that managerial stakeholders are primarily focused on *process* measurement. Since they are responsible for guiding a project (through the software process), in which effort, duration and readiness for release are important matters (§2.1.2). In contrast functional-, software- and test engineers are more concentrated on (detailed) *product* measurement. Because they are responsible for parts of the software system under development. This doesn't mean, however, that the research failed concerning the objective of satisfying *multiple* stakeholder needs. For instance, the current programme directly addresses the information needs of at least three different stakeholders (i.e. management-related roles). Additionally, it is concluded that engineers *also* benefit from the current programme, since they are more likely to work on projects where goals are clearly defined and the software (measurement) process is established [Min00] (assuming the programme is well implemented). Moreover, the measures specified in chapter 6 (which are defined in response to *questions* listed in this chapter) reflect to a large extend the measures used *in* the Personal Software Process (PSP) [Hum96]. Which is a method for improving project planning and quality assurance for individual *engineers* by collecting and analyzing software project data[4]. A prerequisite for satisfying multiple information needs, however, is that measurement data is available to *all* project members (§2.4).

[4]The PSP is related to the Team Software Process (TSP), a method for organizing PSP engineers in cooperative teams.

Chapter 6

Measurement Definition

THE previous chapter described the information needs of the stakeholders in the measurement programme. This section elaborates the definition of measures with respect to these information needs.

6.1 Granularity

As can be seen from the list in §6.2.1, measures are defined at different levels of *granularity*. This is done in order to satisfy the information needs of the stakeholders involved in the measurement programme. However, during the process of measurement definition it was found that one must caution not to specify *too* many levels of granularity. Since each level of detail requires extra effort with respect to data collection, analysis and interpretation work. Currently there are five levels of granularity (illustrated in 6.1). The highest level is "organization". Measures specified at this level can be used for creating an organizational baseline and measuring project portfolio[1]. From there improvement initiatives can be initiated. Eventually these measures can also be used for *benchmarking* purposes between projects, development centers and possibly other companies. Nevertheless as indicated before (§4.3.1), the current programme is not focused on this level. As a result organization measurement is suggested as future work (§8.3.1). However, in order to *facilitate* this work, explicit attention is paid to lower level measures (e.g. M1, M13, M24) that provide value to organizational level measurement.

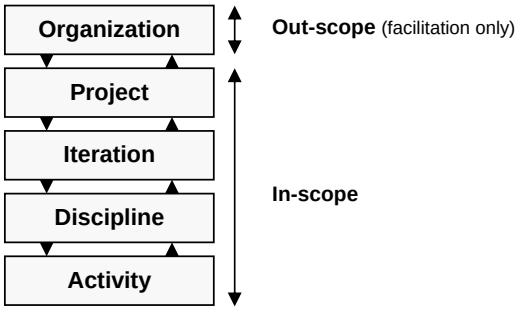


Figure 6.1: *Levels of granularity*

The *project* level, situated below the organization level, is decomposed in one or more *iterations*. The term iteration is used in this instance since it is an essential part of RUP, the *de facto* system development methodology at Daniro J-Technologies (§1.1). Subsequently, iterations consist of *activities* that belong to certain *disciplines*. For example an activity such as "test user login'" belongs to discipline "system testing". The activity level is the lowest level defined in this measurement programme. Measures at this level can be *aggregated* (i.e. 'lifted') to discipline, iteration and project level. Measures on other levels, e.g. artifact, system or component, are not included in

[1] The whole of past and present projects, performed by Daniro (J-Technologies)

this programme because stakeholder questions did not target these entities. Furthermore, it is suggested to start on the activity level and expand to lower levels when necessary. In addition, measurement on the activity level requires (depending on the exact measure in question) little changes to the work process of stakeholders. Since a decomposition of work in activities is readily available in projects: Often a so-called "Work Breakdown Structure" (WBS) is created by managers and team leaders. This is a hierarchical tree structure of tasks that need to be performed in order to complete a certain project. The bottom of the hierarchy contains *activities* that cannot be broken down any further. Usually a WBS is created for planning purposes (resource allocation, assignment of responsibilities), but it also facilitates measurement.

Staff level

Although one might expect to see measures per *staff member* (e.g. programmer, tester, manager), this level of detail is explicitly not included in the programme because measuring individual members is considered a risky undertaking [Wie97]. Chances are that managers will use (or "exploit") this information to *judge*, instead of *motivate*, staff members. Considering that measurement programmes involve the cooperation of multiple people, misuse of measurement data may results in decreased participant buy-in as well as decreased data integrity. As a result the success rate of the measurement programme is reduced (§2.4). In order to avoid this, measurement should not concentrate on (individual) judgment or reward. Instead it should be used as a basis for providing *feedback* to participants (§2.3.1).

Time level

Since this measurement programme is concerned with *process* measures, stakeholders are often interested in the development of a certain attribute *over time* (i.e. throughout the process). The granularity levels in figure 6.1 already imply a form of time ordering. For instance, measures on the *iteration* level (which are sequential in time) can be *aggregated* to provide a measure of the development of a certain attribute throughout the *project*. However, there's a difference between measures that yield results on a certain level of granularity and the time at which data collection for a particular measure takes place. That is, collection and reporting moments are not necessarily related. So even though the activity level is the lowest level possible, it doesn't mean that measures on that level are only collected once for each activity. When stakeholders require information about the development of a certain activity over time, data collection can take place at arbitrary moments. To conclude, the level of granularity for a certain measure is (in the first place) determined by the information needs of stakeholders. This in turn determines the necessary aggregation of data.

6.2 Measures

This paragraph highlights the measures defined in relation to the productivity and defects goals. The rationale behind important (and non-obvious) measures is explained. Furthermore the alternatives and possible drawbacks associated with those measures are discussed. Measures marked with an asterisk (*) are given high priority and included in the initial programme (§6.3).

6.2.1 Productivity measures

The information needs concerning productivity of the stakeholders at Daniro mainly evolve around cost and progress. By taking stakeholder questions into account, the present author proposed a set of measures related to these attributes. The measures are listed and discussed in this paragraph.

Overview

Q1. Is the project on schedule?
- **M1**. Estimation inaccuracy
- **M2**. Time till budgeted end date
- **M3**. Budgeted iteration duration vs. actual and remaining iteration duration*

Q2. Is the project within budget?
- **M4**. Cumulative budgeted effort planned vs. expected effort spent*

- **M5**. Budgeted effort planned vs. actual- and remaining effort spent *per iteration**
- **M6**. Budgeted effort planned vs. actual- and remaining effort spent *per discipline**
- **M7**. Product size divided by the amount of effort *planned* vs. Product size divided by the amount of effort *spent**

Q3. How much of the intended functionality is implemented?
- **M8**. Estimated vs. actual product size*
- **M9**. Percentage of budgeted product size implemented vs. Expert effort estimate*
- **M10**. Percentage of specification implemented

Q4. How is the project team assembled?
- **M11**. Number of participants
- **M12**. Participant skill level (seniority)

Q5. What's our overall performance?
- **M13**. Actual duration vs. Actual effort spent in relation to the Actual product size per iteration*
- **M14**. Process Productivity

Duration

The inaccuracy of duration estimates (M1) is the first measure defined in response to the question concerning schedule adherence (Q1). This measure is defined for use by organization management to facilitate *future* cross-project comparisons (§6.1). The estimation inaccuracy is computed once, after project completion: by extracting the budgeted duration from the actual duration, divided by the actual duration. This results in the relative (%) error of duration estimates and shows whether the project was over- or underestimated. This information is useful in creating an organizational baseline and for possible improvement of project planning and customer bids.

The duration or cycle time of the project is measured on the *iteration* level. Since iterations are timeboxed (fixed in length) the budgeted duration of an iteration should not be exceeded. However, in practice this is not uncommon. Therefore measurement on this level of granularity is suggested. Measurement on lower levels is not (directly) included in the programme because stakeholders are already aware of schedule overruns on these levels. This is mainly due to the fact that stakeholders, particularly project managers, work with project schedules on an *activity* level. Usually in the form of Gantt charts (based on Work Breakdown Structures). As a result the initial focus lies on measuring the duration of iterations. However, in order to compute this measure, information from the activity level needs to be aggregated. Naturally, this poses a demand on the correct registration of activity durations.

Effort

As stated in chapter 5 stakeholder questions didn't directly addresses the "money" aspect in software projects. As a result (advanced) cost-related measures, based on method such as Earned-Value Analysis, are not included in the programme. However, a number of effort-related measures that address cost aspects are included. Most notably measure M4. This measure sets out the (cumulative) development of the budgeted effort over time, against the expected (actual + remaining) effort over time. The data needed for this measure is aggregated from the activity level. The latter allows stakeholders to analyze whether or not the project is within budget limits. However, it doesn't show the cause of a (sudden) effort fluctuation. Therefore two measures on lower levels of granularity were proposed. The data for these measures is also collected on the activity level and aggregated to respectively the *discipline* and the *iteration* level. On the iteration level the budgeted effort is outlined against the actual and the remaining effort (together the expected effort). The same holds true for the discipline level, together these two measures allow stakeholders to monitor their budget within a project. Furthermore, it helps to determine what kind of activities (e.g. architecture, requirement engineering, configuration management, ...) require addition budget. For additional explanation; prototype charts of effort measures are illustrated in appendix D.

Although tracking effort might look simple at first sight, the reality is different. One problem that posed itself during this project was the (conflicting) purpose of effort data and terminology used (§5.1). This terminology mismatch wasn't solely limited to literature. Stakeholders on various positions used different terms, based on their specific information needs. For instance, high level

management is primarily interested in effort budgeted vs. actuals, whereas the Project Office (controllers) are more interested in the remaining effort. Project managers on the other hand, look specifically at the expected effort to see whether they are within budget limits.

Size

Finally, with respect to question Q3, a measure of the estimated and actual size of the product is proposed. Furthermore, for projects that use (simple) counting methods to quantify size, a measure is proposed that represent the percentage of "specification(s)" implemented. Clearly this is a subjective measure, since it is hard to tell whether or not a specification (i.e. use case) is implemented either half, full not at all. Nevertheless, as opposed to Function Point Analysis, this measure provides stakeholders with the most *recent* size information. After all it includes artifacts that are still very much in development. Stakeholders expressed their interest in this information, because it allows them to make better schedule and effort estimates. However, the most interesting size-related measure is M9. This measure sets out the development in size (presumably in FP) against the *actual* effort divided by the *expected* effort. Although size and effort are two distinct variables in project management, the *relative* development of these factors over time should follow a somewhat equal path. That is, if one assumes that the "completeness" of the project (% of budgeted effort used) follows the same path as the "readiness" of the product (% FP implemented). The calucation of project "completeness", or *percentage complete*, is frequently used among Daniro project managers as a means of expert judgement. In this light, measure M9 is considerd to represent the relation between two estimation techniques: function point analysis and expert judgement. However, this measures will have to prove itself over time, since there are a number of issues associated with its use. For instance, the relative amount of FP implemented is based on the *originally* budgeted FP (i.e. at project inception) while the other factor, actual effort/expected effort, is based on a regulary updated effort budget. To conclude, this measure is included since the interests of stakeholders is strong. However, additional (fine-tuning) work is expected during implementation.

Overall performance

Measure thirteen; *process productivity* [Put02] requires addition explanation. This measure is said to cover productivity during the entire software life cycle. Therefore process productivity is build on three other measures; *size*, *effort* and *duration*, as discussed in the previous paragraph. The latter variable, duration or time, is remarkable since productivity is conventionally defined as size divided by effort [Fen00; Max01]. However, Putnam *et al.* observed that productivity in a software project is very much dependent on the schedule planned at the start of the project. Therefore time is included as a variable. Nevertheless size, effort and time do no equally affect productivity. During an empirical study of measurement data from a large number of software projects Putnam found that there is a non-linear relation between the three measures. As a result two exponents were added to balance process productivity. As a result, process productivity is calculated as follows:

$$Process\ productivity = \frac{Size}{((Effort \times 12)/Skills)^{1/3} \times (Time \times 12)^{4/3}}$$

Where size is either lines of code (LOC) or function points (§2.2.2), effort and time is measured in months and skills is a complexity factor. The skills factor dependents on the size of the product and varies between 0.16 to 0.39. The number is used to adjust the process productivity value because the need for skills such as testing, QA and management is expected to grow (significantly) on larger projects [McC04]. The process productivity measure described above is usually not calculated by hand, since it is part of the SLIM cost estimation model and supported by tools of QSM Inc[2]. These tools facilitate project control, estimation and benchmarking. As a result process productivity values are often converted into a position on the "productivity index" scale. One of the QSM tools, SLIM Estimate, is currently in trial use at Daniro's Project Office (§4.1).

Process productivity can be a useful measure in software development projects, but one should be careful not to put too much weight on it. Solely using a number of process productivity to steer

[2] `http://www.qsm.com`

a project is a bad idea. Process productivity should, like any other measure, *not* be considered in isolation. One objection is that process productivity depends on a measure of size (often LOC), and because of the many issues associated with functional size measurement (§2.2.2) this could greatly influence the outcome. Furthermore, the measure is particularly suited for long running projects (lasting a least a couple of months). The reason for including this measure anyhow is to *facilitate* productivity comparisons across projects (§4.4). It is expected that process productivity suits this purpose well, since it produces a single numerical value that captures multiple productivity aspects. However, process productivity doesn't show the (possible) cause of a certain productivity increase/decrease. Therefore a simple measure (M13) that shows a (possible) relationship between iteration duration, size and effort is proposed to fill this gap.

6.2.2 Defect measures

Stakeholders are interested in tracking defects for both economical and quality reasons. In this paragraph, measures related to defect tracking are proposed in response to stakeholder questions.

