LAPPEENRANTA UNIVERSITY OF TECHNOLOGY
Department of Information Technology

SOFTWARE PROCESS IMPROVEMENT USING AN ELECTRONIC PROCESS GUIDE

The subject of this thesis was approved by the council of the Department of Information Technology on the 21st of August, 2001.

Supervisor: Professor Jan Voracek

Instructor: M. Sc. (eng.) Keijo Marttinen

Lappeenranta, November 19, 2001

Markus Mannio
Leirikatu 2 G 2
53600 Lappeenranta
Tel. +358 (0)40 7277 794

ABSTRACT

Author: Markus Mannio

Subject: **Software process improvement using an electronic process guide**

Department: Information technology

Year: 2001

Place: Lappeenranta

Master's thesis. Lappeenranta University of Technology. 72 pages, 8 figures, 2 tables, and 3 appendices.

Supervisor: Professor Jan Voracek

Keywords: software process, software process improvement, process modelling, process guide

Software processes and their improvement have received considerable attention in the recent years due to an increasing interest in software quality. Software process improvement models, such as CMM and SPICE, are being introduced to software companies worldwide in the quest for better software. At the same time it has been realised that effective process improvement and performance require a description of the process to enable thorough process understanding and accurate communication. Various ways for describing a software process exist to serve different purposes. A process guide is a representation of a process focused on process understanding and communication. An electronic process guide is a process guide taking advantage of the possibilities offered by web technologies.

In this work an environment for developing electronic process guides for supporting software process improvement and performance is developed. The environment enables the modelling, customising, and instantiating of a software process as a process guide. The environment is validated by modeling the software process of a telecommunications software company and creating electronic process guides to support the process improvement and execution activities. Finally, the support and possibilities offered by the process guides in the target company are explored and discussed.

TIIVISTELMÄ

Tekijä: Markus Mannio

Nimi: Ohjelmistoprosessin kehittäminen sähköisen prosessioppaan avulla

Osasto: Tietotekniikan osasto

Vuosi: 2001

Paikka: Lappeenranta

Diplomityö. Lappeenrannan teknillinen korkeakoulu. 72 sivua, 8 kuvaa, 2 taulukkoa ja 3 liitettä.

Tarkastaja: Professori Jan Voracek

Hakusanat: ohjelmistoprosessi, ohjelmistoprosessin kehittäminen, prosessin mallintaminen, prosessiopas

Kasvava kiinnostus ohjelmistojen laatua kohtaan on herättänyt ohjelmistoprosesseihin ja niiden kehittämiseen kohdistuvaa huomiota viime vuosina. Ohjelmistoyritykset ympäri maailmaa ovat ottaneet käyttöön ohjelmistoprosessin kehittämismalleja, kuten CMM ja SPICE, pyrkiessään kohti parempilaatuisia ohjelmistotuotteita. Samalla on huomattu, että tehokas prosessien parantaminen ja suorittaminen tarvitsee tuekseen kuvauksen prosessista, jotta prosessin perusteellinen ymmärtäminen ja kommunikointi olisi mahdollista. Ohjelmistoprosesseja voidaan kuvata monilla eri tavoilla. Prosessiopas on prosessin esitysmuoto, jonka päätarkoituksena on helpottaa prosessin ymmärtämistä ja kommunikointia. Elektroninen prosessiopas on Web-teknologiaa hyödyntävä prosessiopas.

Tässä työssä luodaan kehitysympäristö elektronisille prosessioppaille, joiden tarkoituksena on tukea ohjelmistoprosessin kehittämistä ja suorittamista. Ympäristö mahdollistaa ohjelmistoprosessin mallintamisen sekä yksilöllisten oppaiden luomisen ja muokkaamisen. Kehitysympäristöä käytetään mallintamaan tietoliikenneohjelmistoja valmistavan yrityksen ohjelmistoprosessia sekä luomaan elektronisia prosessioppaita tukemaan prosessin kehitystä ja suorittamista. Lopuksi pohditaan prosessioppaiden tarjoamaa tukea sekä mahdollisuuksia kohdeyrityksessä.

PREFACE

This thesis was made in Lappeenranta for the Department of Information Technology of Lappeenranta University of Technology and the work was carried out in Intellitel Communications Ltd.

I would like to thank my instructor Keijo Marttinen and Ilkka Toivanen for invaluable suggestions, guidance, and support throughout this work. Also, thanks are extended to the supervisor of this work Professor Jan Voracek.

My sincerest appreciation goes to my parents for offering me the chance to do what I want, and to Suvi for bearing me while I have been doing it. Very special thanks are reserved for Joonas and Einari for providing endless entertainment and food for refreshing thoughts. And while I'm at it, I must thank Caro and Urho as well.
I only wish that they could read this.

Lappeenranta, November 19, 2001

Markus Mannio

Contents

ABBREVIATIONS

AD	Activity Diagram
APPL/A	Ada Process Programming Language
CMM	Capability Maturity Model
CPI	Continuous Process Improvement
EPG	Electronic Process Guide
HTML	Hypertext Markup Language
IDEF0	Integration Definition Language 0
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
KPA	Key Process Area
MVP-L	Multi-View Process modeling Language
OMG	Object Management Group
PAL	Process Asset Library
PML	Process Modelling Language
PSEE	Process-centered Software Engineering Environment
SADT	Structured Analysis and Design Technique
SEI	Software Engineering Institute
SPI	Software Process Improvement
SPICE	Software Process Improvement and Capability Determination
SPIE	Software Process Improvement Engineering
SSD	Static Structural Diagram

TCM	Toolkit for Conceptual Modeling
UML	Unified Modeling Language
UP	Unified Process
UPM	Unified Process Model

1 INTRODUCTION

1.1 Background

During the last couple of decades software has become commonplace in the daily lives of many people. Whether they acknowledge it or not, many people are in contact with software or machines controlled with software almost constantly. This not only includes the obvious, such as computers or mobile phones, but also the more inconspicuous home electronic and many common services. Groceries paid with a credit card update the databases of a credit company and modern washing machines, coffee-makers, or cars include embedded software more often than not. Since software has become an integral part of these services and devices, their performance and functionalities are affected by the characteristics of the underlying software. The overall quality of the service or product, whether it is a computer or a washing machine, is affected by the quality of the software. Whereas quality has been a focus for traditional engineering and manufacturing from the 1930s, the quality of software has received attention only from the 1980s.

Much of the current interest towards software quality is based on the assumption that the quality of the software is highly dependent on the quality of the process used to develop the software - the *software process* [1, 2, 3]. That is, the quality of a software product is heavily dependent on the people, organisation, and procedures used to create and deliver it [1]. Consequently, the assumption has lead to the notion that improving the software process should result in improved software. Unfortunately, software products and processes are seen as complex entities whose successful improvement in a software developing organisations seem to require considerable effort [1, 2, 3, 4]. From an organisational viewpoint *software process improvement* (SPI) needs supporting organisational structures, financial investment, cultural environment, and top management support [2]. These aspects constitute the environment where successful SPI is possible, and the absence of any of these elements is likely to cause the failure of an SPI effort [2]. From an individual viewpoint the software developers need information about the methods, techniques, and tools to carry out the software process. Above all, software process improvement seems to require precise definition and description of the process itself [1, 4, 5, 6]. Indeed, most of the widely accepted software quality and improvement models, such

as *capability maturity model* (CMM) and *software process improvement and capability determination* (SPICE), introduce explicit process description as one of the most important factors supporting process improvement [4] to function as the baseline for process changes required by SPI.

Representations of a software process based on an explicit process definition can be used in a software developing organisation also for numerous other purposes apart from SPI, such as vehicles for process understanding and communication, and support for process execution and management [1, 7, 9, 8]. Various formalisms for creating process definitions and representations have been suggested by researchers to aid in presenting complex software processes in a comprehensive way, but unfortunately most of these notations or *process modelling languages* (PML) have proven to be too complex to be adopted in practise [1]. Practitioners' primary concerns seem to be understanding and communicating the processes, and consequently process representations should be intuitive and easy to use [1]. Software engineers need to understand and communicate the process they are performing as well as the process engineers need to understand and communicate the process they are improving. The need for understandable representations of relatively loosely-defined and frequently changing processes has recently been acknowledged by researchers [1, 9], and approaches for supporting process technology have been suggested [9, 10, 11].

Electronic process guides (EPG) are web-based software process representations that are focused on communicating the software process in a manner that can be easily understood and followed by software engineers [9]. In addition to providing information to the software engineers, an EPG can also be used as a supporting tool for SPI efforts. Web technologies enable the use of EPG as a vehicle for collecting, storing, and analysing process related information which could provide the process engineers invaluable information about the software process. Although the development of EPGs for offering guidance to process performers has been presented [9, 10], the development and use of EPGs for supporting SPI has not been widely discussed. In this work the creation and use of EPGs for supporting software process improvement and performance to aid in improving software quality is studied.

1.2 Scope and objectives

The main objective of this thesis was to develop an electronic process guide environment for processes where representations of the software process can be created. A process guide developed within the environment was to support software process improvement, increase process understanding, and facilitate communication within and between different user groups. Software and process engineers were selected as the primary user groups for the process guides, and the primary requirements for the guides were understandability and the ease of their use. The framework was validated by modelling the software process of a telecommunications software company and exploring how the electronic process guides based on the model could support the software improvement and development activities in the company. The underlying general hypothesis was that a properly designed process guide can be a valuable aid in *all* phases of software process improvement.

1.3 Structure of the work

The rest of the thesis is organised in the following way.

Chapter 2 focuses on the software process and software process improvement. Software process and process improvement context, concepts, and definitions are presented.

Chapter 3 describes how the software process presented in Chapter 1 can be defined and represented. The topics of this chapter include the conceptual framework, and the different methodologies and languages for representing software processes. Especially process guides as the means to represent processes are discussed.

Chapter 4 describes the structure and development of an electronic process guide framework. This chapter is closely related to Chapter 3 and the software process concepts presented in Chapter 2.

Chapter 5 discusses how the EPGs developed in the framework support the software process improvement and execution activities in a telecommunications software company.