Overview

Q6. How many defects are found in this project?
- **M15**. Number of defects found and fixed per iteration*
- **M16**. Defect density

Q7. When are defects being found?
- **M17**. Defect detection time by discipline*
- **M18**. Defects by type (internal/external)

Q8. What effort is required to fix defects?
- **M19**. Budgeted vs. actual effort spent on defect fixes per iteration*

Q9. What causes the defects in this project?
- **M20**. Defects by origin

Q10. What is the impact of defects on the project?
- **M21**. Defects by severity*
- **M22**. Defects by priority*

Q11. How effective is the defect solving process?
- **M23**. Effort spent on re-opened defects (rework on fixes)
- **M24**. Defect indices

Classification

The previous chapter showed that stakeholders are interested in defects from two perspectives (both quality and productivity, or efficiency). This poses a demand on defect measures since different types of information needs will have to be satisfied. Obviously the more information one collects about defects, the more (interesting) measures one can define. However the more information one collects about defects, the more weight one puts on the people registering defects. Therefore balance needs to be found between the measurement information required and the amount of defect data collected. Consulting literature on this subject learned that there are several ways of classifying defects during software development projects [Fre01]. A well-known defect classification scheme, which is especially interesting from a measurement perspective, is *Orthogonal Defect Classification* (ODC) [Chi92]. This scheme, developed by Chillarege *et al.* at IBM, is focused on finding the cause-effect relation of software defects on the development process. That is, defects categorized by the *type* of fix can be related to activities in the development process. Thus as Chillarege *et al.* state "ODC essentially means that we categorize defects into classes that collectively point to the part of the process that needs attention". To derive this information the ODC scheme consists of eight attributes; activity, trigger, impact, target, source, age, type and qualifier. These attributes can be used alongside the more conventional attributes such as *severity* and *priority* to capture both defect cause and effect. In ODC the defect *trigger* and defect *type* attribute are of particular importance. The purpose of the type attribute is to define the semantics of the defect fix. For example when a developer solves a defect in the source code by properly initializing a global variable, the fix is said to be of type "assignment". Originally IBM defined a limited set

of eight different defect types (among others; interface, documentation and algorithm). These types were formulated to capture "orthogonal" information. Meaning that types are specified at a level that is generic enough to apply throughout the development process, but specific enough to be associated with a few number of phases. Thus assigning a defect type tells something about the possible phase where the defect was inserted. Chillarege *et al.* state that, with help of this information it is possible to determine how the product moves through the development process. Another attribute that assists in this process is the *defect trigger*. This attribute is used to record the condition or environment in which the defect (of a specific type) occurred. Examples are: concurrency situations, specification errors and so on.

Conclusively it can be said that the ODC scheme is a powerful technique to extract measurement information from defect data. Although it can be used to produce a variety of measures it is mainly targeted at the cause or *origin* of defects. Nevertheless powerful as ODC may be, the present author opted not to include measures related to defect types and triggers (or other attributes of the scheme) in the programme. There are a number of reasons for this decision. First, including ODC related measures in the programme would require (significant) changes in the work process and tooling of developers, with respect to defect reporting and analysis. Also the ODC scheme needs tailoring before it can be used in Daniro development projects. The original defect types defined by IBM are mostly outdated and not applicable anymore within the context of (web)application development. Although the scheme has been updated over time to include more modern information, it is still necessary to reach agreement about certain issues within a specific project context. Since process measurement is only in its initial stage at Daniro, the decision was taken not to make these kind of (premature) changes to the development process prior to measurement implementation (§5.2). A second reason for omitting including ODC measures is that quality improvement isn't the only motivation behind defect collection. As indicated in chapter 5 schedule and progress information, or *productivity* in general, is an important driver behind defect information needs. This is mostly because the time and effort put in solving defects is classified as work that, in an ideal situation, could have been prevented. Even though mistakes are human, it is in the stakeholders' interest to understand the development of defects over time in order to control both process quality and productivity. From a productivity point of view ODC is not the scheme of choice, since it doesn't specifically address productivity. Although some attributes are useful in this matter, the costs (process changes, tailoring) aren't expected to line up with the benefits (semantic classification, origin capture).

Productivity and impact

Thus even though ODC is superior in defect classification when viewed from a purely technical perspective, the scheme is not adopted due to environmental factors. The initial list of defect measures proposed in this measurement programme (as presented above) contains a total of five defect attributes. These attributes include; severity, priority, origin, effort and discipline (in which the defect was found). Despite the fact that ODC was not adopted, a measure of defect *origin* is included in the measurement programme. As stated before the rationale behind this is primarily economical; stakeholders wish to be aware of the risky phases in a development project. Information about phases responsible for defect insertion allow (managerial) stakeholders to take preventive actions (i.e. adjust schedule and effort planning). In small projects it is expected that information such as the origin of defects can be determined informally by individual developers. However in larger projects, stakeholders may choose to adopt a more formal process such as Defect Causal Analysis [Car98]. Nevertheless, finding defect origin is often labor intensive and therefore excluded from initial programme implementation (§6.3).

The last measure in need of explanation is *defect indices* (M24). This process measure is defined by IEEE Std. 982 [Dob88] as "an indication of whether the defect removal process employed at a given phase of development was employed in a sufficiently effective manner". In addition, the measure can also be used as a product measure by providing a "continuing, relative index of how correct the software is as it proceeds through the development cycle". However, in terms of this research the *process* purpose (i.e. defect removal effectiveness) is of particular interest. Essentially the defect index is calculated as the weighted sum of all minor, major and blocking defects per iteration. Divided by the product size. More specific, the number of defects associated with each

severity level is weighted after being divided by the total number of defects found. This process yields the so-called *phase index*, because it is applied on all defects found within a specific phase (or in this case iteration). Thereafter, in order to calculate the overall defect index, every phase index is assigned an incremental weight (one for the first iteration, two for the second and so-forth). Finally the weighted phase indices are divided by the product size, producing a value of *defect index*.

One reasons for including the defect index measure in this programme is to facilitate cross-project comparisons, like process productivity (§6.2.1). This can be accomplished quite easily, since the measure produces a value that captures the relation between multiple defect properties. A word of warning however, is that the experience base of this measure is relatively small (as indicated by literature research of the present author) [Dob88]. As a result it is difficult to determine whether a certain value on the defect index is either "good" or "bad". Therefore it is key to the success of this measure to establish a (organizational) baseline. Once the defect index of a number of projects is known it becomes possible to (more accurately) analyze the defect removal effectiveness of certain projects.

6.3 Prioritization

A fairly large number of measures have been presented in the above paragraphs. These measures cannot be implemented all at once, because the measurement programme should *start small* (§5.2). Therefore, decisions needed to be made with respect to vital questions in need of measurement.

In order to do so, the measurement programme was presented to stakeholders in an interactive presentation session. During this session stakeholders were encouraged to comment on the proposed (goals), questions and measures. Furthermore rough priorities were assigned to the questions and measures presented. Afterward, as a means of validation, stakeholders were sent a survey and asked to assign detailed priorities to measures. The survey also included a column with *hypothetical* priorities, reflecting the rough priorities assigned during the presentation. Eventually this process led to a number of changes in both questions and measures, which have been incorporated in the programme. In total 13 measures have been selected for use in the *initial* measurement programme. These measures are marked with an asterisk (*) in the paragraphs above. Obviously as the programme progresses, more measures can be selected (from this document) and implemented in the organization. For the initial start however, a basic set of 13 measures is suggested as listed in appendix F.

As can be seen, most measures included in the initial programme are concerned with productivity. Furthermore a number of defect-related measures are included. Based on goal prioritization conducted earlier on (§5.2), the decision was taken *not* to include scope measures in the initial programme. Therefore scope related measures are discussed separately in appendix B.

6.4 Concluding remarks

In this chapter, productivity and defect measures were defined in response to stakeholder information needs. Hereby, productivity measures relate to *duration*, *effort* and *size*, whereas defect measures focus on understanding *impact* and *effort*. As agreed most measures focus locally on *project* measurement, at either one of four granularity levels. In addition, two measures for organization-level measurement were proposed. These measures (M14 & M24), combine the above productivity- and defect aspects in a single measure for (future) benchmarking purposes.

The previous chapters (4 and 5) concluded that the programme addresses the needs of *multiple stakeholders* (although primarily management-related roles), and is focused on *strategic*-level measurement. That is, the programme is applicable to multiple projects (§2.3.1). However, during the process of measurement definition it was noticed that it is in some cases difficult to mandate a standard set of software measures. Specifically because projects are "unique" in nature. That is, there are *variations* in techniques, methods and teams within each software development project to discern [Jon03a]. These variations are manifested through different ways of size measurement (FP, UCP, ...), varying ways of defect registration, different methods for requirements management, and most importantly different customer demands.

The precise impact and extend of variation is unknown[3]. The assumed rationale behind the observed variation, however, is that Daniro is a service-oriented organization. This makes measurement somewhat harder since *outsourcing* projects are in certain ways *customer imitated* [Rif01]. In other words Daniro has to comply to the demands of customers concerning technical and process characteristics [Jon03a]. This is different from most measurement programmes discussed in literature. Which are established in formal organizations, concerned with *product* development. Examples are NASA/SEL, US DoD, IBM, HP, Motorola, Nokia and Schlumberger RPS.

Nevertheless, since the current programme focuses on *strategic* measurement, the case is made that a standard set of software measures [Kil01] can still be established. Especially since the organization is increasingly focused on *operation excellence* [Rif01]: fixed-price projects with a limited but well-practiced set of tooling, techniques and people. The *software factory* and *development line* initiative, discussed in chapter 1, are a good example of this movement.

[3] As stated before, process modeling wasn't applied during this thesis project. Nevertheless it is expected to be useful in the future, for gaining insight into software process variations (§8.3.2)

Chapter 7

Implementation Aspects

As indicated before, *programme implementation* is outside the scope of this research. Nevertheless in preparation of implementation and in order to ensure an effective programme, some implementation-related aspects need to be considered. This includes; measurement analysis and interpretation, as well as the operational definition of measures and data collection procedures.

7.1 Measurement Support System

In the previous chapter, priorities were assigned to the measures proposed by the present author. This led to some changes in both questions and measures. However the presentation of measures to stakeholders, and subsequent prioritization, only confirmed the focus of the programme. Whether or not the measures suffice to provide the desired information is hard to determine. The list of measures, as provided in chapter 6, is simply too abstract for stakeholders to get a clear picture about the information that results from a particular measure. Therefore the decision was taken to create a number of *charts*, *tables* and *forms* that provide stakeholders with a more visual representation of the information they can expect from the programme. This decision is in line with the *indicator* step described by Park *et al.* in §2.3.1. Furthermore it's in line with work of Solingen *et al.*, who advices to create an "analysis plan" [Sol99] during GQM definition to facilitate data analysis and interpretation by stakeholders. It's important to note that indicators in a typical analysis plan contain *hypothetical* data, since no actual measurement data is available yet.

To give stakeholders a clear picture of the measurement programme, it is key to workout the entire process from data registration to measurement presentation. In order to do so, the decision was taken to create an *analysis plan* in the form of a prototype *Measurement Support System* (MSS).

7.1.1 Construction

A MSS is considered to be an essential part of every measurement programme [Sol99], since it supports stakeholders during various stages of the measurement process. The MSS constructed during this thesis project facilitates registration of measurement data, storage of data and presentation of measures in the form of charts and tables (appendix D). Since this MSS concerns a prototype, it was important to archive rapid results. Therefore it was decided to build the system with help of a spreadsheet tool (MS Excel). Other, more important, reasons for using spreadsheets were *flexibility* and *accessibility* [Sol99]. Since a measurement programme is typically subjective to change, the MSS should be easily adaptable to changing requirements. Furthermore, accessibility is important to facilitate interpretation by stakeholders (during feedback sessions §2.3.1). A third reasons for using spreadsheets, is that current *status reports* (§4.1) are also written in Excel. Which indicates that stakeholder are already familiar with spreadsheet usage.

Results

The effort put into constructing the MSS resulted in a spreadsheet (appendix D), in which data is registered and presented on three different levels of granularity (§6.1). Namely; *project*, *iteration* and *activity* (the *discipline* level is incorporated in the activity level). Users of the MSS arn't,

however, required to actively register data on all levels of granularity. Most measurement data can be periodically filled in on the "progress" level, and aggregated to higher granularity levels. Which means that users only need to *track* a few basic attributes (i.e. effort, size and start/end dates) to produce the majority of the measures proposed in chapter 6. The latter mainly applies to productivity-related measures. Defect measurement comprises only one sheets, and registration of defect data is fairly straightforward[1]. Apart from data registration, the MSS included indicators for analysis and interpretation purposes. Specifically; *bar* charts (M5, M6, M15 & M19), *line* charts (M4 & M9), *pie* charts (M17, M21 & M22) and a *bubble* chart (M13).

7.1.2 Usage and Validation

In order to validate the MSS, a series of personal *walkthroughs* [Lau02] with key stakeholders were organized. These stakeholders were selected because they're directly involved (responsible, accountable or informed §7.2.2) in the registration and analysis of measurement data. During the walkthroughs every sheet in the MSS was discussed, and stakeholders were encouraged to comment on the charts and tables presented. As indicated the presented charts, tables and sheets were filled with *dummy data*. Furthermore questions were asked such as "what does this chart tell you?" or "what other information do you need in order to understand this chart?" and "who else would benefit from this information?". The walkthroughs were organized as follows:

- Stakeholders included a project manager, a controller, and a contract manager.
- The initial MSS used in walkthroughs included $\pm$ 10 measures.
- After walkthroughs 2 new measures were added, and 3 were refined.

Results

During the walkthroughs it was noticed that stakeholders got a better understanding of certain measures just by looking at specific charts. Even though the data was fake, stakeholders were able to provide feedback on the measures proposed. Examples of concrete changes incorporated in the measurement programme as a result of this feedback are: progress registration of activities based on effort and duration (fig. D.6), and the cumulative development of effort over time (fig. D.9). Also, while discussing the MSS, numerous new questions and measures arose. It was decided, however, not to include every new idea since it would complicate initial programme implementation. After all, the measurement programme should *start small* and extend over time (§2.4).

Apart from validation walkthroughs, the MSS is also suited for future use in *pilot projects*. That is, the MSS is developed as an *evolutionary* prototype and can therefore be extend with new functionality as the programme progresses.

Validity

Because the MSS lacks *real* measurement data, it was not possible to validate whether the measures (in the MSS) *actually* satisfy the goals and questions stated (in chapters 4 and 5). However, the walkthroughs provided enough insight to determine whether or not the measures *can* satisfy goals and questions. That is, the walkthroughs aimed at validating the measurement programme, by checking whether the measures have the *potential* and/or *ability* to satisfy stakeholder information needs (i.e. questions $\Rightarrow$ goals). As a result, it isn't possible to draw conclusions about validity of the measures with respect to the satisfactory of questions and goals. Nevertheless, it *is* concluded that the selected measures (appendix F) are useful to Daniro for future *research* and/or organizational *implementation*.

7.2 Measurement Specifications

Thus far, attention has been given in this thesis to the identification of *goals*, the formulation of *questions*, and the definition of *measures*. This has led to a comprehensive list of software measures, which essentially forms the foundation of the measurement programme. However a clear *specification* of these measures and associating working methods is missing. Measurement

[1] Especially since defect sheets in the MSS, as well as others, contain *drop-down lists* to simplify data entry.

specifications are important because stakeholders need to understand what the measured values represent [Wie97]. Park *et al.* state "only then can we expect people to collect values consistently, and only then can others interpret the results correctly and apply them to reach valid conclusions" [Par96]. Therefore, with respect to research question nr. 10, the decision was taken to investigate: what information should to be specified in the programme about software measures. The forthcoming two paragraphs briefly highlight a few areas relevant to this research question. In addition, measurement specifications can be found in appendix C. Due to time constraints, however, not all thirteen measures which have been selected for initial implementation §6.3 have been (formally) specified. Instead, two clearly different[2] measures (M6 and M15) were selected which served as input for the example specifications in appendix C.

7.2.1 Formalizing measures

As indicated before, the measures listed in chapter 6 are loosely defined. In order to make these measure *operational* (i.e. repeatable and communicable to others [Par96]) a more formal specification is required. There is, however, no general agreement about the contents of such specification. This is presumably caused by *contextual issues*. For instance, it was noticed that measurement specifications play a prominent role in "formal environments", such as the US DoD. Here measures are specified in great detail (examples [Jon03c; Par96; Bas95]), while in other environments less attention is devoted to this matter (examples; [Sol99; Bir97]). Nevertheless, most literature on software measurement mandates a form of specification[3]. Therefore, the following (typical) elements were included in the example specifications of appendix C:

- glossary of terms and/or abbreviations [Lis05; Jon03c]
- possible limitations and/or side-effects [Kan04; Sol99]
- formula required for measurement calculation [Lis05; Jon03c]
- measurement type (direct/indirect and objective/subjective) [Kan04; Jon03c]
- dependencies [Lis05]

The latter is an (optional) set of references to other measures that provide information relative to the current measure, or that need to be used in conjunction with the current measure for the sake of interpretation validity. Although most elements listed above are straightforward, the measurement *type* was difficult to specify indisputably: whether or not a measure is direct/indirect (and partially subjective/objective) depends one the interpretation of stakeholders [Kan04]. In this case, however, the decision was made to classify a measure as "direct" if it includes a *single attribute*, and "indirect" when it requires *multiple attributes*.

Measurement Theory

This reliance on measurement *attributes* emphasis the importance of measurement theory (§2.2.1). Which concerns amongst others, the definition and validation of theories underlying software measures. There is, however, so much *controversy* about the use of measurement theory (and specifically *scales*) in literature, that it is impossible to be certain about the validity of measures. A study in this direction is also outside the scope of this research. Therefore it is decided to act pragmatically on this matter and only specify the *entities*, *attributes* and *unit(s)* or *range(s)* of each measure. The latter is in line with the case study of [Loc03], concerning the specification and validation of requirements management measures. The advantage of specifying these elements is twofold: first it can be seen as an initial step toward *theoretical validation* [Kit95]. Secondly, it informs stakeholders about the properties of the software process that a certain measure is suppose to quantify [Kan04; Fen94]. Considering measurement attributes, distinction is made between *internal* and *external* attributes. An internal attribute of a certain entity depends only on the entity itself (i.e. size). While an external attribute depends on the context of the entity in question (i.e. maintainability). From a practitioners perspective, external attributes are the most interesting.

[2] One measure is concerned with *productivity*, the other with *defects*

[3] It goes, however, beyond the scope of this research to provide an exhaustive list of specification elements. So only elements supported by literature, that are relevant in the context of this measurement programme are highlighted.

To conclude, there are many more things that one can specify about software measures. However, based on the literature consulted, the elements specified in appendix C are expected to suffice for initial implementation.

7.2.2 Data Collection and Reporting

Besides a specification of the measure itself, it is (even more) important to specify related working methods [Sol99]. Such as; procedures for *data collection* and *reporting* (e.g. analysis/interpretation). Like the "formalization" of measures (§7.2.1), there are tons of properties that one can specify about matters related to data collection and reporting. However, the advice here is also to "keep things simple" (§5.2). Furthermore the availability of the MSS takes some of the tedious data registration/reporting work out of hands. Meaning that detailed descriptions of, for instance "data elements" [Par96], can be omitted from specification.

Referring literature, it becomes clear that *responsibilities* and *time* make up the primary elements of most data collection and reporting procedures. That is, people involved in measurement activities need to be appointed, and their responsibilities should be defined. In order to do so the present author adopted the simple but well-known RACI[4] responsibility model. In addition, the time (e.g. activity or phase) at which data collection takes place needs to be specified. This is organized according to the granularity levels discussed in §6.1 (e.g. collect after iteration X). Furthermore the *instrument(s)* used for data collection and reporting are specified [Sol99; Jon03c]. Since the programme is only in its initial stage, *no* extensive procedures for (statistical) analysis are specified. Optional interpretation hints [Lis05] are, however, included to aid in feedback sessions (§2.3.1).

7.3 Concluding remarks

In this chapter a MSS was constructed, for the purpose of supporting (future) *pilot projects*. Additionally the MSS served as a means of programme *validation*, which in this case refers to a number of *walkthroughs* performed with key stakeholders. During these walkthroughs measurement indicators were presented and discussed. The latter appeared to give stakeholders a (more) concise mental picture of the measurement programme. Nevertheless, more validation work is needed in this area. Furthermore in this chapter, a *first attempt* was made to investigate what aspects about software measures need to be specified in the measurement programme. This led to the realization of two example specifications; describing measurement characteristics and data collection procedures.

Near the end of this thesis project, it is concluded that a number of important measurement issues have been addressed. This includes amongst others, the identification of goals, the definition of measures, and the first steps towards the establishment of measurement protocols (e.g. specifications, MSS). Relating this effort to the Measurement-CMM [Nie98] of Niessink and Van Vliet, learns that the current programme is focused on achieving maturity level 2 "repeatable". More specific, the *measurement design* key process area is addressed (listed in appendix E). The latter Measurement-CMM illustrates that the current programme is only in its initial stage. Meaning that a lot of work is yet to be done. Specifically measurement *implementation* (i.e. collection, analysis and feedback), which is known to account for roughly 70% of the total programme effort [Kil01]. However, measurement *design* requires also additional research. Nevertheless, this research brought a number of important prerequisites (of the measurement design KPA from appendix E) in place for Daniro, to archive a higher level of measurement/improvement capability.

[4] **R**esponsible; people that get the job done,
Accountable; people that take the credit for success or responsibility for failure,
Consulted; people whose opinions are sought,
Informed; people that are kept up-to-date on progress.
SOURCE: Wikipedia

Chapter 8

Conclusions and Future Work

THIS chapter summarizes the conclusions drawn, with respect to the results achieved during the thesis project. An evaluation is also given about the work performed and the difficulties/opportunities encountered. Furthermore a recommendation for programme implementation is provided, and a number of (secondary) ideas for future work are discussed.

8.1 Conclusions

The objective of this thesis project was to investigate the possibilities of establishing a measurement programme at Daniro J-Technologies. Thereby the primary focus laid on programme *design*. In light of this, the following central research question was posed: "*what process-related measures, with respect to organizational goals, can be defined to satisfy the information needs of stakeholders in the context of software development projects?*". In order to provide an answer to this central question and meet the objective of the thesis project, a number of detailed research questions were formulated (§1.2.2). These research questions have been answered in the body of this thesis. In this section, conclusions are drawn with respect to the *main* questions[1,2] stated.

8.1.1 Information; *strategic measurement for understanding purposes*

Based on stakeholders interviews, it is concluded that Daniro J-Technologies is focused on four *measurement goals*; understanding productivity, baselining productivity, understanding defects and understanding scope$_2$. As a result the organization is willing to establish a *strategic* measurement programme$_1$, because the goals apply to generally any development project. Within these software development projects there are usually five groups of stakeholders to discern: organization management, project management, functional design, technical construction and testing$_3$. The information needs of these stakeholders served as a basis for *questions* formulated, in response to measurement goals. However, the proposed measurement programme addresses mainly management-related information needs$_4$. This stems from the fact that high priority goals and questions are formulated from a management point of view. The rationale behind the latter is that managerial stakeholder are primarily focused on *process* measurement. In contrast functional, software- and test engineers are more concentrated on (detailed) *product* measurement.

The information needs of organization- and project management stakeholders are, however, not the same. Project management is generally concerned with *controlling* cost and resources aspects *in* projects. Whereas organization management is concerned with *improving* the allocation and use of these aspects *across* projects$_5$. Nevertheless, since the measurement programme is only in its initial (i.e. planning and definition) stage, *understanding* of these aspects must be reached first$_1$. Therefore it is decided to focus locally on project-level measurement, but facilitate future organizational use. In short, the purpose of the measurement programme is (in order of priority); understanding *productivity*, *defect* reporting and *scope* changes in software development projects. The *scope* goal is, however, excluded from initial organizational implementation.

[1] Since the research questions in §1.2.2 are ordered by topic, the conclusion is structured accordingly

[2] Conclusions are marked with numbers in subscript (e.g. $_3$), that refer to (answered) research questions listed in §1.2.2

8.1.2 Measurement; *project measures to address multiple stakeholders*

In literature, there's a wealth of software measures available in all sorts of areas (e.g. risk management, technical complexity, product stability)$_{7}$. Also at Daniro various measures were found, which can roughly be divided in two categories: source code quality measures and finance-related measures$_{7}$. In order to decide what measures needed to be included in the programme, the GQM method was applied$_{6}$. Based on the information needs (i.e. questions) of stakeholders discussed above, relevant measures were studied. In light of this, the conclusion is drawn that most measures in the programme relate to productivity$_{6}$. That is, measures of *duration*, *effort* and *size*. It should be noted, however, that only a small number of measures *directly* concern productivity in the economical sense of size/effort (e.g. # function points per hour). Factors *depending* on productivity (schedule adherence, costs) and factors *influencing* productivity (readiness for release, staffing) account for the majority of the productivity-related information needs, and thus measures$_{5,8}$. With respect to defect information needs, measures focused on understanding defect *impact* and *effort* in development projects were included.

As concluded earlier, *managerial* stakeholder acted as the primary source of information in this research (§8.1.1). However, it's important to note that *engineers* (and other project members) can also benefit from the proposed measurement programme. For instance, future decisions based on the proposed measures directly affect the work of engineers participating in development projects. Moreover, software process methodologies focused on individual engineers (PSP) or engineering groups (TSP), also prescribe the type of software measures discussed in this thesis$_{8,14}$.

8.1.3 Organization; *measures in outsourcing projects*

During the process of measurement definition, it was noticed that it's in some cases difficult to mandate a standard set of software measures in development projects. Meaning that projects are in nature "unique", which translates to *variations* in techniques, methods and teams$_{11}$. The precise impact and extend of variation is, however, unknown by the present author. Nevertheless, the rationale behind the observed variation is assumed to be caused by the fact that Daniro is a service-oriented organization. This makes measurement somewhat harder since *outsourcing* projects are in certain ways *customer imitated*$_{11}$. This is different from most measurement programmes discussed in literature, which are established in more formal organizations. Nevertheless, since the current programme focuses on strategic measurement, the case is made that a standard set of software measures can still be established. Especially since the organization is increasingly focused on *operation excellence*, of which the *software factory* initiative is a good example$_{15}$.

Overall, it is concluded that a number of important measurement issues have been addressed in this research. This includes amongst others, the identification of goals, the definition of measures, and the first steps towards the establishment of measurement protocols. These activities brought a number of prerequisites in place for Daniro, to archive a higher level of measurement/improvement capability (Measurement-CMM, CMMI)$_{14}$. Nevertheless, this effort only focused on programme *design*. Most work is yet to be done in in the field of measurement *implementation* (i.e. collection, analysis and feedback).

8.1.4 Final conclusion

Considering the central research question posed in §1.2.2; the present author concludes that primarily *productivity*- and *defect*-related measures can be defined, with respect to organizational goals. Thereby productivity measures relate to *duration*, *effort* and *size*, whereas defect measures focus on understanding *impact* and *effort*. To conclude the latter, stakeholders from various disciplines were involved in the measurement design process. And the context of *development projects* was also taken into account. However, further (practical) research is required to conclude whether the proposed measures satisfy the *information needs* of *stakeholders* in development projects. Hence, this research can be classified as a first (theoretical) step toward the establishment of a measurement programme at Daniro J-Technologies.