The process improvement part of Chapter 2 and Chapter 4 as a whole form the background for this chapter.

In *Chapter 6* the EPG environment presented in Chapter 4 and its possible uses described in Chapter 5 are discussed and conclusions are also made.

Finally, *Chapter 7* presents a brief summary of the thesis.

2 THE SOFTWARE PROCESS

Various definitions for the software process have been suggested [1, 3, 5]. Commonly a process is understood, as defined in the Cambridge International Dictionary of English [12]

> a series of actions or events that are part of a system or a continuing development, or a series of actions that are done to achieve a particular result

In software development context the particular result mentioned in the definition can be thought to be, for example, the software product fulfilling the requirements set to it, or more broadly a satisfied user or customer. Fuggetta [1] defines the software process as

> the coherent set of policies, organisational structures, technologies, procedures, and artifacts that are needed to conceive, develop, deploy, and maintain a software product

A software process with the above definition or the general process definition in software development context with the mentioned end results, especially in the broader sense, can not exist without supporting infrastructure [3]. Fuggetta [1] mentions four concepts and contributions that the software process exploits:

1. software development technology,
2. software development methods, techniques, and guidelines,
3. organisational behaviour, and
4. marketing and economy.

Although this work concentrates on software process definition and representations to aid in SPI and process execution, it should be acknowledged that these issues cover only one

part of effective utilisation of process technology in software developing organisation. A number of organisational, cultural, technological, and economic factors needed to support the software process [1] are out of the scope of this work. These issues are covered more extensively in software process literature, for example in [3].

In this chapter the basic concepts of software processes and their improvement are discussed. First, the basic software process concepts and context are explained in more detail. Next, general software process improvement concepts are presented. Finally, a view to some of the more common practical software process improvement and assessment models is given.

2.1 Software process concepts

In this subchapter some essential software process concepts are discussed. The concepts are described in a framework for software process and software development presented in Figure 1. The square boxes in the figure represent objects or artifacts, while the rounded boxes show the activities. The figure encompasses all the activities from process engineering to the results of executing the software, as can be seen by looking at the uppermost and lowest boxes in the figure. Next the different phases in software process and development are discussed in more detail.

2.1.1 Process engineering

Process engineering is the activity of performing a process engineering process [6]. The uppermost box in Figure 1 represents the process engineering process i.e. the description of how processes are engineered. The second box shows the actual execution of the process engineering process, and thus it represents process engineering. In process terminology the execution of a process according to a process definition is often called *enactment* [13], which is also used in this work from here on. Processes are enacted by agents that are entities, such as individuals, groups, or machines [6, 13]; agents as performers of process activities are discussed more thoroughly with process modelling in Chapter 3. The

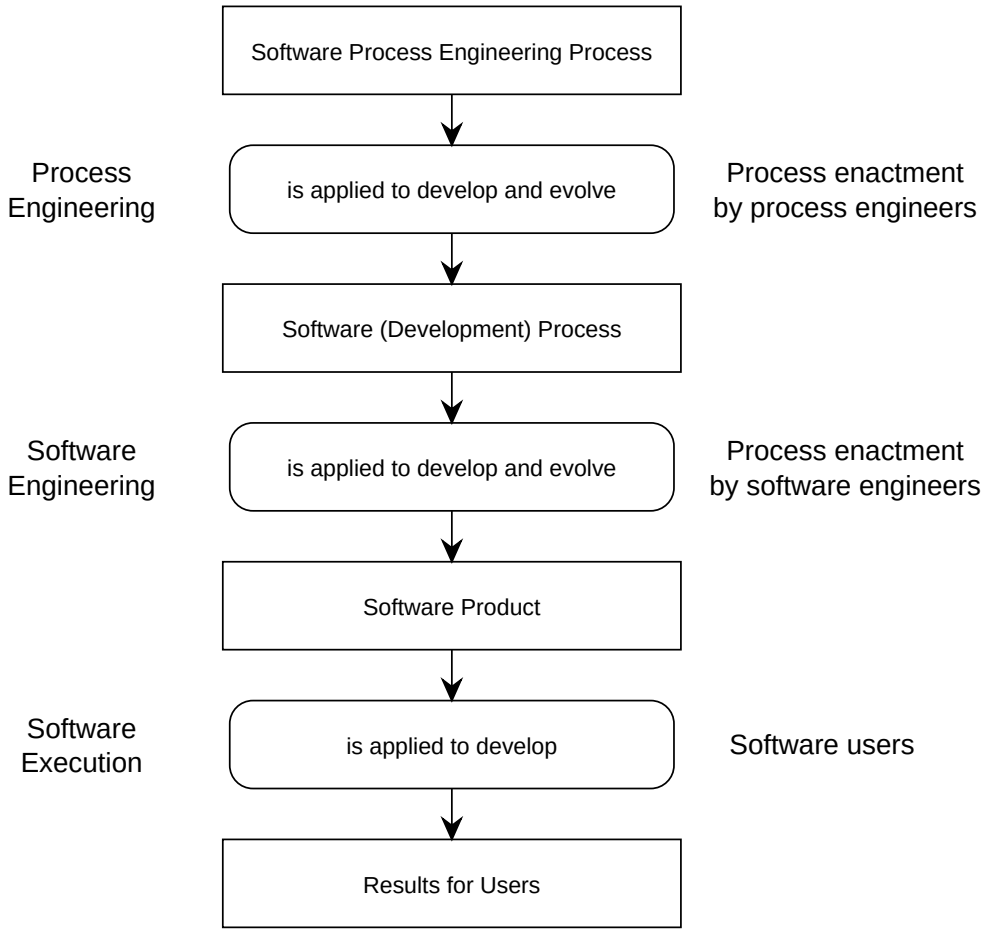


Figure 1: The Process Framework [6].

process engineering process definition in the first box is a meta-process, because it is a process operating on processes. Conceptually more abstract levels of processes are not needed because the meta-process can be enacted to develop and evolve itself [6]. This could be presented in Figure 1 as a loop involving the first, process engineering process, and the second, process engineering, box.

In the context of developing software, the uppermost box represents the definition of the process of developing a software process. In the second box, representing process engineering, the process shown in the uppermost box is enacted and as a result a software process is created as shown in the third box. Thus, in software process engineering activities the three uppermost boxes are involved. The actual software process engineering, in the second box, is usually performed by process engineers or quality engineers. Generally, process engineering is performed by the people responsible for the creation and improvement of software process definitions and models in software developing organisations.

Software development and process development enjoy numerous similarities. For example, process development can be described in software engineering terms, thus including activities like planning, design, instantiation, and validation [6]. It has even been suggested that software processes are a form of software themselves and that process engineering methods could be borrowed from software development [5]. Process engineering is not just an isolated activity in software development, but it is also highly dependent of other aspects of the software developing organisation. For example, a cultural change and supporting infrastructure are needed to switch from the more traditional product-focused organisation to a more process-focused one [3].

2.1.2 Software engineering

Software engineering is a technology consisting of a process, a set of methods, and tools to build computer software [14]. Software engineering activities encompass the three midmost boxes in Figure 1. The first box represents the result of the previous software process engineering activities, the *software process*. In the next box the software process is enacted by software engineers with the help of appropriate methods and tools, and as

a result a computer software product is obtained as shown in the following square box. Although the software engineers are shown in the figure “only” as the performers of a defined process, it is generally acknowledged that building a software requires creativity that the process discipline should encourage rather than stifle [3].

Finally, the three lowest boxes in Figure 1 describe a software user using the software to accomplish something of value for himself.

2.1.3 Process definition

The two uppermost square boxes in Figure 1 represent process definitions. Feiler and Humphrey [13] define process definition as

> an implementation of process design in the form of a partially ordered set of process steps that is enactable.

They also present the notions that a process definition may consist of (sub)process definitions, and at a lower level of abstraction each process step may be further refined into more detailed process steps. Process definitions whose “levels of abstraction are fully refined” are referred as complete and fit for enactment. An important observation is pointed out about the completeness of process definitions, which is considered to be dependent on the context and performers of the process. This also supports the idea of Kellner et al. [6] that processes should be engineered to meet specific goals and objectives, subject to the real-world constraints that apply. Current skill levels of existing staff and limits on cycle time are presented as examples of such constraints. Although there is no clear technical limit to the level of refinement for a software process, Feiler and Humphrey [13] list a number of practical concerns that affect the limit of refinement including

- resources and time available for process definition,
- level of capability or understanding of the process users,

- scale of the projects for which the process is designed, and
- scalability of the process itself.

In practise, finding the right level of abstraction for process definitions is difficult since definitions refined to excessive detail often result in too complex process representations while too general definitions can be completely unusable [15].

Scalability of the process, mentioned above, is closely connected to another important concept, namely the reuse of software processes. Clearly it is not reasonable to define completely new processes for each software development project from nothing, especially when software process development can be very expensive and time consuming [13]. It is likely that in software developing organisations different projects will have similar needs and common activities, but it is also highly probable that some of the needs of different development projects are very different. Feiler and Humphrey [13] address this problem of shared process definitions by suggesting the development and use of a set of general purpose, reusable process elements. As in software engineering a library of reusable components can be used, also in process engineering a library of reusable process components can be seen as a valuable asset for an organisation. Such *process asset library* (PAL) is discussed, for example, in [6], [8], and [16]. If suitable general processes are available an enactable process can be instantiated from their definitions. Thus, instantiation in process terminology is the act of creating enactable processes, including all the elements required for enactment, from process definitions [13].

2.1.4 The process life cycle

Above the basic concepts of software processes were presented. Now a description, which ties the previously concepts together is described.

The description presented in Figure 2, the process cycle, is originally presented by Madhavji [17] and further explained, for example, in [8]. The process cycle can in effect be thought as a view to a process life cycle containing four parts: description and definition, customisation and instantiation, enactment, and improvement. The process cycle

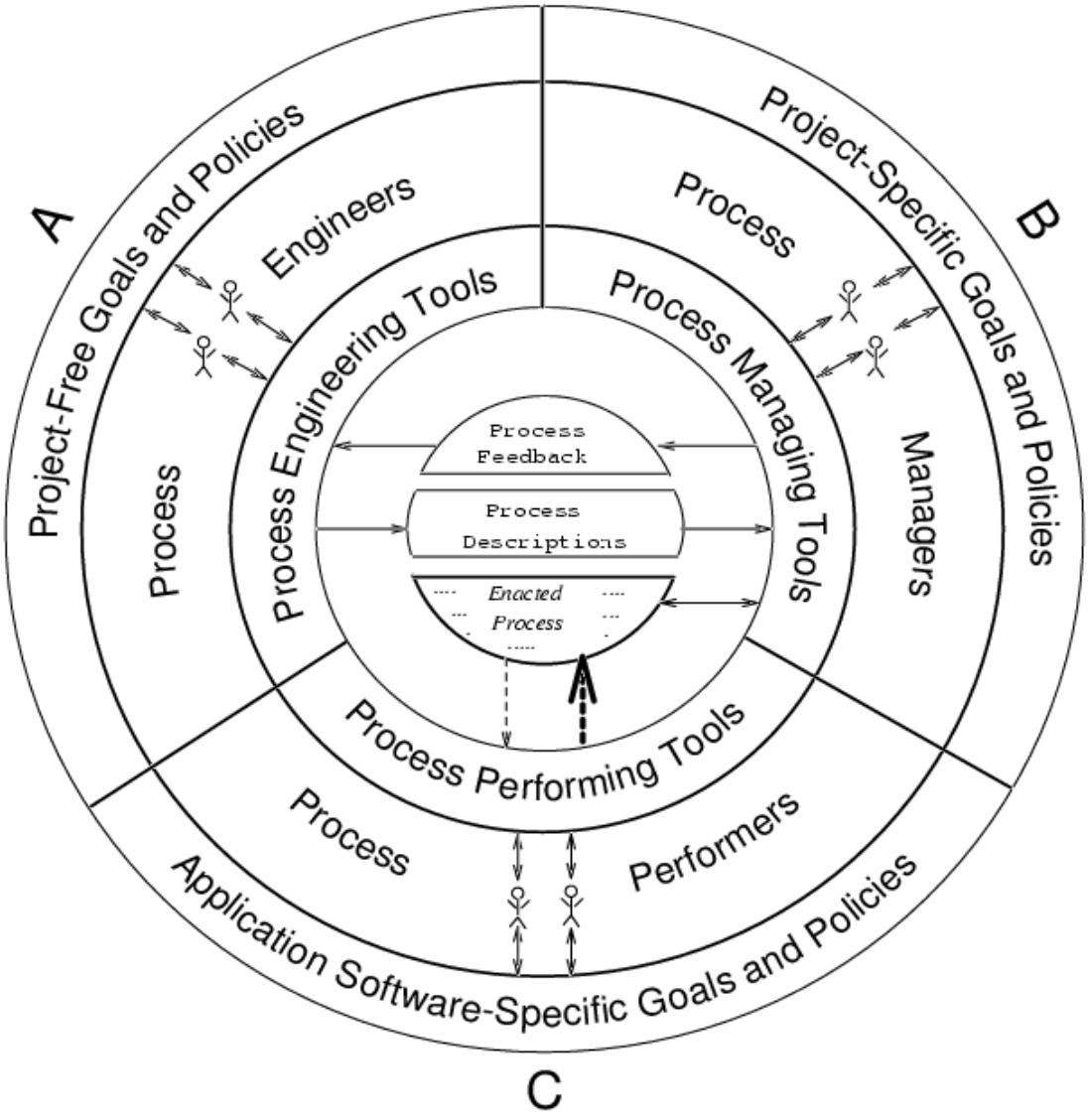


Figure 2: The Process Cycle [8].

describes the relationships between the key roles, tools, and goals in different parts of a process life cycle. The cycle also includes an important link between process and software engineering, namely *process management*.

The outer sections of the cycle are divided into three sectors, which represent the engineering, managing, and enactment of software processes. Each of the three sectors has its own goals and policies, key roles, and tools. The center of the cycle describes the primary items transferred between the different sectors of the cycle. Sector *A* encompasses the main process engineering activities where process engineers design, construct, and improve generic process definitions. These definitions are not tailored to any specific project, and they can be stored to a PAL. Sector *B* introduces the process management aspect where process managers are acting in a central role. According to Figure 2 process managers tailor the generic process definitions to a specific use, and thus instantiate processes. The instantiated processes are enacted in sector *C* by process performers with application specific goals and policies. In the context of software development presented in Figure 1 process engineering contains sectors *A* and *B*, and software engineering is included in sector *C*.

In the center of the cycle the main items moving between the different sectors are described. The process definitions are created and updated in sector *A*, from where they are transferred to sector *B* for instantiation. The instantiated processes are enacted in sector *C*, while process managers receive process feedback from the process performers. This feedback might reflect, for example, the smoothness with which assigned tasks are carried out, bottlenecks in the process, negative effects caused by timing constraints, or the need for additional process steps or removal of superfluous ones [8]. Process managers may make changes to the project-specific process instances, or suggest changes to the process engineers maintaining the more general process definitions in sector *A*.

2.2 Software process improvement

In the previous section the basic concepts of software process engineering were presented. Also, a description of software process lifecycle glueing the process concepts together

was introduced. In this subchapter the focus is shifted towards improving the software process.

A large number of different models and guidelines for improving the software process have been introduced in the last decade [3, 6, 18]. In fact, even the process cycle described earlier in Figure 2 contains, among other things, an inherent mechanism for software improvement through process feedback and experience. Although there seems to be a plethora of different software process improvement paradigms to choose from, most of the more widely accepted models are based upon the same ideological background. Many of the current SPI efforts are built upon the work begun by quality researchers over six decades ago: *Walter Shewhart* presented the "plan, do, check, act" cycle for quality improvement in the 30's, which *Edwards Deming* among others further developed and demonstrated [3]. Adaptations of the quality movements cyclical improvement model to processes are seen in process management literature in the concept of *continuous process improvement* (CPI). One of the most influential persons in process improvement in software development has been *Watts Humphrey*, who adapted the lessons learnt in the quality movement for software development [3].

An example of CPI in the context of software development is presented in [6] as the basic cycle of the helical model of *software process improvement engineering* (SPIE), which is shown in Figure 3. The basic concepts presented in the helical model are generally also found in any other cyclical process improvement model (e.g. [18]), and the basic cycle of the helical model is used here as an example to clarify these concepts in the next couple of subsections. To illustrate this point references to Humphrey's process improvement cycle, reviewed in [3], are also made.

2.2.1 Evaluate process

Although the cycle in Figure 3 can be entered at any phase, the natural way to begin improving a process is to try to understand the current process. If a process model does not exist, the current process can be modeled in this phase to aid in understanding as described in Chapter 3. Thus, modelling the process at this phase is *descriptive modelling*, which in effect tries to describe the current process as it is performed [7]. Also, all available in-

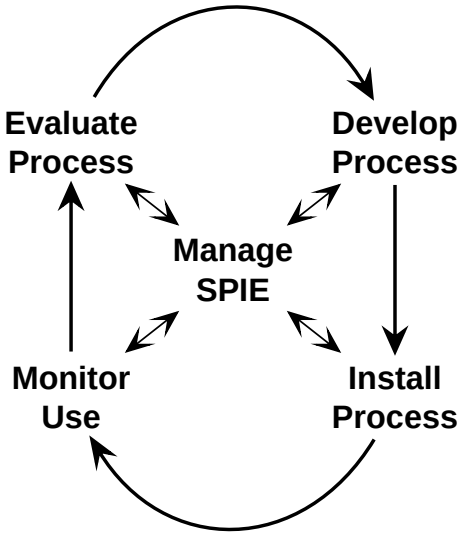


Figure 3: Helical Model – The Basic Cycle [6].

formation about the process should be used to evaluate the process. Both qualitative and quantitative analysis should be used: the information may be qualitative and subjective at the beginning, but should shift towards more quantitative information as the process improvement activities proceed. Measurement of the quantitative information, such as defect counts and cycle times, should be included as part of the improved process. Other sources of process information include process performers, as shown in the process cycle in Figure 2, other projects, and literature [6]. This phase corresponds with the first phase of Humphrey's six step process improvement cycle consisting of understanding the current process [3].

2.2.2 Develop process

In this phase the actual improved process is developed and changes to the process are applied. According to [6], activities performed in this phase are:

- determine goals and constraints for the process,

- plan process development,
- identify and understand process alternatives,
- analyse and choose process alternatives,
- define the improved process,
- reduce risks, and
- document and package the process definition materials.

Some of the steps are somewhat self-explanatory whereas other activities may need clarification. The activities focusing on the process alternatives include identifying, understanding, and choosing the reusable process assets that may be stored in a project asset library. Also, alternative approaches for the process development are considered and a decision is made based on the goals, constraints, and qualitative and quantitative considerations of the process. In defining the process, modelling and the development of a process guide are considered as an appropriate approaches. At this phase the modelling activities include mainly *prescriptive modelling* in which – as opposed to descriptive modelling mentioned in previous phase – the process is modeled as it *should* be performed [7].

Reducing risks includes activities, such as verification and validation of the defined process. In the last step, the results of the previous phases are documented according the next intended use of the materials. The final result may be a detailed model, a high-level model, a process guide, or training material depending on the purpose of the process improvement [6]. In comparison with Humphrey's cycle, this phase of the helical cycle includes the three following steps of Humphrey's cycle: developing a vision of the desired process, establishing a list of required process improvement actions, and producing a plan to accomplish the required actions [3].

2.2.3 Install process

In this part of the cycle presented in Figure 3 the infrastructure for supporting the enactment of the process is established, and the installation of the process to projects is planned.

Process infrastructure provides operational support for the process activities and management consisting of the organisational and managerial roles and responsibilities as well as technical facilities [3]. Infrastructure-related activities in this phase include assigning staff, and training the intended process performers in the new process and in the supporting methods and tools [6]. Installing the process to a project requires careful planning and possibly tailoring of the process from the more general process definitions. Referring to the process cycle in Figure 2, the development activities in the previous phase are mainly located in sector *A*, whereas the installing activities are performed by the process managers in sector *B* of the cycle. The fifth step in Humphrey's six step cycle is about implementing the process improvement plan [3], which is partly realised in the previous phase of the helical cycle and partly in this phase.