8.2 Evaluation

This (personal) *evaluation* reflects on the work performed and process followed during the thesis project, as well as the difficulties/opportunities encountered. The paragraph is organized according to the three *types of activities* defined in the research approach of fig. 3.1 (i.e. *information*, *GQM* and *validation*).

8.2.1 Information gathering

A significant part of this research was spent on information gathering. This refers to both *literature research*, as well as *knowledge acquisition* through interviews and conversations. Overall literature research went rather well, although not all measures and related techniques applied to the case of Daniro, the concepts found (e.g. GQM, SPI, ...) proved to be useful. Interviews on the other hand, took up a lot of time (i.e. preparation, analysis, reasoning, relating literature) which wasn't quite expected. Nevertheless, the resulting information contributed to a better understanding of organizational goals and context. Additionally, as an indicator of success, most issues discussed with stakeholders were incorporated in the programme.

Even though knowledge acquisition delivered plausible results, it is certainly a difficult matter. For instance, it was found that stakeholders have divergent information needs, and different views on software measurement concepts. Understanding these issues (and associating jargon) required insight in the (software development) project environment. This was a challenging undertaking considering the limited time available. Especially for someone (inexperienced and) unfamiliar with the organization; like the present author. Therefore, the next time, the present author would likely start with a "kick-off session". That is, a workshop or presentation about software measurement programmes (targeted at project participants). This session should take place *before* knowledge acquisition (§4.2, e.g. GQM interviews [Sol99]). In this way, it might be possible to get stakeholders on the same line, and formulate goals (and rudiment questions) at programme *inception*. So to speak, this could (drastically) shorten the time spent on knowledge acquisition. The questions remains, however, whether this approach would have worked in the context of this research. Since the scope at project inception was rather wide, and stakeholders weren't (all) known.

8.2.2 Goal/Question/Metric method

One result of literature research was the adopted of the GQM method. This method appeared to be useful because of its relative *flexibly* and *simplicity*. Flexible, because GQM is not focused on a particular type of stakeholder and suits both project and organization level measurement. Simple, because the basics of GQM can be explained to someone unfamiliar in the field in just a couple of minutes. However, even though GQM may sound like a straightforward method, practice was somewhat more complicated. For example:

- In literature (e.g. [Sol99; Par96]) GQM is often presented as a stepwise process, with extensive *user involvement*. This includes frequent meetings, interviews and presentations with stakeholders. However, frequent meetings arn't easily arranged when:
  - Stakeholders and scope of the programme aren't/isn't known/decided yet
  - Commitment is hardly established, since the programme is its initial/research stage
  - There isn't an *urgent* need to establish a measurement programme[3]
- Goals are difficult to formulate, when stakeholders have divergent needs/views. Especially when the above issues (like urgency) are taken into consideration.
- Goals only represent *consensus* between stakeholder concerns (solution: viewpoints §4.3.1)
- Formulating questions at the "right" level of abstraction takes a number of iterations.
- GQM offers little support with the actual study and *definition* of measures (as opposed to, e.g. PSM or more statistically-oriented methods)[4].

[3] Stakeholders saw measurement as a useful contribution to the continuous effort for (organizational) improvement. However, there wasn't an "alarming situation" in the sense that measurement needed to be established to prevent from "disaster". Although this is a positive observation, it makes establishing commitment and motivation a bit harder [Can06].

[4] For instance; there are no *standard* measures to choose from, or specific criteria for measurement definition/formalization available. GQM assumes a more or less "unique" situation/context for each measurement programme.

The present author noticed the above issues during GQM application. However, these issues arn't (all) caused by GQM. Some originate from inexperience, others because of contextual limitations. Overall, however, GQM was found to be a *valuable method* for structuring the measurement programme (as well as this research/thesis). Question definition for instance, was helpful because it aids in understanding stakeholder needs, and provides measures with an explicit purpose.

8.2.3 Validation and implementation

Scientific as well as practical *validation* was difficult in this research, since the focus laid on *design*-aspects of the measurement programme. Although attention was given to validation (i.e. prioritization, presentation, walkthroughs, literature survey), it was noticed that the programme could only be validated *for real* after or during implementation. That is, when *actual* data is collected and measurement results are analyzed and interpreted. In that case, conclusions can be drawn about the success/validity of the programme *with respect to goal attainment*. However, due to time constraints, organizational implementation was out of scope. On the other hand, there were (as indicated in §1.2.1) important reasons to focus on measurement programme design (i.e. definition and planning in [Sol99] terms). Definition is after all a mandatory step *before* data collection and measurement analysis/interpretation. Nevertheless, excluding implementation from the present research posed a number of difficulties. For instance, with help of presentations and walkthroughs, it was not possible to validate whether the proposed measures *actually* satisfy the goals and questions stated. The present author recognizes this (potential) weakness in the research. Nonetheless, the thesis project resulted in the initial design of a *well-defined* measurement programme. Which, by belief of stakeholders and the present author, greatly *raises* the chance of successful implementation (as described in the following paragraphs).

8.3 Future Work

Quite some future work results from this thesis project due to its (partially) *exploratory* nature (§3.1). To structure this future work, difference is made between activities requiring immediate attention (*recommendation*) and activities of lower priority (*remaining work*).

8.3.1 Recommendation

The research in this thesis project focused only on the *definition* (and to some extend *planning*) part of the measurement programme. However, by merely defining the programme no real improvement is made. Therefore, in order to achieve the goals stated, one or more *pilot projects* have to be selected for *implementation*. This selection should be made with care, since the success of the entire measurement programme depends on the experiences gained from these projects. Hereby one should make sure that, amongst other, proper measurement specifications (§7.2) are in place to inform stakeholders about the programme. Once pilot projects have been selected, measurement data collection can take place. In this case, the present author suggests to start small. Begin on the *basic* level (figure 8.1) by collecting high priority productivity and defect data (appendix F). Let team leaders and project managers register this data in the (prototype) *Measurement Support System*, and allow them to submit periodic status reports to the Project Office (§4.1). Once these activities go as planned, programme implementation can be extended. However, it is not recommenced to directly increase the number of measures. One should first try to focus on more effective use of existing measurement data. Start by making *hypotheses* [Sol99] about measurement outcome, before collecting the actual data, in order to increase the overall learning capability. Then start with *feedback sessions* (§2.3.1) to provide participants with up to date information about the progress and quality of the project. In this way everybody will feel they have something to gain from the programme [Bri96] and measurement will become less of a burden during daily work.

So the main recommendation toward Daniro can be summarized in a single phrase, namely: "implement". Nevertheless, it should be noted that work proposed in this recommendation is largely based on best practices from literature (ch. 2). Although the concepts behind these practices are universally applicable, the exact realization may differ per project or organization. Therefore it is suggested to further investigate these activities during programme implementation.

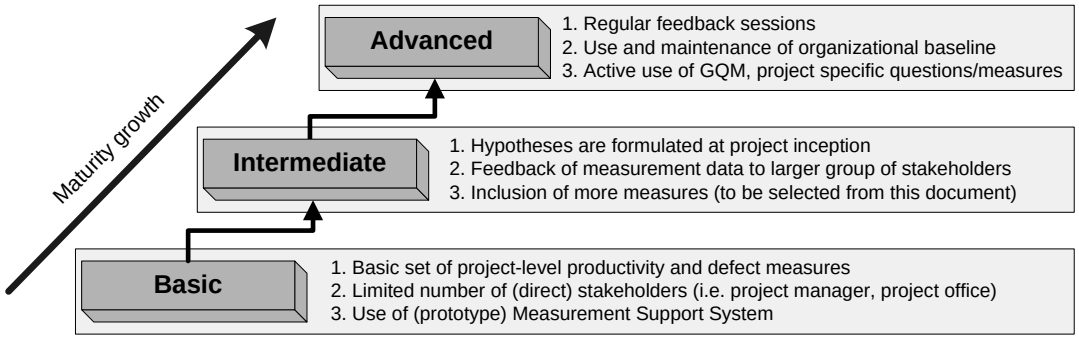


Figure 8.1: *Conceptual model of the proposed introduction plan*

8.3.2 Remaining Work

Apart from the main recommendation concerning measurement programme implementation, the activities listed below are (in no particular order) proposed as future work.

Establish measurement infrastructure

Although a prototype Measurement Support System was developed (§7.1), a complete measurement infrastructure is currently missing. Most of the information used in the measurement programme needs to be collected manually during software development projects. Although, some information such as defects and hour logs are already registered in an information system, this data still needs to be extracted and converted (to spreadsheets) by hand in order to calculate the required measures. Since the measurement programme is only in its initial phase an advanced measurement infrastructure is currently not required. However, as the organization and the measurement practice grows more mature the need for measurement support tools is assumed to increase. Therefore a study concerning the establishment of a measurement infrastructure is suggested. Ideally this infrastructure would integrate seamlessly with the existing software development line (§1.1) to provide a single platform for project support.

On the commercial market several measurement support systems are available from vendors such as IBM Rational (ProjectConsole), QSM (SLIM Metrics), Distributive Management (DataDrill) and Predicate Logic (TychoMetrics). There are also a few early open source initiatives such as Hackystat[5] from the University of Hawaii and PROM[6] by the Free University of Bolzano-Bozen. Furthermore a GQM specific measurement system exists which is marketed by VTT Electronics (MetriFlame). However it is unknown whether this system is still maintained. A study concerning the establishment of a measurement infrastructure should turn out which, of these, tools suit the measurement programme best. Furthermore this research should focus on the customization of existing systems (eg. issues tracking and hour registration) with respect to the measurement programme described in this thesis.

Provide training in software measurement

Training of employees is considered a critical factor (§2.4) in the successful establishment of software measurement programme. Therefore it is suggested that employees at Daniro, who are affected by the programme, receive training in order to use measures properly and effectively during their daily work. It is expected that, depending on the role of the individual, a one-day training (or "development day" as this is called in Daniro terms) would suffice in order to introduce employees to the concepts and practical use of software measurement. For the initial programme presented in this thesis a full day training course will not be necessary, since it only impacts a relatively small number of project stakeholders. However, as more people are affected by the programme (due to increased implementation) training is assumed to become a prerequisite for success. This is espe-

[5] `http://www.hackystat.org`
[6] `https://prom.case.unibz.it/`

cially true when measurement practice advances (figure 8.1) and for example; feedback sessions are organized, organizational benchmarking is set-up and GQM is used by project participants.

Produce and analyze software process models

Software process modeling is considered to be an important activity in establishing a measurement programme [Sol99; Bri96]. By explicitly modeling process entities (e.g. activities, roles, tools and artifacts) and the relations among them, a more formal understanding of the software process can be reached. Hereby it is vital to concentrate on *descriptive* models, that capture the process as it is actually takes place in practice. This allows one to, amongst others, pin-point objects to be measured and people responsible for measurement. Furthermore it helps to identify the existence of process inconsistencies that could influence measurement implementation or outcome [Som99].

However due to the *exploratory* (§3.1) character of this research, as well as the initially wide scope and the limited amount of time available, process modeling was not applied in this thesis project. Nevertheless, modeling is advisable for reasons described above and therefore suggested as future work. In the academic world, software process modeling already received a lot of attention. As a result a fair number of process modeling techniques and formalisms have been proposed. Examples are: MVP-L, APPL/A, APEL and SLANG. However, these formalisms all serve a different purpose and are not equally suited for the job. Yet, in the case of Daniro, *understanding* the software process is seen as the primary goal (as opposed to automatic process enactment or process programming [Cur92]). Therefore the present author recommends the use of SPEM; the Software Process Engineering Metamodel[7]. This graphical formalism, based on UML, is also used in process models of the RUP methodology. Hence it is assumed that stakeholders can adopt this formalism with relative ease, since both UML and RUP are practiced at Daniro.

[7] `http://www.omg.org/technology/documents/formal/spem.htm`

Bibliography

[Aae97] I. Aaen, P. Bøtcher, and L. Mathiassen. Software factories. In *Proceedings of the 20th Information Systems Research Seminar*. 1997. (cited on page 1).

[Abr04] A. Abran, P. Bourque, R. Dupuis, and J. W. Moore, editors. *Guide to the Software Engineering Body of Knowledge (SWEBOK)*, chap. 8 "Software Engineering Management", pp. 1–13. IEEE Press, 2004. (cited on pages 2 and 6).

[Bas94] V. R. Basili, G. Caldiera, and H. D. Rombach. The goal question metric approach. *Encyclopedia of Software Engineering*, vol. 1:pp. 578–583, 1994. (cited on pages ix and 10).

[Bas95] M. J. Bassman, F. McGarry, R. Pajerski, et al. Software measurement guidebook. Tech. Rep. SEL-94-102, NASA Goddard Space Flight Center, Software Engineering Laboratory, 1995. (cited on pages 6, 10, 12, 13, 24, and 39).

[Bir97] A. Birk, P. Giese, R. Kempkens, D. Rombach, and G. Ruhe, editors. *Goal-Oriented Measurement using GQM*. PERFECT consortium, Fraunhofer IESE, 1997. (cited on pages 11 and 39).

[Boe01] B. Boehm and V. R. Basili. Software defect reduction top 10 list. *IEEE Computer*, vol. 34(1):pp. 135–137, 2001. (cited on page 28).

[Bri96] L. C. Briand, C. M. Differding, and H. D. Rombach. Practical guidelines for measurement-based process improvement. *Software Process: Improvement and Practice*, vol. 2(4):pp. 253–280, Dec 1996. (cited on pages 9, 10, 11, 12, 19, 44, and 46).