2.2.4 Monitor process use

This part of the helical cycle concentrates on monitoring the enactment of the process. Some basic usage issues to monitor are whether the process is understood and followed, and what problems arise during its use [6]. Also, it should be ensured that the measurements specified in the process definition are taken and that any variations of the process are identified and documented. The process performers should be assisted in usage, and improvement suggestions and other feedback should be recorded. This phase is conceptually included in the first step of Humphrey's software improvement cycle where the current status of the development process is evaluated [3].

2.2.5 SPI Management

As any large-scale project, SPI activities require management activities for coordination and control. In the helical cycle the management part is placed in the center of the main cycle to illustrate the fact that it is needed in all the other phases. In the center of the Figure 3, the term SPIE is used with the definition of the activities of software process engineering that are specifically related to SPI activities [6]. The management activities are executed concurrently with the other phases of the cycle. Kellner et al. [6] suggest

that each major SPIE undertaking should be managed like a project. Project management is a large research area in itself and is out of the scope of this work – for those interested in project management a wide array of literature is available.

2.2.6 Process maturity and capability

The main assumption behind the cyclical software improvement models is the notion that while the phases of a model are executed in cyclical fashion, the software process gradually improves i.e. gets better. As a consequence, the software process is said to become more *mature*. The maturity of a process can be measured, for example, using one of the reference models presented in the next subchapter. Process capability is another concept closely related to process maturity. Whereas process maturity is a characteristic that is difficult to measure objectively without a reference model, process capability can be measured more easily via the results if a process. Zahran [3] defines process capability as

> The range of expected results that can be achieved by following a process

Zahran [3] also presents an assumption that the higher the level of the process maturity of an organisation, the higher its process capability.

2.3 Software process improvement and assessment models

In the previous subchapter generic steps for cyclical software process improvement were discussed. In this subchapter a brief overview to some widely accepted practical models focusing on software process improvement and assessment is presented. Any of the models will fit in the more general CPI steps presented in Figure 2. Although the models have different architecture and focus, all of them can be and are used in software development organisations as the basis of designing and improving their software processes. Process improvement models focus on how to carry out the process of improving a process, while

process assessment concentrates on the determination of the degree of maturity of a process with respect to a quality model [1]. Since the assessment of the current state of the software process is an integral part of any software process improvement and through process assessment the software process can be improved, a strict line cannot be drawn between the two types of models.

2.3.1 Capability maturity model

The *capability maturity model* (CMM) was developed by the *Software Engineering Institute* (SEI) based on the work of Watts Humphrey [3]. A widely accepted definition of CMM presented, for example, by Zahran [3] is

> Capability Maturity Model: A description of the stages through which software organizations evolve as they define, implement, measure, control and improve their software processes.

The CMM is focused on process assessment and it can be used in an organisation to determine the capabilities of their current process and identify the most critical aspects of the process for software process improvement [3]. CMM was initially focused towards the software process but after the generic nature of the model was fully discovered, several variations of the model were developed. CMM for software (SW-CMM) is the model concentrating on the software process.

The CMM includes five levels of process maturity, which define the scale for measuring an organisation's software process and the process' capability [3]. Each of the five levels, except the first *Initial Level*, is composed of several *key process areas* (KPA), which in turn consist of key practises organised into five sections called common features [19]. The KPAs are unique among the maturity levels and include the common areas in software development, such as *Requirements Management*. Each of the KPAs has its own goals, which are accomplished through addressing the common features of the KPA by following the key practises related to the feature.

2.3.2 Software process improvement and capability determination

The *software process improvement and capability determination* (SPICE) is a project launched in 1993 to assist in developing a new *International Organization for Standardization* (ISO) and *International Electrotechnical Commission* (IEC) standard for software process improvement and assessment. The goals of the SPICE project were set to aid in the standardisation process in developing a full international standard ISO/IEC 15504, to conduct industrial trials of the emerging standard, and to promote the standard to create market awareness in the software industry [20].

Structurally, in contrast to one-dimensional CMM, the proposed ISO/IEC 15504 contains a two-dimensional reference model for describing processes and process capabilities. The process dimension of the reference model defines 40 processes and groups them into three life cycle process groupings which contain five process categories, according to the type of activity they address. The objectives of each process are described in terms of a purpose statement. The reference model does not define how, or in what order, the elements of the process purpose statements are to be achieved nor the detailed tasks and activities related to the processes. The other dimension is the process capability dimension which is characterized by a series of measurable process attributes that are applicable to any process. The capability dimension consists of six capability levels which provide a way of progressing through improvement of the capability of any process [20].

In addition to the reference model, the ISO/IEC 15504 contains guides to performing an assessment, qualification of assessors, and process improvement. The guide for process improvement describes how to define the inputs to and use the results of an assessment for the purposes of process improvement. It also includes examples of the process improvement in a variety of situations [3].

Varkoi and Mäkinen [19] present a comparison between CMM and SPICE in software process assessment. Although no simple and general way was found to conclude that certain SPICE capabilities correspond to CMM maturity levels, it is concluded that at large the same information can be acquired with both models to support process improvement purposes.

2.3.3 BOOTSTRAP

The BOOTSTRAP methodology is a result of a European Community project consisting of the development of the BOOTSTRAP model and method, and industry trials. Although the project finished in 1993, the methodology has been further developed and marketed by the BOOTSTRAP institute. BOOTSTRAP contains a software process assessment method and provides also guidelines for transforming the assessment results into an action plan and prioritising the actions. The assessment method is three-dimensional containing organisational, methodological, and technological dimensions. The capability scale of the assessment method is based on the CMMs five-level maturity scale with some modifications [3].

3 PROCESS REPRESENTATION

A process representation is a description, depiction, likeness, or portrayal of a process [9]. A process without a representation is difficult to perform, control, or improve. If the process is represented only by an image in the minds of the people performing the process, one can argue if the process even exists. A process representation captures the aspects of a process definition that are relevant to the a particular task [13], which may be, for example, improving or enacting a software process. The nature of the task implies the users and the requirements, which in turn affect the structure and composition of the representation. As a consequence, a representation of a process for process improvement purposes may differ notably from a representation of the same process for some other purpose. Three key modes of process representations, primarily distinguished by form and usage, have been suggested: process models, process templates and forms, and process guides [9]. Although the modes may superficially differ notably from each other, they enjoy similarities in the basic conceptual framework and process definition.

In relation to the concepts presented in the previous chapter, process representations are created and updated in sector *A* of the process cycle in Figure 2, and respectively in the evaluation and development phases of the software process improvement cycle in Figure 3. Also, the representations may be modified in the instantiation of the process in sector *B* of the process cycle, and in the installation phase of the helical software process improvement cycle.

In the rest of this chapter a survey into the basics of any software process representation is made, and different types of representations for different purposes are described. First, the common conceptual framework underlying a software process representation is described. Second, the three different modes of process representations are described. Finally, different *process modelling languages* (PML) and methodologies for implementing the conceptual framework for different purposes are investigated.

3.1 Conceptual framework

In this part a general conceptual framework for software process representations is described. The framework is focused upon the information that is needed for human enactment of the process and it is described in natural language – the languages and methodologies for practical implementations are described in the next subchapters. The intention of the framework is to describe the types of information that should be in a process definition or representation. The framework is originally presented in [21] and further discussed also in [9]. Integrating measurement to a conceptual framework of a model is discussed in more detail in [22]. A process, its conceptual framework, and representation are presented in Figure 4, which is adapted from [21]. The figure consists of three parts and their interconnections, which are labeled on the right side of the image. The uppermost part describes the actual process, which is transformed into a more abstract conceptual schema, which in turn is described as a representation of the original process. A schema can be thought as an implementation of the conceptual framework. Next, the different aspects of the conceptual framework are explained in more detail.

3.1.1 Entities

Typical considerations about a software process are:

1. what happens and how it is done,
2. what things are used and produced,
3. who does it, and
4. when it is done.

These four basic questions are also presented at the uppermost part of Figure 4 where they are connected to the actual process. From the first three of these questions, three principal entity classes or abstractions involved in software processes can be identified [21].

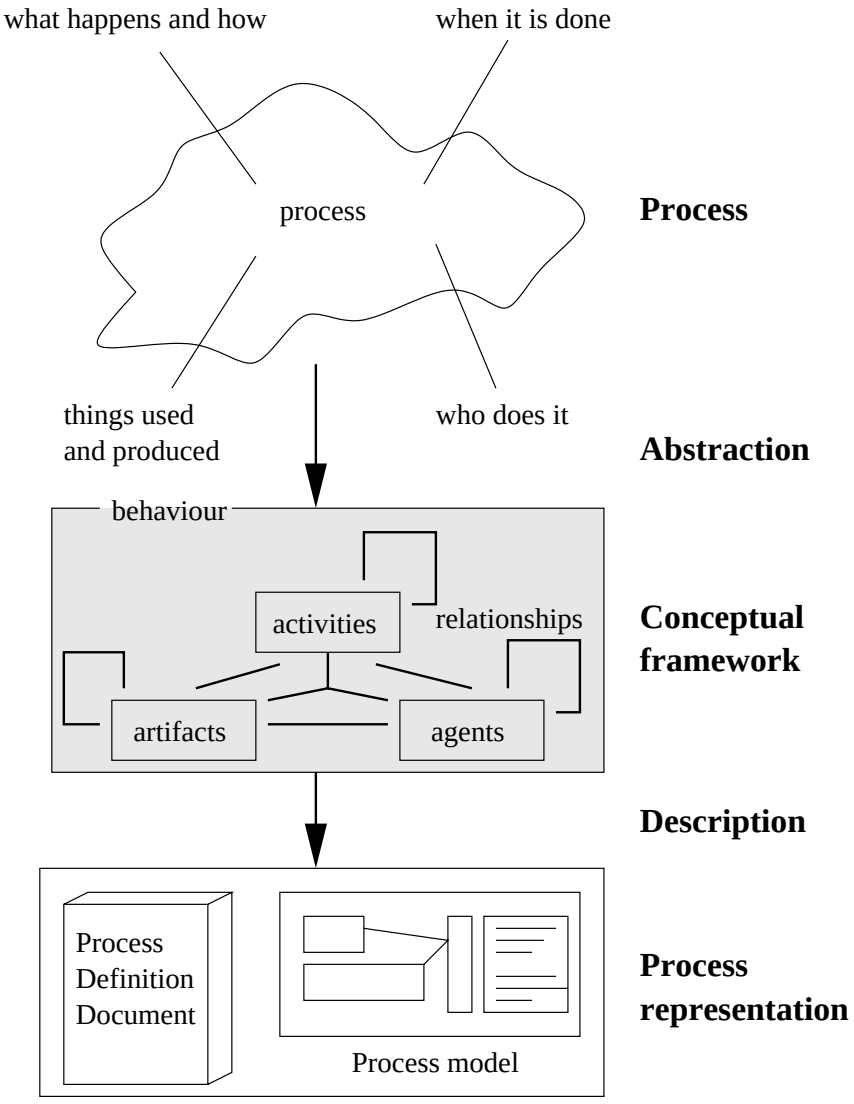


Figure 4: Conceptual framework for software process.

Activities what and how is done

Artifacts things used and produced (by activities)

Agents who or what does it (i.e., performs the activity)

The entities are shown as boxes in the midmost part of Figure 4. Even though an activity describes both the "what" and the "how" in a process, it is often convenient to separate the two information. The "how" part can be described in a separate *procedure*, which is a separate description within an activity [21]. The tools of the process can be considered to be a part of the procedure but also a separate entity can be used to represent them as suggested, for example, in [1].

The three basic classes are common for all kinds of processes. Further classes can be added to a schema to enable a better fit for a particular use, for example by creating subclasses from the principal classes. The agent class, for instance, can be subclassed to a functional agent (commonly called a role) and organisational agent classes [9]. A process usually involves many instances of each of the encompassed entity classes, which comprises the actual data in an instantiated process.

3.1.2 Relationships

Entity classes by themselves are not enough to form a complete process representation. The interactions between and within the classes, and the behaviour of the entities over time also have to be defined [21]. The lines in the middle part of Figure 4 represent the relationships of the basic entity classes. The relationships within a class are shown in Figure 4 as lines whose both ends are in the same entity. Examples of relationships within the basic entity classes include

- activity may comprise of activities, and
- agent manages another agent.

The relationships among the basic entity classes are shown in the figure as lines between the boxes. Examples of relationships among the basic entity classes include

- activities are performed by agents, and
- agents own artifacts.

The entities with their constraints, entity hierarchy, and the relationships within and among the entity classes make up the static part of a software process representation [23]. Thus, the static part gives the description of the elements or entities that are included in a representation.

3.1.3 Behavioural information

Since processes are essentially dynamic in nature, their behaviour over time is a critical aspect [21]. The static part of a process does not take time into account and consequently it ignores the behavioural aspect of the entities. The behavioural aspect of a process is described in the dynamic part of a process representation, which describes the ordering of the tasks to be executed during the process enactment [23]. The dynamic part provides the answer to the "when" question in the uppermost part of Figure 4, and it is also shown in the midmost part as the grayed area affecting all the entities. Examples of the behavioural information of the dynamic part, presented in [21], include

- when (under which circumstances) an activity can begin,
- when (under which circumstances) the state of an artifact is to be changed, and
- when (under which circumstances) alternative paths are to be taken.

Two distinctive paradigms for the control of the dynamic part of a process exist: proactive and reactive. Reactive paradigm is event driven by triggers or exceptions, while proactive is mainly based on precedence relationships [24]. Thus, reactive control is concerned

with performing actions in response to some events, while proactive control is established according to pre-made rules. According to [23], four kind of precedence relationships can be distinguished:

Strong precedences: An activity can begun only when another activity has finished.

Weak precedences: An activity (*a*) can begin after another activity (*b*) has begun and *b* must finish before *a* finishes.

Start synchronising precedences: An activity must begin before another activity is begun.

End synchronising precedences: An activity must finish before another activity finishes.

As it can be seen, weak precedences can be constructed by forming a combination of the synchronising precedences.

3.2 Representation types

Representations of a software process may be developed for many different purposes. One process may have numerous differing representations for different uses in an software developing organisation. Process representations are developed for particular users, user needs and requirements, and usage scenarios. Their intended use affects what they contain and how they are structured [9]. For example, the helical software improvement model in Figure 3 refers to process representations or models in various phases. Some possible purposes for the representations listed in [1], [9], [8], and [7] include

- increase process understanding in organisation,
- facilitate communication about the process,
- aid in training and education of personnel,
- offer support for process enactment,

- facilitate work tracking by capturing information,
- offer support for software process improvement,
- aid in designing a new process,
- aid in finding tasks to automate,
- aid in process simulation and optimisation, and
- support process management.

In this subchapter three types of process representations are described: process models, process templates and forms, and process guides. Each of the three types may serve different purposes for different users.

3.2.1 Process models

A process model is a relatively detailed, formal or semiformal representation of a process [9]. The primary users of a process model are the process engineers or quality engineers, who are responsible for process engineering activities. Process model can be represented in textual or graphical form depending on the intended use of the model. Traditionally process models have been modeled by using a detailed and formal PML. Their main purpose has been automating or formalising a process for machine assisted enactment and serving as a basis for process engineering [9]. Process modelling can be descriptive or prescriptive depending on whether the process is modeled *as it is* or *as it should be*. Becker et al. [7] present an approach for descriptive modelling based on practical experience comprising the following eight steps:

1. state objectives and scope,
2. select or develop a process modelling schema,
3. select (a set of) process modelling languages,

4. select and tailor tools,
5. elicit process knowledge,
6. create model,
7. analyse the process model, and
8. analyse the process.

First four of the steps are proposed to be executed only infrequently while the remaining four are suggested to be executed every time process modelling is performed. In the first step the objectives of the modelling are uncovered based on the intended use of the model. Next, the modelling schema is designed – for example by using the conceptual framework described in the previous subchapter. In the third and fourth step the PMLs, and the supporting tools are selected. The tool can be as simple as a text-editor or drawing tool depending on the type of the PML. The next two steps concentrate on the actual information gathering and modelling the process according the selected schema and using the selected tools. The last two steps include the detection of inconsistencies in the process model and monitoring the enactment of the process for qualitative or quantitative analysis. The approach is also consistent with the modelling meta-process used in the industrial case presented in [4]. Prescriptive modelling can also be performed using similar steps to those above with minor modifications. Usually, in SPI, descriptive modelling is performed first to work as the baseline for prescriptive modeling activities, as suggested in [6].

Many of the existing PMLs are complex, detailed, extremely sophisticated, and strongly oriented towards detailed modelling of processes. These models are often fine-grained, comprehensive descriptions of well-defined, relatively static processes [9]. However, recently it has been suggested that these highly complex, detailed, and static models may not be what is needed by the software developers [1, 9]. Practitioners are not using the complex PMLs and the models created with them because the languages do not respond to their needs. Although the detailed and complex modelling of a process is justified by the desire to be precise and to provide enough information to enable process automation, the practitioners' most important need is often to describe processes with the purpose of communicating and understanding them. This may hinder the adoption of PMLs in practise by creating significant barriers for their acceptance [1]. Sutton and Osterweil [25]

present the balancing of the technical aspects and ease of use of the modelling languages as a key issue for the adoption of these PMLs.

3.2.2 Process templates and forms

Process templates and forms are structured textual process representations where the information is organised into predefined slots, and often arranged in a hierarchical fashion. They are intended to support process engineers in organising, recording, and reporting process information. Templates may serve as interview guides and as a vehicle for recording elicited information when gathering information through interviews during descriptive process modelling. Templates and forms have also been used as a medium for reporting process information in standard format to process participants [9].

3.2.3 Process guides

A process guide is intended to describe a particular process for the main purpose of supporting human enactment of that process. As a consequence, process participants are the primary intended users of process guides. A process guide is a structured, work-flow oriented reference document for a particular process with the main purpose of supporting process participants in carrying out the intended process. Thus, a process guide should be easy to understand, communicate, and follow. Additionally a process guide can also support process training, planning, and certification. Process guides generally include both text and graphics, and contain selected portions of an underlying process definition or model. But as a contrast to process models, process guides should contain much more information than simply a formatted process model [9].

Software development processes are so complex entities that the performers need to be provided with process knowledge to perform their tasks adequately [10]. Process documentation exists in many organisations in the form of paper manuals and process handbooks, which can be thought as paper process guides. However, process performers are frequently dissatisfied with this documentation and in many cases it is simply not used.

The format and content of paper process guides in software developing organisations are often deficient and fail to provide the necessary information in suitable format [9]. Even if a paper process guide contains all necessary information, some serious limitations are still present. Kellner et al. [9] present suggestions for the content and form of paper process guides and also point out numerous problems related to both their development and usability. Problems with paper process guides presented in [9] and also in [10] and [11] include navigation, searching, maintaining, customising, distributing, long update cycles, information repetition, managing dynamic content, and version control. Some of these problems can be solved by using formal process modelling languages to develop process documentation, but still other problems and challenges remain [9].

Electronic process guides are web-based process guides that have been suggested to solve many of the problems related to paper process guides mentioned above [9, 10, 11]. As organisation intranets have become more common also process documentation has been made available through a web-browser. Kellner et al. [9] present three different commonly used methods for moving from paper guides to electronic ones: documents can be made available for download from intranet in their native format, a straightforward conversion of documents from native format into some more browser-friendly language (e.g. HTML [26]) can be made, or a more extensive conversion can be made so that cross-references within a document are converted to hyperlinks. Any of these methods will aid in solving some of the issues with paper guides by facilitating the maintenance and distribution of process documentation. However, an EPG should provide more extensive usage of web-based technology than just links to process documentation. Some of the requirements presented for EPGs include a solid conceptual framework, adequate information content, easy and effective navigation, and clear and informative user interface [9, 10].