[Bro95] F. P. Brooks. *The Mythical Man-Month: Essays on Software Engineering, 20th Anniversary Edition*. Addison-Wesley Professional, August 1995. (cited on page 26).

[Can03] J. J. Cannegieter. *Software Process Improvement*. Ten Hagen en Stam Uitgevers, 2003. (cited on page 5).

[Can06] J. J. Cannegieter and R. van Solingen. *De Kleine CMMI: De basisuitrusting voor continue prestatieverbetering*. Academic Service, 2006. (cited on pages 5, 43, and 69).

[Car98] D. N. Card. Learning from our mistakes with defect causal analysis. *IEEE Software*, vol. 15(1):pp. 56–63, 1998. (cited on page 34).

[Car05] E. R. Carroll. Estimating software based on use case points. In *OOPSLA '05: Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pp. 257–265. ACM Press, New York, NY, USA, 2005. (cited on page 9).

[Chi92] R. Chillarege, I. Bhandari, J. Chaar, M. Halliday, D. Moebus, B. Ray, and M.-Y. Wong. Orthogonal defect classification - a concept for in-process measurements. *IEEE Transactions on Software Engineering*, vol. 18(11):pp. 943–956, 1992. (cited on page 33).

[Cur92] B. Curtis, M. I. Kellner, and J. Over. Process modeling. *Communications of the ACM*, vol. 35(9):pp. 75–90, 1992. (cited on pages 19 and 46).

[Das92] M. K. Daskalantonakis. A practical view of software measurement and implementation experiences within motorola. *IEEE Transactions on Software Engineering*, vol. 18(11):pp. 998–1010, 1992. (cited on page 12).

[Dia97] M. Diaz and J. Sligo. How software process improvement helped motorola. *IEEE Software*, vol. 14(5):pp. 75–81, 1997. (cited on pages 23 and 25).

[Dob88] J. Dobbins et al. Guide for the use of: IEEE standard dictionary of measures to produce reliable software. Tech. Rep. IEEE Std 982.2, IEEE Computer Society, 1988. (cited on pages 34 and 35).

[Fen94] N. E. Fenton. Software measurement: A necessary scientific basis. *IEEE Transactions on Software Engineering*, vol. 20(3):pp. 199–205, 1994. (cited on pages 7, 8, and 39).

[Fen00] N. E. Fenton and M. Neil. Software metrics: roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, pp. 357–370. ACM Press, New York, NY, USA, 2000. (cited on pages 7 and 32).

[Fre01] B. Freimut. Developing and using defect classification schemes. Tech. Rep. 072.01/E, Fraunhofer IESE, September 2001. (cited on page 33).

[Gar06] F. García, M. F. Bertoa, C. Calero, A. Vallecillo, F. Ruiz, M. Piattini, and M. Genero. Towards a consistent terminology for software measurement. *Information & Software Technology*, vol. 48(8):pp. 631–644, 2006. (cited on page 8).

[Gra94] R. B. Grady. Successfully applying software metrics. *IEEE Computer*, vol. 27(9):pp. 18–25, september 1994. (cited on pages 8 and 21).

[Gre03] J. Greenfield and K. Short. Software factories: assembling applications with patterns, models, frameworks and tools. In *OOPSLA '03: Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pp. 16–27. ACM Press, New York, NY, USA, 2003. (cited on page 1).

[Gri00] R. Grit. *Projectmanagement*. Wolters-Noordhoff, 2000. (cited on page 6).

[Hal97] T. Hall and N. E. Fenton. Implementing effective software metrics programs. *IEEE Software*, vol. 14(2):pp. 55–65, 1997. (cited on pages 12 and 13).

[Hum96] W. S. Humphrey. Using a defined and measured personal software process. *IEEE Software*, vol. 13(3):pp. 77–88, 1996. (cited on page 28).

[Jon95] C. Jones. Backfiring: Converting lines-of-code to function points. *IEEE Computer*, vol. 28(11):pp. 87–88, 1995. (cited on page 26).

[Jon96] C. Jones. Software change management. *IEEE Computer*, vol. 29(2):pp. 80–82, 1996. (cited on pages 27 and 54).

[Jon03a] C. Jones. Variations in software development practices . *IEEE Software*, vol. 20(6):pp. 22–27, 2003. (cited on pages 35 and 36).

[Jon03b] C. L. Jones. Implementing a successful measurement program: Tried and true practices and tools. *Cutter IT Journal*, vol. 16(11):pp. 12–18, nov 2003. (cited on pages 11 and 12).

[Jon03c] C. L. Jones et al. Practical software and systems measurement: A foundation for objective project management, March 2003. Guide v4.0b1, Part I, Available online (registration required): `http://www.psmsc.com/`. (cited on pages ix, 11, 12, 13, 20, 39, and 40).

[Kan04] C. Kaner and W. P. Bond. Software engineering metrics: What do they measure and how do we know? In *10th International Software Metrics Symposium (METRICS 2004)*. September 2004. (cited on pages 6, 27, and 39).

[Kil01] T. Kilpi. Implementing a software metrics program at nokia. *IEEE Software*, vol. 18(6):pp. 72–77, 2001. (cited on pages 36 and 40).

[Kit95] B. Kitchenham, S. L. Pfleeger, and N. E. Fenton. Towards a framework for software measurement validation. *IEEE Transactions on Software Engineering*, vol. 21(12):pp. 929–944, 1995. (cited on pages 8 and 39).

[Kit97] B. Kitchenham, S. L. Pfleeger, and N. E. Fenton. Reply to: Comments on "towards a framework for software measurement validation". *IEEE Transactions on Software Engineering*, vol. 23(3):p. 189, 1997. (cited on page 8).

[Lat98] F. van Latum, R. van Solingen, M. Oivo, B. Hoisl, D. Rombach, and G. Ruhe. Adopting GQM-based measurement in an industrial environment. *IEEE Software*, vol. 15(1):pp. 78–86, 1998. (cited on pages 10, 12, and 18).

[Lau02] S. Lauesen. *Software Requirements: Styles and Techniques*. Addison-Wesley, 2002. (cited on pages 38 and 54).

[Lis05] B. List, R. M. Bruckner, and J. Kapaun. Holistic software process performance measurement: From the stakeholders' perspective. In *International Workshop on Business Process Monitoring & Performance Management (BPMPM'05)*, pp. 941–947. 2005. Workshop proceedings published as part of DEXA'05. (cited on pages 39 and 40).

[Loc03] A. Loconsole and J. Börstler. Theoretical validation and case study of requirements management measures. Tech. Rep. UMINF 03.02, Umeå University, July 2003. (cited on page 39).

[Low90] G. C. Low and D. R. Jeffery. Function points in the estimation and evaluation of the software process. *IEEE Transactions on Software Engineering*, vol. 16(1):pp. 64–71, 1990. (cited on pages 8 and 9).

[Max01] K. D. Maxwell. Collecting data for comparability: Benchmarking software development productivity. *IEEE Software*, vol. 18(5):pp. 22–25, September/October 2001. (cited on pages 20, 21, and 32).

[McC00] S. McConnell. The best influences on software engineering. *IEEE Software*, vol. 17(1):pp. 10–17, 2000. (cited on page iii).

[McC04] S. McConnell. *Code Complete, Second Edition*, chap. 27, pp. 649–659. Microsoft Press, Redmond, WA, USA, 2004. (cited on page 32).

[Min00] A. Minkiewicz. Software measurement - what's in it for me? In *Software Management/Applications of Software Measurement Conference (SM/ASM 2000)*. March 2000. (cited on page 28).

[Miz98] O. Mizuno, T. Kikuno, K. Inagaki, Y. Takagi, and K. Sakamoto. Analyzing effects of cost estimation accuracy on quality and productivity. In *ICSE '98: Proceedings of the 20th international conference on software engineering*, pp. 410–420. IEEE Computer Society, Washington, DC, USA, 1998. (cited on page 9).

[Mor97] S. Morasca, L. C. Briand, E. J. Weyuker, V. R. Basili, and M. V. Zelkowitz. Comments on "towards a framework for software measurement validation". *IEEE Transactions on Software Engineering*, vol. 23(3):pp. 187–188, 1997. (cited on page 8).

[Nie98] F. Niessink and H. van Vliet. Towards mature measurement programs. In *CSMR '98: Proceedings of the 2nd Euromicro Conference on Software Maintenance and Reengineering*, pp. 82–88. IEEE Computer Society, Washington, DC, USA, 1998. (cited on pages 40 and 69).

[Nie01] F. Niessink and H. van Vliet. Measurement program success factors revisited. *Information and Software Technology*, vol. 43(10):pp. 617–628, August 2001. (cited on pages 12 and 13).

[Off97] R. J. Offen and R. Jeffery. Establishing software measurement programs. *IEEE Software*, vol. 14(2):pp. 45–53, 1997. (cited on pages 8 and 16).

[Par96] R. E. Park, W. B. Goethert, and W. A. Florac. Goal-Driven Software Measurement - A Guidebook. Tech. Rep. CMU/SEI-96-HB-002, Software Engineering Institute, Carnegie Mellon University, August 1996. (cited on pages 7, 8, 10, 12, 13, 19, 24, 39, 40, and 43).

[Put02] L. H. Putnam and W. Myers. Control the software beast with metrics-based management. *STSC CrossTalk*, pp. 19–21, August 2002. (cited on page 32).

[Rif91] S. Rifkin and C. Cox. Measurement in practice. Tech. Rep. CMU/SEI-91-TR-16, Software Engineering Institute, Carnegie Mellon University, July 1991. (cited on pages 12 and 13).

[Rif01] S. Rifkin. What makes measuring software so hard? *IEEE Software*, vol. 18(3):pp. 41–45, 2001. (cited on page 36).

[Sca95] W. Scacchi. Understanding software productivity. In D. Hurley, editor, *Advances in Software Engineering and Knowledge Engineering*, vol. 4, pp. 37–50. World Scientific Publishing Company, 1995. (cited on page 25).

[Sol99] R. van Solingen and E. Berghout. *The Goal/Question/Metric Method: a practical guide for quality improvement of software development*. McGraw-Hill Publishing Company, 1999. (cited on pages ix, 5, 6, 7, 9, 10, 11, 12, 13, 15, 18, 19, 20, 25, 37, 39, 40, 43, 44, and 46).

[Sol00] R. van Solingen. *Product Focused Software Process Improvement*. Ph.D. thesis, Eindhoven University of Technology, The Netherlands, 2000. (cited on pages 11 and 12).

[Som99] I. Sommerville, P. Sawyer, and S. Viller. Managing process inconsistency using viewpoints. *IEEE Transactions on Software Engineering*, vol. 25(6):pp. 784–799, 1999. (cited on pages 19 and 46).

[Sta95] The CHAOS Report. Tech. Rep. T23E-T10E, The Standish Group, 1995. (cited on page 53).

[Wie97] K. E. Wiegers. Metrics: Ten traps to avoid. *Software Development Magazine*, vol. 5(10), 1997. `http://www.processimpact.com/articles/mtraps.pdf`. (cited on pages 12, 30, and 39).

[Wie99] K. E. Wiegers. A software metrics primer. *Software Development Magazine*, vol. 5(7), 1999. `http://www.processimpact.com/articles/metrics_primer.pdf`. (cited on pages 12, 13, and 16).

[Woo01] T. L. Woodings and G. A. Bundell. A framework for software project metrics. In *ESCOM'01: Proceedings of the 12th European Conference on Software Control and Metrics*. 2001. (cited on page 7).

Appendix A

Interview questions

THIS appendix presents an overview of the questions asked during the characterization and goal identification interviews with the various stakeholders.

General questions

1. Can you describe your job, specifically your role in software development projects?
2. What is your view/vision of software projects, processes and measurement?
3. How would you characterize the software development process at Daniro (in terms like orientating, defined, optimal, repeatable, improving, ad hoc, formal, informal, agile, etc)?
4. Do you think that the software process within Daniro needs improvement?
5. What are your biggest concerns in this organization?
6. What would you like to achieve or improve in this organization?

Measurement

7. In what kind of information are you primarily interested?
8. What is your main source of information?
9. Why would you like to start a measurement programme? What is your motivation? What purpose should it serve:
   a) Understand: Provide insight in potential improvement area's
   b) Control: Early identification of problems, signals when steering is needed
   c) Improve:Report the effects of improvement initiatives, techniques, ...
   d) Predict: Make better plans, more accurate (cost) estimates, ...
   e) ...
10. Can you name a few metrics that you're familiar with?
11. What is currently measured / registered in the software project / organization?
12. What are possible measurement goals?
13. What do you expect from measurement results?
14. What do you think is the impact of introducing / continuing a measurement programme?
15. How much time do you expect to spend on collecting, validating, analyzing and interpreting measurement results?
16. How much time would you (or your colleagues) spent at max on this activity?

17. Do you think that software (process) measurement fits within the development line philosophy of Daniro?
18. How do you feel about scaling a possible J-Technologies measurement programme to other Development Centres, such as Microsoft, Oracle, Midrange and Mainframe, . . . ?
19. Is there a lot of variation in software projects? What are the main differences and/or similarities?

Specific questions

Organization Management

1. What are the business goals of J-Technologies?
2. Can you prioritize these goals?
3. How do these goals relate to (high-level) Daniro SI&D goals?
4. Are these goals dependent on the type of project (fixed-price, fixed-date, on-site, in-house, . . .)?
5. Apply these goals to all Daniro projects? What are the main differences and/or similarities?
6. What other factors are of influence on these goals?
7. How are these goals achieved? What information do you need in order to know this?
8. Are there any results of software process assessments, planning evaluations, audits or risk assessments available?

Project Management

9. How is a typical project team assembled? Does this vary a lot?
10. What type of problems occur the most during development projects? Does this also apply to your current project?
11. What kind of problems or issues cost the most?
12. Do you follow a particular type of software process methodology (iterative, incremental, waterfall, RUP, . . .)

Functional

13. Do you see any improvement opportunities in this software project? If so, what would you like to achieve or improve?
14. How to you specify software requirements in this project?
15. How are requirements managed (and traced)?
16. How are requirements validated?
17. How are change requests handled?

Technical

18. Do you follow a particular type of software process methodology (iterative, incremental, waterfall, RUP, . . .)
19. Do you see any improvement opportunities in this software project? If so, what would you like to achieve or improve?
20. What kind of issues do you encounter during your daily work?
21. What (kind of) artifacts do you typically produce during a software project?
22. What kind of tools do you use in your daily work?

Appendix B

Scope Measurement

NOTE: scope measurement is part of the proposed measurement programme. The body of the thesis, however, is mainly focused on initial programme implementation. And since *all* scoping measures are excluded from the *basic* level (§8.3.1) the discussion is placed apart in this appendix.

THE Standish Group indicates in its widely known CHAOS report [Sta95] that incomplete and changing requirements are among the top causes of project failure. Together these issues account for more than 24% of all failures (according to participant responses). The Standish study, as well as others, show that *requirements management* is not a trivial matter in software development projects. This belief was also comfirmed by (some of) the stakeholders at Daniro who placed "manage project scope" as one of their improvement goals in the measurement programme.

Information needs

One problem that stakeholders at different organizational levels face, is managing the *scope* of software projects. That is, what needs to be realized? and how does it affect our project? These issues directly relate to (managing) software requirements.

Although requirements are volatile, there is a certain time during a (fixed-price) project at which requirements are formally approved. Changes in functionality submitted after this date result in *change requests*, which are charged to the customer. A difficult issue in this matter is to establish agreement about the *cost* of change. Currently project managers (together with contract and/or account managers) need to convince the customer of a certain price. However, it is difficult for Daniro to come up with the "right" price. That is, a price acceptable for both Daniro as well as the customer. Price calculation is hard because the impact of change requests is difficult to quantify. This includes the possible impact on *schedule* (increase duration) and *costs* (increased effort) as well as the impact on the product[1]. Hereby it's important to note that there's are difference between *change requests* and *requirements*. As illustrated in B.1 a change request submitted by a

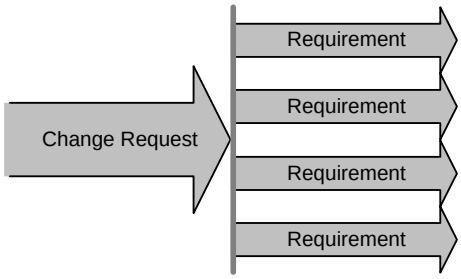


Figure B.1: *From change request to requirements*

[1]Product change, in terms of technical or architectural impact, is however beyond the scope of this research.

customer may, after acceptance, result in several (individual) requirements. This being said, it is vital from a (scope) management point of view to measure both these change requests as well as the resulting requirements.

Quality Point of View

Furthermore, it is expected that measuring change requests results in useful information about more subjective quality matters. Such as the level of *user involvement* and the *stability* of the project. For instance, a sudden increase in change requests after a release (in a late stadium of the project) might indicate insufficient user testing during project inception. These issues are, however, of lower priority than the cost and progress information needs discussed above and therefore not (directly) included as questions, for measurement definition.

Measures

Taking the considerations of the above paragraph into account, the decision was made to measure the *amount of change requests* (M29) as well as the associating *effort of change* (M32). This refers to the budgeted effort, plus actual effort required for change implementation. To aid in the quantification of these costs, a measure of *requirements count* (M25) was included. That is, once stakeholders *understand* the *size* of change requests (through the number of requirements resulting from it), cost estimation and/or price calculation is assumed to become easier.

Requirements count

Measure M25 (in relation to M26) may seem obvious at first sight. However, counting requirements in a modern software development project is not a trivial matter. If one only assumes the use of *feature requirements*[2] then counting is relatively easy. Nonetheless customer demands are hardly ever specified as feature requirements, when one follows modern methods in requirements engineering [Lau02]. Consequently requirement specifications could well contain a mix of use cases, task descriptions, diagrams, models and prototype screens. Even when measure M25 is only limited to *functional requirements* there are still a lot of ways in which requirements can be specified. This is especially true at Daniro where software development projects are conducted for different (type of) customers. As a result elicitation and specification techniques are often selected based on the specific needs of the customer.

Then how to solve this issue one might ask? Unfortunately there is no single answer. Measuring requirements primarily depends on the way requirements are managed throughout the project. Only when requirements management (and associated tooling) is consistently applied, it become possible to faithfully count the number of requirements as well as the development of these requirements over time. In the case of Daniro requirements management is currently not consistent across projects. Changing this situation is inherently difficult and time consuming, hence it is beyond the scope of this thesis. An alternative to this dilemma is the use function- or use case points as a measure of requirements size. Functional size measurement is after all a measure of the functionality from the perspective of what gets delivered to the (end)user (§2.2.2. However, an important disadvantage in scope measurement is that FPA or UCP measures result in abstract numbers, that do not represent individual user requirements. As such the priority (M28) or status (M27) of requirements cannot be quantified. Therefore it is suggested that the exact method for counting requirements (M25), just like the method for quantifying functionality (M8), is decided individually per project.

Nevertheless, estimating costs of change with *functional size measurement* methods is a reasonable option. Or as Jones [Jon96] states “Software outsourcers and contractors can now derive the function-point totals of software during the requirements phase, which means that downstream costs of changing requirements can be quantified”.

[2] Plain text requirements such as “the system shall . . . ”

Resulting measures

Conclusively, in response to the goal of "managing project scope", the following measures are proposed (note; numbering starts where *productivity* and *defect* measures leave off):

Q13. How many requirements are defined in this project?
  M25. Number of functional requirements defined
Q14. How many requirements are fulfilled?
  M26. Ratio requirements allocated vs. requirements implemented per iteration
Q15. What is the status of the requirements?
  M27. Distribution of requirements over status classes (approved, rejected, . . .)
Q16. How are the requirements prioritized?
  M28. Distribution of requirements over priority classes
Q17. How many change requests have been submitted?
  M29. Number of change requests submitted per week
Q18. What is the purpose of change requests?
  M30. Distribution of change requests over type classes (corrective, perfective, . . .)
Q29. What effort is required to implement change requests?
  M31. Cumulative budgeted vs. actual effort spent on change requests, per iteration

Appendix C

Measurement Specification

THIS appendix presents example "formal" specifications of two high priority measure in the measurement programme. In short the internal characteristics, responsibilities, data collection method and reporting procedure of the measure are highlighted.

M6: Budgeted effort planned vs. actual- and remaining effort spent per discipline		
Description:	This measure sets out the *budgeted* amount of effort (for activities belonging to a certain discipline) against the expected amount of effort. Where expected effort is the *actual* amount of effort spent + the amount of effort *remaining.*	
Possible limitations and/or side-effects:	N/A	
Dependencies:	M4 & M5	
Characteristics		
Definitions:	*Discipline*:	A set of similar activities grouped by category (e.g. write test script, conduct regressing tests, discuss findings, etc all belong to discipline: *system testing*)
	Budgeted:	An estimate of the amount of effort needed to complete a set of activities, belonging to a certain discipline.
	Actual:	Amount of effort already invested in activities of a specific discipline (certain number).
	Remaining	Estimate of the amount of effort required to complete the current activities (in progress) of a certain discipline (uncertain number)
	Effort	Amount of man-hours spent working on a specific activity
Formula:	N/A	
Type:	☒ Indirect ☐ Direct	☒ Subjective ☐ Objective
Entity:	Project schedule	
Internal attribute(s):	Effort	
External attribute(s):	Costs, Progress	
Unit/Range:	Man hours	
Responsibilities		
Responsible:	Project manager	
Accountable:	Project manager	
Consulted:	Team leaders	
Informed:	Project group, Project steering group	
Data collection		
Method:	This measure is automatically aggregated from the activity level (figure D.6). So, data collection takes place at the same time as activity progress registration.	
Data source:	Project Planning (MS Project)	
Data storage:	Measurement Support System (Excel)	
Frequency:	Simultaneously with the progress registration of activities	
Reporting		
Type:	(Modified) Stacked bar chart	
Frequency:	Bi-weekly	
Indicator:	Figure D.9	
Interpretation hints:	Note that remaining + actual effort = expected effort	

Figure C.1: *Example measurement specification (M6)*

M15: Number of defects found per iteration	
Description:	The number of open and closed defects found in the current iteration
Possible limitations and/or side-effects:	The number of defect doesn't say anything about the time required to fix defects (M19) or the impact on the project (M21). A high number of open defects at the end of an iteration is not necessarily a bad thing. Furthermore a (sudden) decrease in defect count does not necessarily mean that the product is of high quality, and ready for release. It may well indicate a decrease in the *thoroughness of testing*. Therefore it is suggested to keep the amount of effort spent on testing activities (M6) in mind during data analysis.
Dependencies:	M6, M19, M21
Characteristics	
Definitions:	*Iteration*: A short period of time within a certain project. During this period tasks of several disciplines (e.g. development, testing) are carried out in parallel. This usually results in the release of one or more artifacts.
	Artifact: A product (e.g. vision document, source code, test report, …) developed during as part of a project.
	Open Defect that is not yet complete resolved
	Closed Opposite of *open*
Formula:	N/A
Type:	☐ Indirect ☒ Subjective ☒ Direct ☐ Objective
Entity:	Project artifacts
Internal attribute(s):	Defects
External attribute(s):	Readiness for release, Quality of work performed
Unit/Range:	Natural numbers
Responsibilities	
Responsible:	Test leader
Accountable:	Project manager
Consulted:	Test team, Construction team
Informed:	Project group, Project steering group
Data collection	
Method:	The registration of defects in an issue tracker is a necessary prerequisite of this measure. SMART-Java offers the option for exporting defect information from the issue tracker to CSV (comma separated values) format. This allows one to import defect data in the Measurement Support System with minimal effort.
Data source:	Issue tracker (Mantis / GForge)
Data storage:	Measurement Support System (Excel)
Frequency:	Once at the end of each iteration
Reporting	
Type:	Stacked bar chart
Frequency:	After each iteration
Indicator:	Figure B.5
Interpretation hints:	N/A

Figure C.2: *Example measurement specification (M15)*

Appendix D

Measurement Support System

THIS appendix lists the most important sheets (or "screens") of the *prototype* Measurement Support System (MSS). This system is developed to facilitate data registration and analysis during software development projects. The sheets are written in Dutch to better integrate with existing project (measurement) reports. Note that **data in the sheets is fake** and solely used for demonstration purposes.

Project metrieken administratie v0.2

Project		
Projectnaam:	Foobar	
Duratie		
Begin datum:	6-5-2006	
Verwachte eind datum:	15-9-2006	
Werkelijke eind datum:		
Tijd resterend:	71 dagen	
(tot verwachte/geplande eind datum)		
Omvang		
Product omvang meeteenheid:	(NESMA) Functie punten	
Geschatte product omvang:	500	FP
Gerealiseerde omvang (tot dusver):	183	FP
Inspanning		
Verwachte inspanning (man uren):	2300	
Werkelijke inspanning (man uren):	98	
Productiviteit		
Geschatte aantal FP per uur	0.2	
Werkelijk aantal FP per uur	1.9	
Defects		
Defect index (oplossing efficiency)	0.1	
Defect dichtheid (# defects per FP)	0.2	

Legenda	
	= Kop tekst (read-only)
	= Invoer veld
	= Label veld (read-only)
	= Calculatie veld (read-only)

Figure D.1: *Project level view of the Measurement Support System*

Iteratie (naam)	Gebudgeteerde begin datum	Gebudgeteerde eind datum	Werkelijke/verwachte begin datum	Werkelijke/verwachte eind datum	Verwachte duratie	Werkelijke duratie
Iteratie 1	6-5-2006	1-6-2006	6-5-2006	20-6-2006	26	45
Iteratie 2	2-6-2006	3-7-2006			31	0
Iteratie 3	4-7-2006	10-8-2006			37	0
Iteratie 4	11-8-2006	22-8-2006			11	0
Iteratie 5	23-8-2006	15-9-2006			23	0
					0	0
					0	0
					0	0
					0	0
					0	0
					0	0
					0	0
					0	0
					0	0
					0	0
					0	0
					0	0
					0	0
					0	0
					0	0
					0	0
					0	0
					0	0
					0	0
					0	0
					0	0
					0	0
					0	0
					0	0

Figure D.2: *Iteration planning with effort, duration and size data*

Gebudgeteerde inspanning	Werkelijke inspanning	Nog te verwachten inspanning	Gerealiseerde omvang (FP)	Tijdstip van omvangsm	# Gevonden defects
80	98	56	10	20-5-2006	0
111	0	0	60	7-6-2006	10
103	0	0			17
60	0	0			15
0	0	0			0
0	0	0			0
0	0	0			0
0	0	0			0
0	0	0			0
0	0	0			0
0	0	0			0
0	0	0			0
0	0	0			0
0	0	0			0
0	0	0			0
0	0	0			0
0	0	0			0
0	0	0			0
0	0	0			0
0	0	0			0
0	0	0			0
0	0	0			0
0	0	0			0
0	0	0			0
0	0	0			0
0	0	0			0
0	0	0			0
0	0	0			0
0	0	0			0