Additional possibilities for EPGs are presented in [11] in the form of storage of process state information, online access to templates and manuals, and links to other resources on other web sites. Although just making the paper documentation available through intranet does not qualify as an EPG in itself, it can still be a good starting point for creating a full scale EPG. An incremental approach for developing a process model or an EPG is considered to be a good way to introduce process technology into an organisation and thus help the practitioners in adjusting to the resulting change [1, 10]. Fuggetta [1] suggests that the process representation may be incomplete, informal, and partial at the

beginning and if needed, the representation can be enriched and made more formal at later stages.

Becker-Kornstaedt et al. [11, 27, 28] discuss a modelling tool and an EPG generator, which allows the generation of EPGs directly from the process models. They suggest that the automated generation of an EPG from the models is a fast, accurate, and inexpensive way to develop and update an EPG.

3.3 Process modelling languages and paradigms

Process modelling languages are languages or formalisms used to express software process models, and consequently other process representations based on the models. Numerous different PMLs have been introduced to software process engineering but no single PML has acquired a dominant position in modelling software processes [24, 25, 29]. Various reasons for not having a single standard language for modelling software processes have been presented, such as excessive strictness, and lacking formality, modularity, human comprehensibility, and reuse abilities [24, 30]. Also, non-linguistic aspects, such as organisational, methodological, and technological support have been presented as obstacles for the widespread acceptance and use of a single language [25]. On the other hand it may be possible, considering the multitude of reasons processes are modeled for, that simply no single PML can exist satisfying all the needs for a process representation. For example, some PMLs are more suitable to high-level modelling while others are better in low-level, fine-grained modelling [30]. Zamli and Lee [29] present five characteristics for PMLs:

Modelling support: A PML must be able to represent the basic conceptual elements of a process discussed in the first part of this chapter. Also, the communication mechanisms between activities and roles and different levels of abstraction should be supported.

Enactment support: Two aspects of enactment support that may be affected by a PML are presented. Enactment support in distributed environment is an issue concerned

with physically distributed environments and support across organisational borders. Enactment support for incompletely specified process models allows the incremental construction of models and enactment of incomplete software processes.

Evaluation support: A PML should support process measurement by providing relevant software metrics.

Evolution support: Support for evolving software processes is divided into two categories. Meta-process support is the support for the process of developing a process that was discussed in Chapter 1 and visualised in Figure 1. The second category, process evolution support, should enable to application of changes to the process.

Human dimension support: Categories for human dimension support presented by [29] include support for visual notation, user awareness, process awareness, and process visualisation. Process visualisation is described as the possibility to have multiple views of the same process from different perspectives.

In contrast to PMLs, which are used to describe process models, a number of *process-centered software engineering environments* (PSEE) exist that support the creation and exploitation of software process models. Common features offered by PSEEs include support for definition, modification, analysis, and enactment of process models.

In this subchapter a brief description of some of more common PMLs and modelling paradigms is presented. Since number of surveys have been published presenting and comparing PMLs [29], only a brief description is given with the focus on the languages and approaches that have been used in practise for modelling software processes.

3.3.1 Multi-View Process modeling Language

Multi-View Process modeling Language (MVP-L) is a language developed for building descriptive models of large, real-world processes and their use for understanding, analysing, guiding, and improving software development projects. MVP-L was primarily designed for modelling in-the-large (i.e. high level) to support process understanding. Even though the language is rule-based and textual, some graphical tools for viewing the

models also exist [30]. The models in the integrated modelling and EPG creation tool, presented in [11, 28], are based on an extended MVP-L.

3.3.2 Ada Process-Programming Language and JIL

The *Ada Process-Programming Language* (APPL/A) is based on the *Ada* programming language with extensions to process modelling. Thus, it is a textual modelling language and modelling the processes with APPL/A is in fact *process programming*. JIL is a formally defined, executable process programming language, which can be though as a successor to the APPL/A [29]. The main goals for the design of JIL included semantic richness, ease of use, and support for visualisation. JIL offers a variety of semantic attributes and process control mechanisms and is based on the Ada programming language [25].

3.3.3 Integration Definition language 0

Integration DEFinition language 0 (IDEF0) is based on the *Structured Analysis and Design Technique* (SADT). IDEF0 includes a graphical modelling language and a methodology for developing models. IDEF0 is a coherent and simple language designed to enhance communication between analysts, developers, and users. In addition to the modelling language, IDEF0 also includes a methodology prescribing procedures and techniques for developing and interpreting models [31]. Although the graphical notation used in IDEF0 lacks the possibility to express some of the behavioural aspects of a process, text can attached to the models to compensate these shortcomings.

3.3.4 Unified Modeling Language

The *Object Management Groups* (OMG) *Unified Modeling Language* (UML) is a graphical language originally designed for visualising, specifying, constructing, and documenting the artifacts of a software-intensive system using an object-oriented approach [32]. However, the use of UML in process modelling has also been investigated [24, 29, 33, 34],

and the OMG has also released an initial submission of a *Unified Process Model* (UPM) [35] which contains a model used to describe software development or related processes using UML. The UML constitutes of several different diagrammatic notations which are proposed to be used for different purposes in system design. For modelling the software process the most suitable notations have to be selected. While the adequacy of the various UML static diagrams for modelling the static part of software processes has been confirmed, modelling the dynamic part of processes with UML has been questioned [24, 33]. As a consequence, some extensions to the UML notation for process modelling have been suggested [23]. The UML is widely used in software engineering in the industry and also actively studied in academia [24].

3.3.5 Process-oriented Modellization and Enaction of software Developments

Process-oriented Modellization and Enaction of software Developments (PROMENADE) is an approach for supporting software process modelling building on the UML notation. In addition to presenting the notations to use in modelling, PROMENADE also presents a conceptual schema for describing a software process. The presented schema can be described within the conceptual framework discussed earlier in this chapter. PROMENADE uses standard static UML diagrams to describe the static part of a process consisting of the entities and relationships. The dynamic part is expressed with a textual notation supporting both proactive and reactive controls, as in JIL [23].

3.3.6 The Unified Process

The *Unified Process* (UP) is a representation of a software development process with the main goal of guiding the developers in implementing and deploying systems [36]. The Unified Process if heavily based on the UML and it gives guidelines how and where to use the different notations of the language in software development. The UP is focused on the lifecycle of a software system with an iterative and incremental approach where each iteration over time consisting of specific phases incrementally adds to the system. In each iteration various models of the system are built that are used and updated in subsequent

iterations. Thus, in addition to the guidelines how to use the UML notations in software development, the UP provides a software lifecycle model consisting of predefined phases, such as requirements, analysis, design, and implementation.

In a sense, the UP can be thought to be a process guide of a general software process that can be tailored to meet the needs of a specific organisation. However, the UP does not seem to have a conceptually solid process model behind it. Recently efforts have been made to formalise the model behind UP by formally explaining the entities and behavioural aspects behind it [36]. One motivation for these efforts is to enable the checking the consistency of the various models of the system [36].

4 THE EPG ENVIRONMENT

In this chapter an environment for process modelling and automatic EPG creation is described. Also, the usage of the environment in modelling and creating EPG instances of the processes of a telecommunications software company is investigated. This serves as a typical example case of modelling the processes of a modern software developing organisation.

First, the conceptual schema on which the process models are based on is discussed. The schema is built on the conceptual framework presented in Chapter 3. Next, the process modelling activities in the environment for modelling the processes of the software company are presented. The activities follow the guidelines presented in the process modelling subsection in Chapter 3. Third, the creation of an EPG from the process model in the environment is described. Finally, a summary of the architecture of the environment is given.

4.1 Conceptual schema

In this subchapter the conceptual schema used in the process modelling in the EPG environment is discussed. The schema follows the conceptual framework described in [21] and [9]. The notation used in some of the following figures is UML *static structural diagram* (SSD) notation. SSDs are also often called “class diagrams” but the term “static structural diagram” is used in this work since it describes the nature of the diagrams in this work more accurately. The notation is presented in detail in [32].

4.1.1 Entities

The structure of the entities used in the environment and in modelling the processes for the software company is described in Figure 5.

The figure is horizontally divided into three parts:

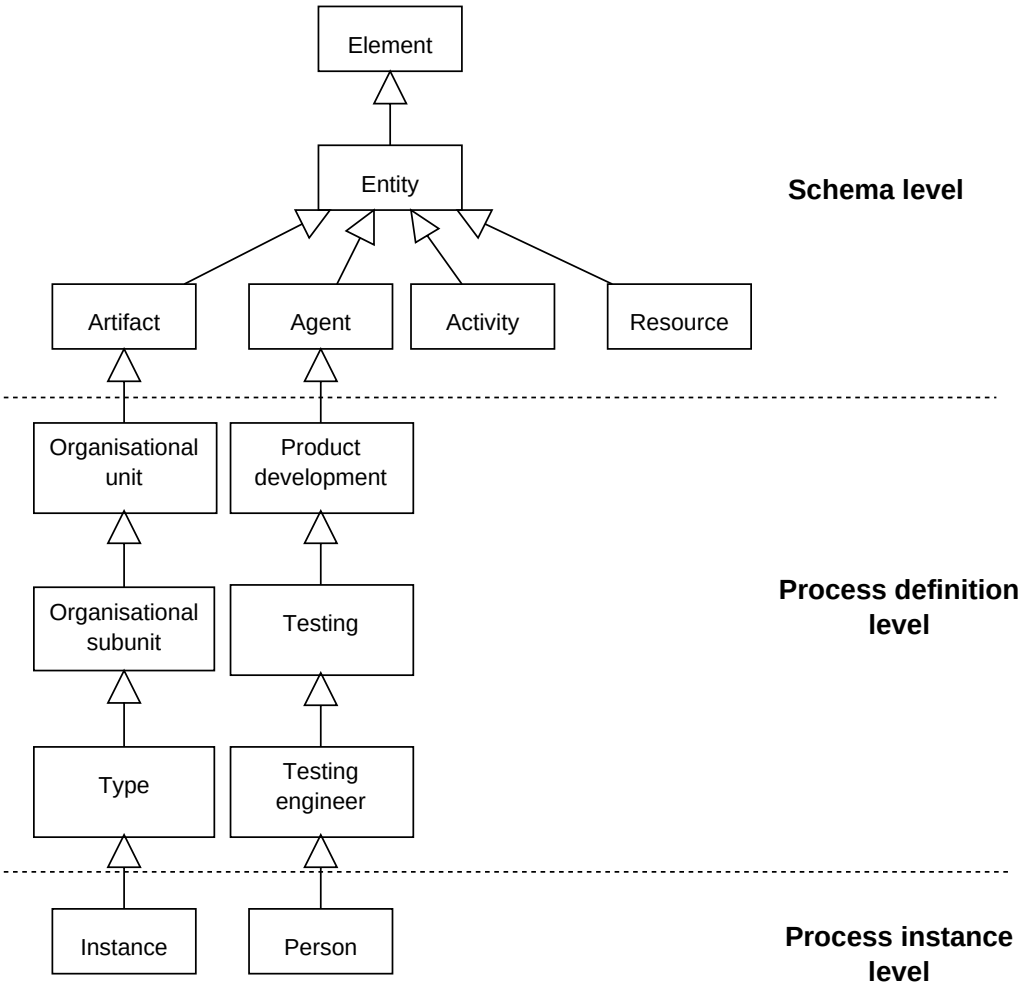


Figure 5: Entity structure.