Figure D.3: *Iteration planning with effort, duration and size data (cont.)*

Activiteit	Discipline	Iteratie	Gebudgeteerde begin datum	Gebudgeteerde eind datum	Verwachte/werkelijke begin datum
Gesprekken met klant	Requirements	Iteratie 1	6-5-2006	9-5-2006	6-5-2006
Viewpoints opstellen	Architectuur	Iteratie 1	7-5-2006	10-5-2006	10-5-2006
Ontwikkelstraat inrichten	Conf. Management	Iteratie 1	18-5-2006	20-5-2006	15-6-2006
Conceptueel model opstellen	Requirements	Iteratie 1	22-5-2006	23-5-2006	22-5-2006
Presentatie aan klant	Requirements	Iteratie 1	28-5-2006	1-6-2006	29-5-2006
Ontwerp UML klassen diagram	(Detail) Ontwerp	Iteratie 2	3-6-2006	12-6-2006	
Test hypotheek invoer component	Systeem testen	Iteratie 3	13-6-2006	14-6-2006	
Bouw component X	Constructie	Iteratie 2	15-6-2006	18-6-2006	
Test risico berekening component	Systeem testen	Iteratie 3	10-6-2006	20-6-2006	
Bouw component A	Constructie	Iteratie 2	28-6-2006	3-7-2006	
Bouw component Q	Constructie	Iteratie 3	21-7-2006	19-7-2006	
Bouw component Y	Constructie	Iteratie 2	4-6-2006	23-7-2006	
Bouw component Z	Constructie	Iteratie 2	7-6-2006	24-7-2006	
Bouw component C	Constructie	Iteratie 3	23-7-2006	29-7-2006	
Deploy iteratie 2 bij klant	Deployment	Iteratie 3	24-7-2006	29-7-2006	
Architectuur updaten	Architectuur	Iteratie 3	18-7-2006	30-7-2006	
Bouw component B	Constructie	Iteratie 3	7-8-2006	9-8-2006	
Bouw component U	Constructie	Iteratie 3	3-8-2006	18-8-2006	
Test beveiliging	Systeem testen	Iteratie 4	5-8-2006	19-8-2006	
Bouw component W	Constructie	Iteratie 4	2-8-2006	20-8-2006	
Bouw component T	Constructie	Iteratie 4	3-8-2006	21-8-2006	

Figure D.4: *Activity planning with effort, duration and type data*

Verwachte/Werkelijke eind datum	Gebudgeteerde inspanning	Werkelijke inspanning	Nog te verwachten inspanning
12-5-2006	5	8	0
30-5-2006	10	30	6
20-5-2006	15	20	0
6-6-2006	20	20	10
20-6-2006	30	20	40
	28	0	0
	29	0	0
	40	0	0
	22	0	0
	23	0	0
	10	0	0
	10	0	0
	10	0	0
	2	0	0
	10	0	0
	4	0	0
	6	0	0
	20	0	0
	10	0	0
	30	0	0
	20	0	0
		0	0
		0	0
		0	0
		0	0
		0	0
		0	0
		0	0
		0	0
		0	0

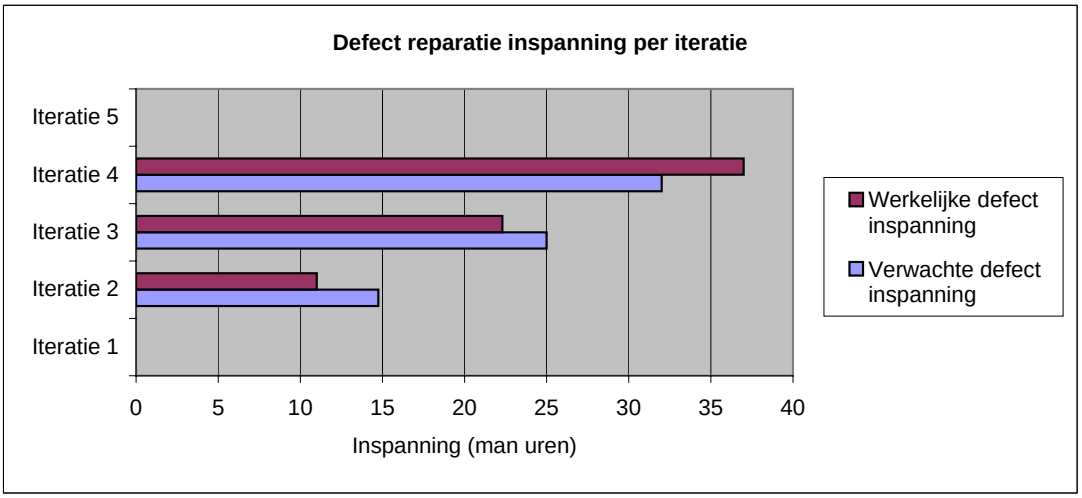
Figure D.5: *Activity planning with effort, duration and type data (cont.)*

Figure D.6: *Periodic progress registration at the activity level*

	Tijdstip		Impact		Inspanning (man uren)		
Defect ID	**Iteratie**	**Discipline**	**Prioriteit**	**Ernst (severity)**	**Gebudgeteerde inspanning**	**Werkelijke inspanning**	**Status**
6237	Iteratie 2	Constructie	Hoog	Major	1	1	Closed
7195	Iteratie 2	Constructie	Laag	Minor	2	1	Closed
7194	Iteratie 3	Constructie	Hoog	Major	0.5	3	Closed
7047	Iteratie 2	(Detail) Ontwerp	Hoog	Block	0.25	1	Closed
7193	Iteratie 2	(Detail) Ontwerp	Hoog	Major	3	1	Closed
7190	Iteratie 4	Conf. Management	Hoog	Minor	4	2	Closed
7191	Iteratie 4	Constructie	Laag	Minor	1	3	Closed
7192	Iteratie 4	Constructie	Hoog	Major	1	1	Closed
7075	Iteratie 3	Constructie	Laag	Minor	2	1	Closed
5482	Iteratie 4	Constructie	Hoog	Minor	10	25	Closed
7189	Iteratie 3	Systeem testen	Hoog	Minor	2	1	Closed
4403	Iteratie 3	Systeem testen	Hoog	Major	3	1	Closed
7188	Iteratie 2	Systeem testen	Laag	Minor	0.5	1	Closed
7187	Iteratie 3	Systeem testen	Laag	Minor	2	2	Closed
7174	Iteratie 3	Systeem testen	Laag	Minor	1	1	Closed
7185	Iteratie 3	Constructie	Laag	Minor	1	1	Closed
7186	Iteratie 3	Constructie	Laag	Major	1	2	Closed
1568	Iteratie 2	Requirements	Laag	Minor	2	1	Closed
4286	Iteratie 3	Constructie	Hoog	Major	1.5	1.3	Closed
7184	Iteratie 4	Constructie	Laag	Minor	1		Open
6844	Iteratie 2	Constructie	Laag	Minor	1		Open
4428	Iteratie 3	Constructie	Laag	Major	1		Open
7183	Iteratie 3	Constructie	Laag	Minor	2		Open
7160	Iteratie 2	Constructie	Laag	Major	3	2	Closed
7182	Iteratie 3	Constructie	Hoog	Minor	1	5	Closed
7181	Iteratie 2	(Detail) Ontwerp	Laag	Minor	1	2	Closed
6769	Iteratie 3	Conf. Management	Hoog	Block	1		Open
7180	Iteratie 3	Constructie	Laag	Minor	2	1	Closed
7179	Iteratie 4	Conf. Management	Laag	Minor	2		Open
7052	Iteratie 2	Constructie	Laag	Minor	1	1	Closed
7178	Iteratie 3	Constructie	Laag	Minor	1		Open
7177	Iteratie 3	Constructie	Laag	Minor	2	2	Closed
7176	Iteratie 4	Constructie	Laag	Minor	2	1	Closed
6534	Iteratie 4	Constructie	Laag	Major	1		Open
7175	Iteratie 4	Conf. Management	Hoog	Block	1	1	Closed
7179	Iteratie 4	Conf. Management	Laag	Minor	2		Open
7052	Iteratie 3	Systeem testen	Laag	Minor	1	1	Closed
7178	Iteratie 4	Systeem testen	Laag	Minor	1		Open
7177	Iteratie 4	Acceptatie testen	Laag	Minor	2	2	Closed
7176	Iteratie 4	Acceptatie testen	Laag	Minor	2	1	Closed
6534	Iteratie 4	Systeem testen	Laag	Major	1		Open
7175	Iteratie 4	Systeem testen	Hoog	Block	1	1	Closed

Figure D.7: *Defect tracking sheet with time, impact and effort data*

Productiviteit rapportage



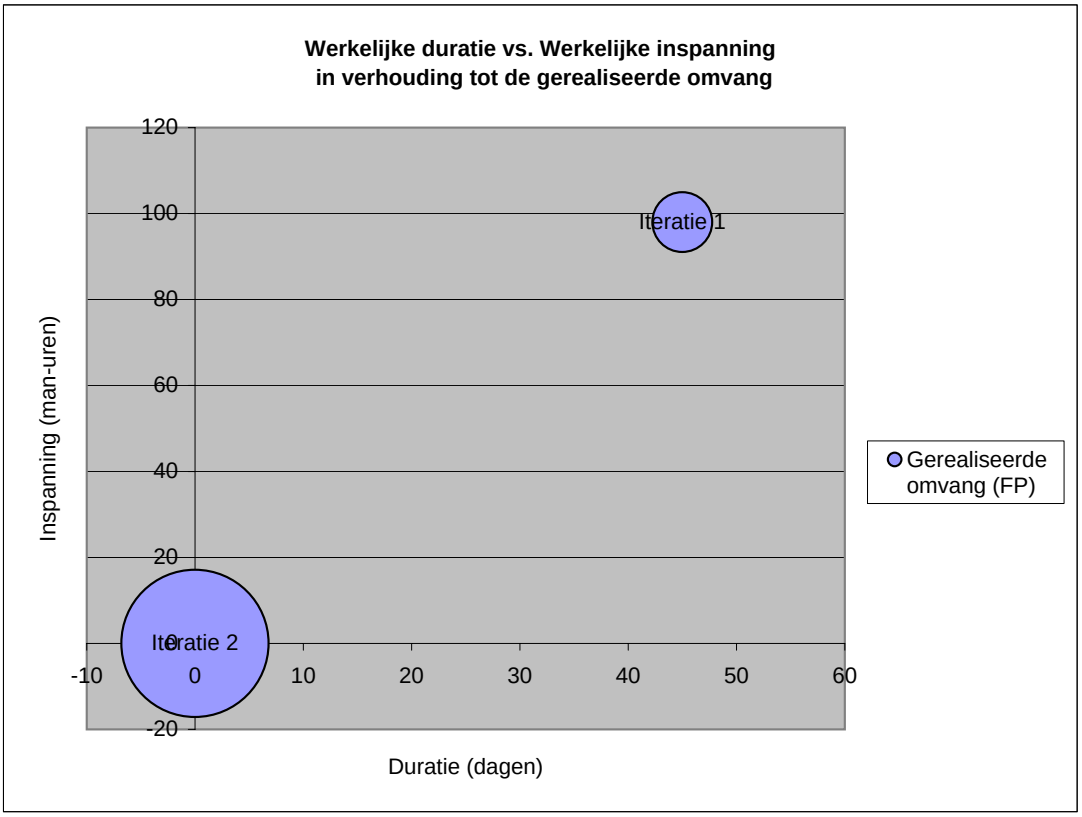
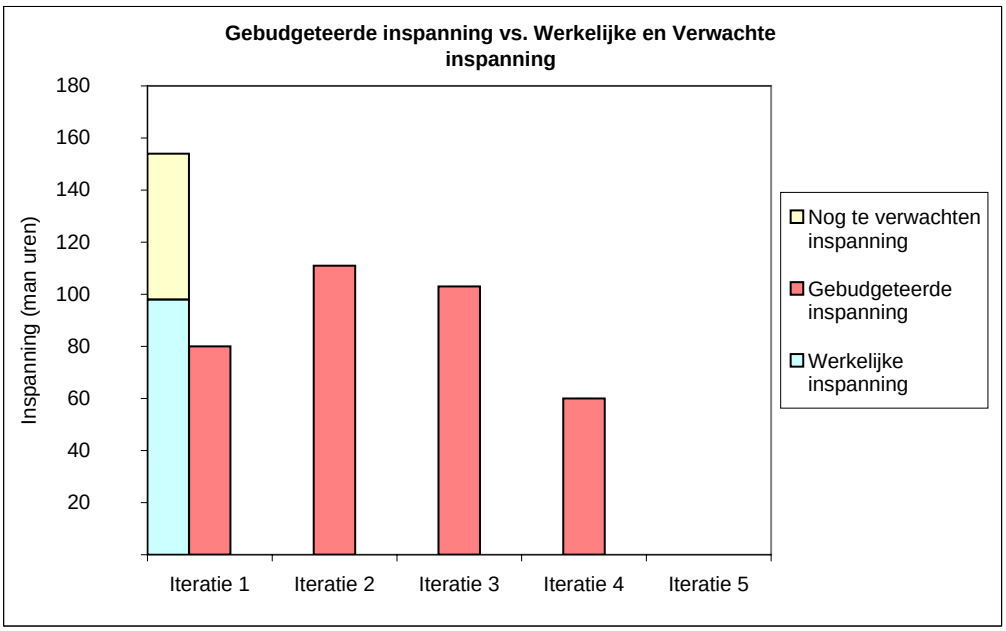


Figure D.8: *Productivity reporting based on progress and costs (showing M19 & M13)*