Schema level	These are the entity classes that are implemented in the conceptual schema itself and thus are independent of the defined processes.
Process definition level	This is a part of a process definition created and updated by the process engineers and may vary depending of the defined process.
Instance level	This level contains the entities assigned by the process managers at the instance level of the process definitions.

In the uppermost part the basic entities of the schema are presented according to the framework described in Chapter 3. The middle part gives a more detailed description of the entities, which were used in the example case process definitions. As shown under the artifact basic entity class, the classes were further subclassed according to the organisational structure of the software company. An example of the subclassing is presented below the agent basic entity class, where examples of the organisational units, agent type, and instance are shown. The lowest part of the figure shows the instance level where the process instances are enacted by specific process performers. A practical example of the entity structure is also presented in Appendix 1.

4.1.2 Relationships

The basic relationships between the basic entity classes are shown in Figure 6 with SSD notation. The figure is incomplete in the sense that it only presents the most important relationships between the basic entity classes at a high level. In practise, a more complex structure is needed to describe the relationships in a database.

4.1.3 Behaviour

The schema supports several different ways to describe process behaviour. Reactive, event-driven control of the process is fully supported while proactive control is supported

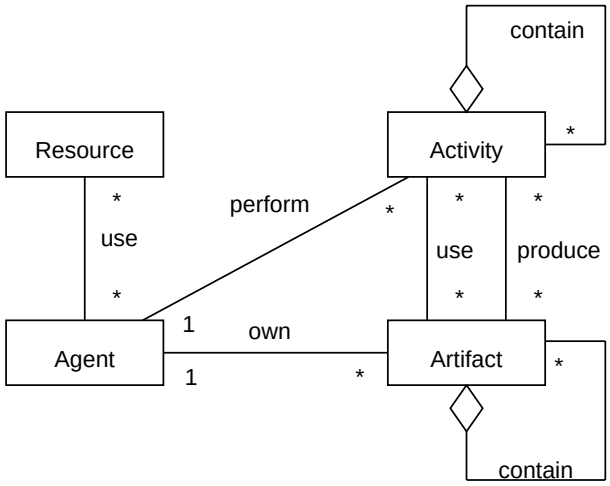


Figure 6: Basic relationships in the model.

with some limitations. Strong precedences of activities are fully supported, whereas weak and synchronising precedences are not supported with atomic activities. The effects of this limitation can be notably lessened by refining the activities to an appropriate detail level by creating subactivities and using reactive control, as described in the following subchapter focusing on modelling languages. Also, artifact and activity states are supported in the environment.

4.2 Process modelling

In this subchapter the phases of the software process modelling activities in the environment and in the software company are described. Although the phases are presented in the context of an example case, they are described in a generic way so that they illustrate the modelling in the environment. The whole descriptive modelling approach is based on the eight-step approach presented by Becker et al. [7]. The first four steps are executed only once, while the subsequent phases may be executed also during further SPI activities.

4.2.1 Modelling objectives and scope

The main objective for the modelling was to create a suitable model from where EPGs could be easily created. The most important goals for the EPGs created from the model were

- help communication about and increase understanding of the processes,
- offer support for the process performers, and
- aid in process improvement.

The EPGs were meant for the use of all the personnel in the organisation. Software engineers, as the performers of the software process, and quality personnel responsible for process improvement were natural target user groups of the EPGs. Also other personnel groups in the company should be aware at the software process at a high level to be able to communicate with other personnel and better understand the software that was produced. Finally, although the software process was the first target of the modelling activities, also other processes in the organisation could be modeled at a later stage in the environment to form an organisation-wide process guide. As a consequence, understandability and the ease of use were set as the prime requirements for the EPGs. Consequently, the prime requirement for the underlying process model was to enable the creation of EPGs fulfilling these requirements.

4.2.2 Conceptual schema

The conceptual schema of the process model was presented in the previous subchapter. As proposed by Becker et al. [7], the objectives of the model influenced the development of the schema. The prime goals and requirements of the model and EPGs included communication and understandability, and consequently the schema was designed as simple as possible to fulfil the needs of the users and the organisation.

4.2.3 Modelling languages

A suitable language or languages had to be selected for the modelling task. To fulfil the requirements set to the model and EPGs, the language(s) should be graphical and readable by all personnel without extensive training. Models concentrating mainly on better understanding of the process should be described with graphical representation, and precision should not be overemphasised in models used for process improvement purposes either [7]. Introducing a completely new and complex language to the organisation was out of the question, since the understandability of the processes was a prime requirement and the overhead of training was to be minimised. It was acknowledged that the selection of a readable, graphical language could mean more informal process descriptions. Since automating or simulating the processes was not an issue and transitions towards more understandable representations are suggested by researchers [1, 9], this was not considered a major problem. Also, presenting the EPGs and the underlying process model with different sets of languages was considered, but the overhead caused by using several languages for different purposes, probable complexity of the system, and lack of supporting tools were some of the reasons for choosing the same language(s) for both modelling and presenting the EPGs.

Table 1: Representation of main relationships

Relationship	Diagram	Representation
Activity is performed by an agent	AD	Activity at swimlane
Activity uses an artifact	AD	Incoming dashed line
Activity produces an artifact	AD	Outgoing dashed line
Activity contains activity	AD	Subactivities
Artifact contains artifact	SSD	Subclasses
Artifact is owned by agent	Form	Textual information
Resource is used by agent	AD	Comment connected to activity

Two different UML notations were selected for modelling and EPG purposes. The main reasons for selecting UML notations was their familiarity, readability, and expressiveness. The static part of the processes were decided to be modeled with UML SSDs, and the dynamic part with UML *activity diagrams* (AD) which have been found to be the most suitable UML notations for modelling the behaviour of processes [33]. Whereas the suitability of SSDs for modelling the static part of processes has been found adequate, the

suitability of ADs for modelling the dynamic part has been questioned [24, 33]. The lack of mechanisms to express proactive control and activity properties have been presented as the main limitations of ADs. In our modelling approach the first issue was circumvented by refining the activities to more detailed subactivities and using reactive control with the subactivities where necessary. The latter of the deficiencies was lessened by introducing textual forms for each entity in the model containing detailed information about the entity in question. The representation of the main relationships illustrated in Figure 6 is described in Table 1 and practical examples of the diagrams are presented in Appendix 1.

4.2.4 Tool selection

Tools were needed to support the generation of the UML diagrams and textual forms. Some requirements for the tools were the ease of use and effortless updating existing models. *Toolkit for Conceptual Modeling* (TCM) [37], which includes support for the needed UML diagrams and includes also limited syntax checking, was selected as the tool to draw the diagrams. Additional advantages of TCM were a well-documented human-readable file format, which was needed for information extraction from the models, and good exporting features for other image formats. The representation of subactivity diagrams was not supported by the tool and file-system directory hierarchy was utilised describe these relationships. Also, version control of the diagrams was not implemented in TCM and external methods were used. Web-browser was selected as the main user interface tool to be used, for example, in updating the textual forms containing the detailed entity properties. Using a familiar interface, such as a web-browser, for accessing and updating the process information supports the fulfilment of the objectives of ease of use set for the process models.

4.2.5 Elicit process knowledge

The initial goal of the modelling was to create a descriptive model and, consequently, eliciting existing process knowledge was necessary to acquire the information to describe the process. The main methods used at this phase were studying existing documentation,

observation, and interviewing involved people. Although previous systematic company-wide process modelling activities had not been executed in the organisation, process documentation existed for many of the individual processes in various forms. Where suitable, up-to-date process documentation did not exist, more emphasis was put to observation and interviews.

4.2.6 Model creation

At this phase the information gathered in the elicitation phase is formalised and modeled based on the conceptual schema with the selected languages and tools to achieve the objectives within the scope of the model. As suggested in [7], the modelling in the environment should begin by modelling the entities. The agents and artifacts with their structural hierarchies, presented in the middle part of Figure 5, were modeled as SSDs using the TCM modelling tool. Each of the subclasses was modeled as a separate SSD file to create a modular tree hierarchy, which could be easily managed and further processed. A practical example of a SSD tree is presented in Appendix 1.

Next, the activities, artifact flow, and process behaviour were modeled as ADs with the drawing tool. The whole process was not described with a single AD but a layered structure was created with different levels of abstraction. At the highest level only a single activity needs to exist (e.g. 'Create software') which can then be refined with subactivities as needed. Thus, a tree-like structure of the software process was generated where the leaves represent the atomic action states. Every node in the tree, with the exception of the atomic leaf nodes, is represented by a UML diagram and consequently a single TCM file. The described structure is highly maintainable enabling the process engineers to concentrate on specific parts of the process at a chosen abstraction level. The processes can be refined to an appropriate level according to the context and performers, as suggested by [13]. The structure also supports the reuse of processes within the model by allowing linking of nodes to different parts of the tree. Incremental modelling, suggested in [10, 13], is also supported by allowing the model to begin at a high abstraction level. A very simple example of an activity structure is presented in Figure 7 and a practical example of an activity/subactivity relationship is shown in two ADs in Appendix 1.