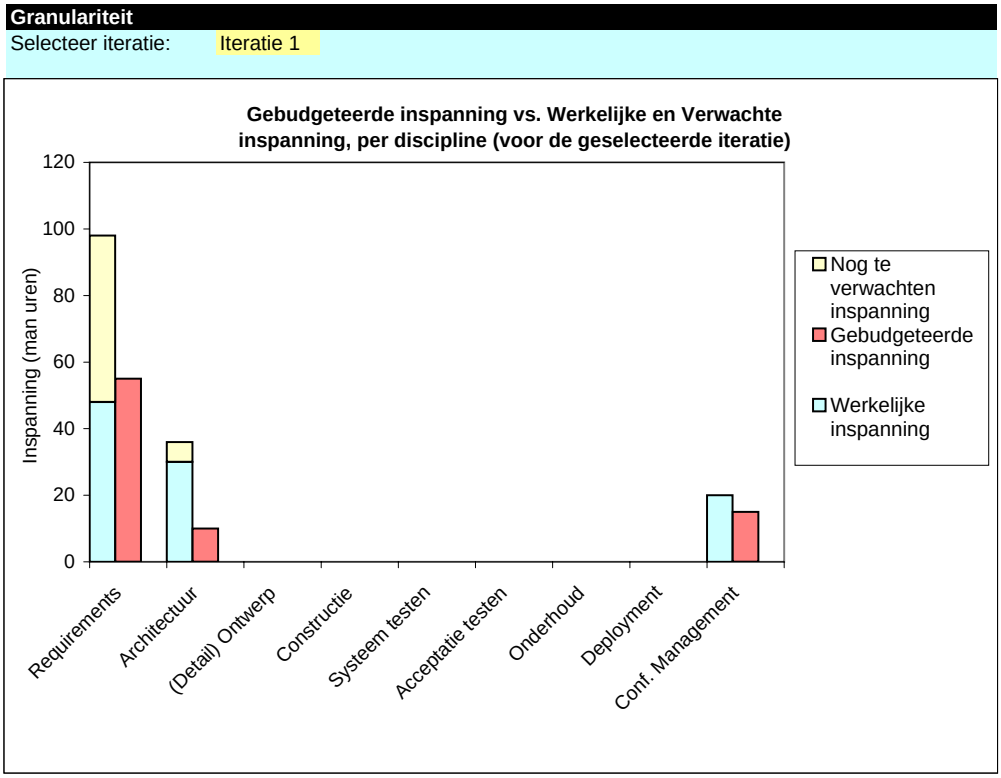


Figure D.9: *Productivity reporting based on progress and costs (showing M5 & M6)*

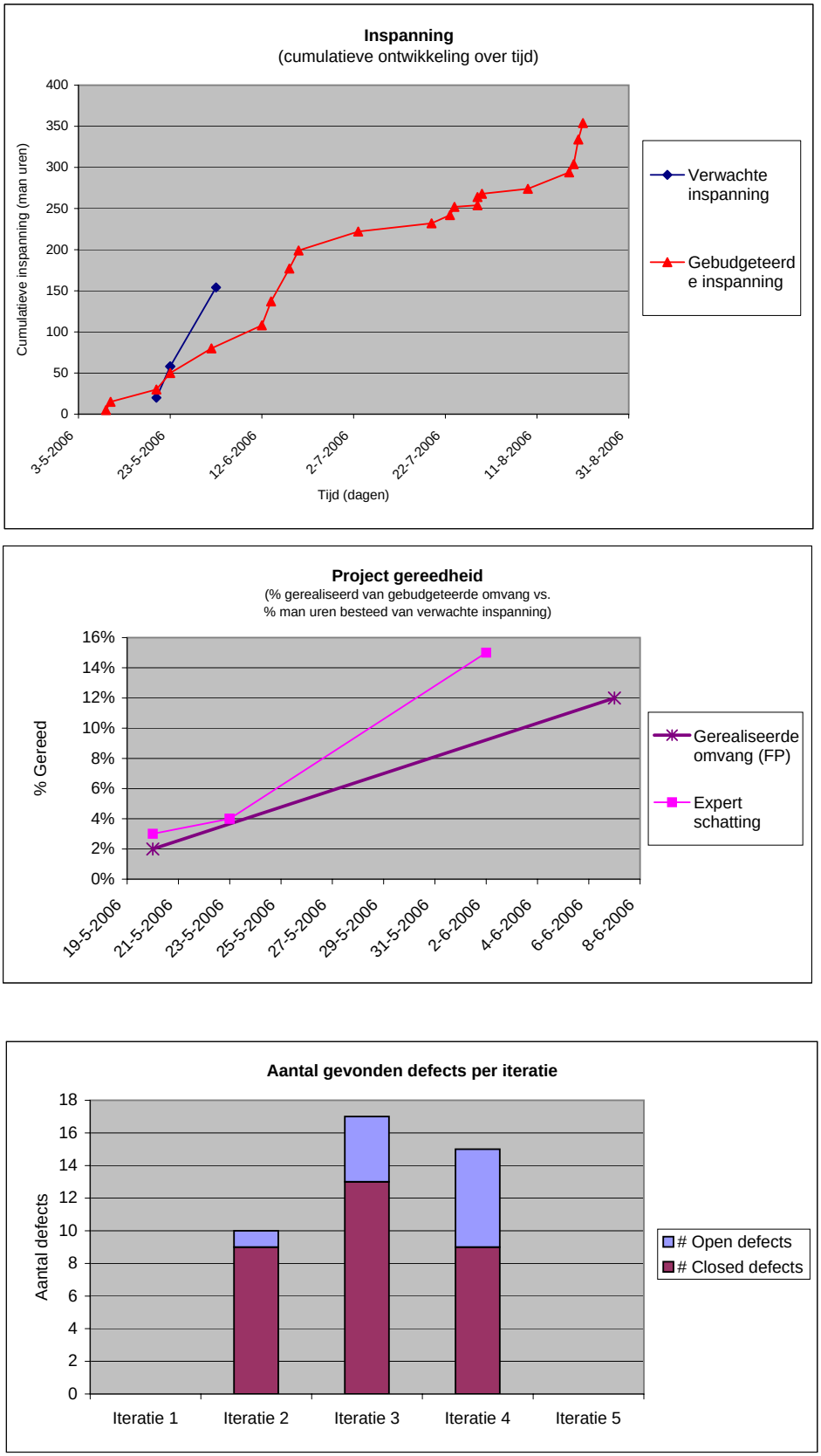


Figure D.10: *Productivity reporting based on progress and costs (showing M4, M9 & M15)*

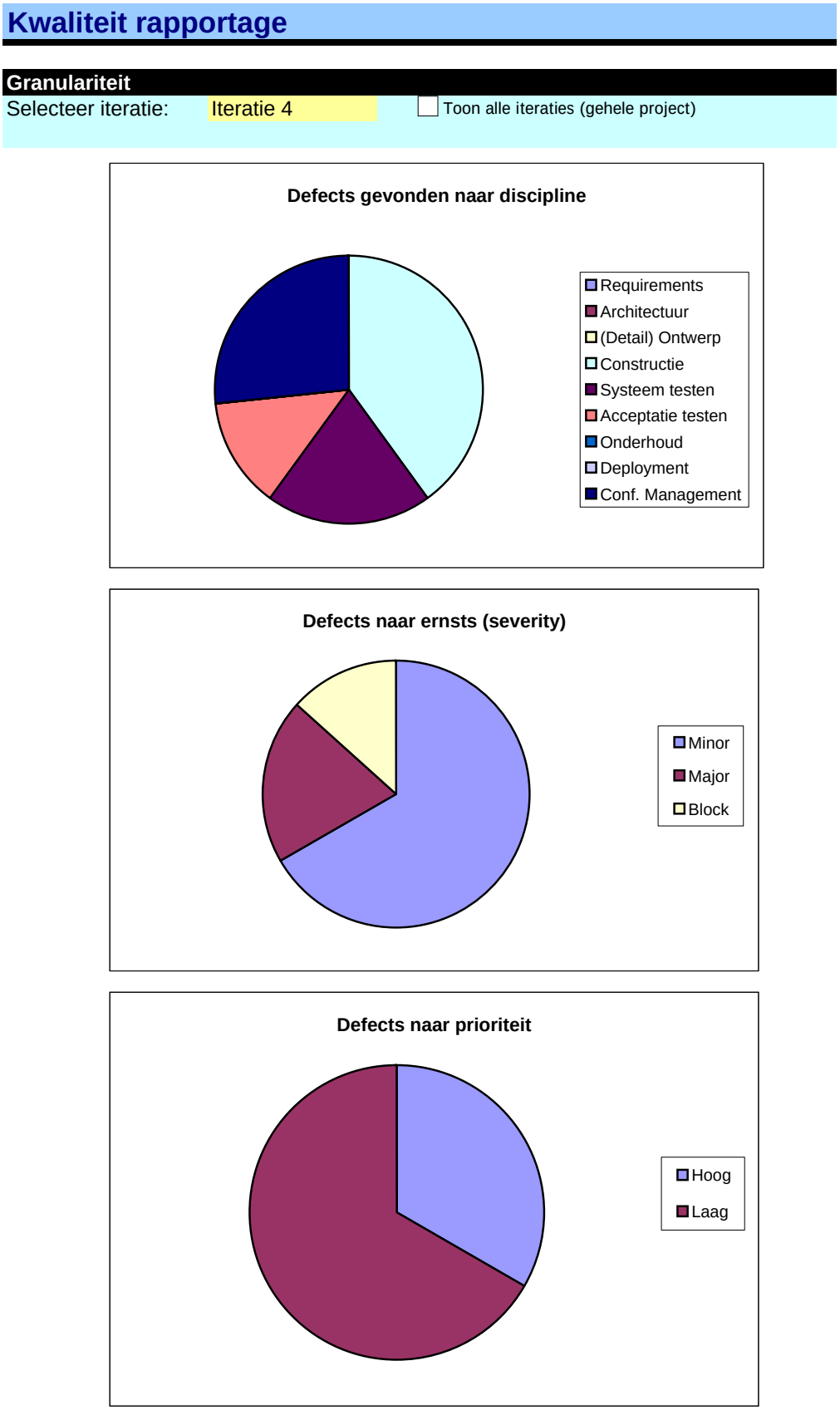


Figure D.11: *Quality reporting based on defect information (showing M17, M21 & M22)*

Appendix E

Overview of Measurement-CMM

NIESSINK and Van Vliet proposed a capability maturity model (in [Nie98]), specifically aimed at software *measurement* programmes. This maturity model, known as M-CMM, is not affiliated with CMM(I) [Can06] nor endorsed by the Software Engineering Institute (SEI). Nevertheless, it provides a useful basis for *reasoning* about the state of measurement practice within an organization. An excerpt of the five maturity levels in M-CMM [Nie98], and associating Key Process Areas, is provided below. This thesis is primarily focused on the *measurement design* KPA.

1. **Initial**: The organization has no defined measurement processes, few measures are gathered, measurement that takes place is solely the result of actions of individuals.

   - No key process areas.

2. **Repeatable**: Basic measurement processes are in place to establish measurement goals, specify measures and measurement protocols, collect and analyse the measures and provide feedback to software engineers and management. The necessary measurement discipline is present to consistently obtain measures.

   ⇒ *Measurement Design*: Measurement goals, measures and measurement protocols are established according to a documented procedure, and goals, measures and protocols are kept consistent with each other. Measurement protocols are managed and controlled.

   - *Measure Collection*: Measures are collected according to the measurement protocol.
   - *Measure Analysis*: Collected measures are analyzed with respect to measurement goals.
   - *Measure Feedback*: The measurement goals, the measurement protocols, the collected measures and the results of the analysis are made available to the people involved in the measurement process.

3. **Defined**: The measurement process is documented, standardized, and integrated in the standard software process of the organization. All projects use a tailored version of the organization's standard measurement process.

   - *Organization Measurement Focus*: Software measurement activities are coordinated across the organization. Strengths and weaknesses of the measurement process are identified and related to the standard measurement process.
   - *Organization Measurement Design*: A standard measurement process for the organization is developed and maintained and information with respect to the use of the standard measurement process is collected, reviewed and made available.
   - *Organization Measure Database*: Collected measures are stored in an organization-wide database and made available.
   - *Training Program*: People are provided with the skills and knowledge needed to perform their roles.

4. **Managed**: The measurement process is quantitatively understood. The costs in terms of effort and money are known. Measurement processes are efficient.

   - *Measurement Cost Management*: The costs of measurement are known and used to guide the Measurement Design Process and the Organization Measurement Design process.
   - *Technology Selection*: The information of measurement costs is used to choose and evaluate technology support for the measurement process.

5. **Optimizing**: Measurements are constantly monitored with respect to their effectiveness and changed where necessary. Measurement goals are set in anticipation of changes in the organization or the environment of the organization.

   - *Measurement Change Management*: The measurement capability is constantly being improved by monitoring the measurement processes and by anticipating changes in the software process or its environment.

Appendix F

"Basic level" measurement

THE measures listed below (and discussed in chapter 6) are *suggested* for *initial implementation* in pilot projects. However, it is important to note that a measurement programme is typically subjected to change[1]. Although this appendix lays down a number of basic measures, it is *by no means complete*.

Productivity

M3. Budgeted iteration duration vs. actual and remaining iteration duration

M4. Cumulative budgeted effort planned vs. expected effort spent

M5. Budgeted effort planned vs. actual- and remaining effort spent *per iteration*

M6. Budgeted effort planned vs. actual- and remaining effort spent *per discipline*

M7. Product size divided by the amount of effort *planned* vs.
Product size divided by the amount of effort *spent*

M8. Estimated vs. actual product size

M9. Percentage of budgeted product size implemented vs. Expert effort estimate

M13. Actual duration vs. Actual effort spent in relation to the
Actual product size per iteration

Defects

M15. Number of defects found and fixed per iteration

M17. Defect detection time by discipline

M19. Budgeted vs. actual effort spent on defect fixes per iteration

M21. Defects by severity

M22. Defects by priority

[1] Stakeholder needs are expected to change, based on the information resulting from the programme. Therefore new questions will arise, and (existing) measures have to be refined and implemented.