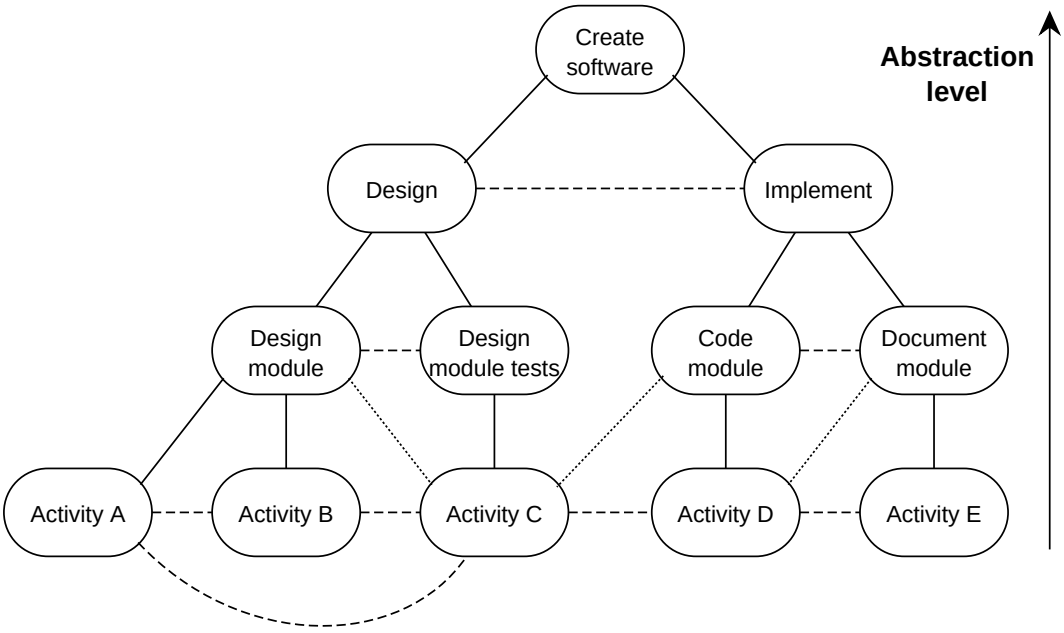


Figure 7: Example structure of an activity tree

The activity-subactivity relationships are illustrated in Figure 7 with the vertically aligned solid lines. One activity may have a maximum of one parent node but an unlimited number of subactivity nodes. The dotted lines represent the links enabling the reuse of the processes in different processes and parts of the tree, i.e. an activity may be a subactivity for multiple activities. For example, in the figure *Activity C* is a common subactivity of three activities but it has only one parent *Design Module Tests* – the other two activities are only linked to *Activity C*. The more horizontally aligned dashed lines represent the behavioural relationships between activities. Usually all the subactivities of a given activity have behavioural relationships with each other, and all the behavioural information presented higher in the tree affect the activities lower in the same branch. Activities that are linked as subactivities are affected by the behavioural information presented in the branch of the linking node, i.e. one activity may be affected by different behavioural information depending on the branch it is accessed from.

Next, process information is extracted from the SSDs and ADs, and inserted to a database. This information includes, for example, the entities with their structure, artifact flow, ac-

tivity/subactivity relationships, etc. The information extraction is fully automated and performed by a program traversing the SSD and AD tree hierarchy, and parsing the TCM source files. After the process database has been created, other information about the process entities may be inserted into the database by filling forms with a web-browser using an administrator process guide described in the next subchapter. Examples of inserted entity properties include entity descriptions, activity objectives and procedures, and the artifact owner relationship. Thus, the problems resulting from ADs deficiencies in presenting activity properties are removed by presenting this information in separate forms.

Although modelling in a bottom-up fashion is suggested as the means for process modelling in a new organisation [7], in the example case the modelling was performed as a combination of top-down and bottom-up approaches. While bottom-up modelling was used when collecting the information about individual process steps from the performers and existing documentation to form larger entities, the top-down approach was used in the higher parts of the structural tree influenced mainly by the organisational structure of the company. In the initial phase some of the most important development processes in the company were modeled in the environment according the method presented here. At this stage a model consisting of a set of over 400 activities, 50 artifacts, and 25 roles was created and presented in the SSD and AD trees discussed above. The maximum depth of the AD tree was five levels.

4.2.7 Model and process analysis

The process model should be analysed to ensure that the process is consistent [7]. Syntactic and semantic analysis of the model should be conducted to avoid inconsistencies and resulting performer misunderstanding and confusion. In the model created, the analysis was performed mostly manually by the process engineers. Although the drawing tool (TCM) included some syntax checking facilities, they were not used extensively because of their limited capabilities. Semantic analysis of the model was performed mainly by the process engineers and the performers of the process validating the modeled processes. Qualitative and quantitative process analysis using the model and the EPGs within the organisation is discussed in the next chapter.

4.3 EPG creation

In this subchapter the creation of EPGs from the process model is described. The model consists of the SSDs, ADs, and a database containing process information, as described in the previous subchapter. The database was created based on the information extracted from the diagrams, and inserted via forms. The created model is a generic model from where instances can be created for different purposes. In fact, a process guide can be seen as an instance of the underlying process model containing the relevant parts of the model depending on the intended use of the guide. An EPG, as any other instance, can be customised to include modified or additional information about the process. In this subchapter the phases involved in creating instances of the generic model, for EPG or other use, are described.

4.3.1 Planning information content

Planning the information content of the the instance depending on its intended use is the first step in instance creation. Decisions about presenting the whole process model or just specific parts have to be made. Also, any additional information needed in the instance, such as notes, external documentation, and hyperlinks connected to specific entities, has to be decided upon. Examples of such information include instance specific guidelines, warnings, and small deviations from the generic process. Large deviations from the underlying process model are not possible in this framework, and a modified process model should be created for such deviations. Although creating a new, modified process model is fairly simple by copying and modifying the existing model, the sufficient generality of the existing model can be questioned in such a case. The inclusion of metrics to measure the enactment of the process, needed especially in SPI activities as described in Chapter 1, has also to be decided.

Several EPG instances were created in the organisation. One organisation-wide guide was developed as a general browsable description of all the processes presenting the process model without any additional information. Also project-specific instances were created for process model validation purposes with additional information and metrics.

4.3.2 Customising layout and metrics

After the content of the EPG has been planned, the format in which the content is presented has to be designed. Since the guide is presented in hypertext format, the layout design involves designing suitable HTML templates for the information. The templates contain HTML with specific tags denoting the insertion of the data from the model. Using HTML enables the possibility to customise the look and contents of the guide according to the preferences of the users. Optionally customising scripts can be created to perform specific actions, such as fetching a current version of a document at the creation time of the guide. Different templates and scripts can be specified for each node of the model hierarchy tree except the leaf nodes, which are not presented by SSDs or ADs. Also, if the need for process measurement was uncovered in the information planning phase, the scripts for including the metrics in the model are prepared if they do not exist already.

In all the instances that were built, the same layout was used so that all the instances had a similar look and only one set of HTML templates was needed. Some customised scripts were used to fetch up-to-date documentation to the guide from different parts of the company intranet at guide build time. For measurement and performer feedback purposes three possibilities were created with different options for process enactment time input. In the organisation-wide guide it was possible to insert text as feedback related to any entity of the model through the web-browser. In the project-specific instances the insertion of activity performance time at process enactment time was added as an additional metric. The third possibility was a “bare” version without any performer input features. Examples of the simple guide layout for ADs and SSDs are presented in Appendix 1 and examples of the textual forms, including a screen describing the insertion of activity performance, are presented in Appendix 2.

4.3.3 Building the guide

At this phase a guide is built based the information provided by the model and the planned information content with the layout defined in the previous phase. The building of the guide from the TCM source files and process database is fully automated, only the name

of the guide, the location of the *root* node of the guide in the process model tree hierarchy, and information about the customising and layout scripts are needed. Thus, the guide may consist of any subtree of the model and concentrate on specific processes. As illustrated in Figure 7, an activity may have links to other activities in the activity tree which also have to be followed to create a complete subtree. For example, if a process guide was to be created about *Implementation*, *Activity C* would be a part of this guide since it is linked to subactivities of the implementation node. The HTML templates chosen at the previous phase are used to create the layout of the guide and the scripts used for measurement are included, if needed. The generated guide consists of three interconnected parts:

Dynamic part: The dynamic part consists of the ADs and the instance-related information presented with the designed layout. The AD itself is an image created from the TCM sources with all entities parsed to HTML imagemaps, so that every entity on-screen is a clickable hyperlink. The imagemapped hyperlinks are used to describe the structure of the selected subtree by illustrating the activity/subactivity structure and links in the tree. Also, the hyperlinks enable the access to the detailed properties of each entity.

Static part: The static part contains the SSDs and the customised instance information presented with a layout designed in the previous phase. As in the dynamic part, the SSDs are represented by images created from TCM sources. The entities are represented by clickable hyperlinks to describe the detailed entity properties and structure of the static part of the model.

Textual part: The textual part contains the detailed entity properties with instance information. Hyperlinks to other relevant parts of the guide and additional information about the relationships of the entities is also provided. Examples of the automatically created web-forms include details of the flow of a specific artifact, activities of a specific agent, etc. Also, the possible measurements inserted in the instance are presented in the textual part of the related entity.

The building of a guide can be managed through a web-interface where the information needed to build the instance is inserted. Examples of the screens of the different parts of a guide are presented in Appendices 1 and 2.

Navigation within the guide is provided with the means described in Table 2. One thing to notice is that at this stage there is not straight links between the dynamic and static parts – navigation between these parts must go through the textual part. The implementation of these straight links has not been considered necessary due to the other navigation mechanisms.

Table 2: EPG navigation.

From	To	How
Activity digrams	Textual forms	Imagemaps and hyperlinks
Static structure diagrams	Textual forms	Imagemaps and hyperlinks
Textual forms	Activity diagrams	Hyperlinks
Textual forms	Static structure diagrams	Hyperlinks
Activity diagrams	Activity diagrams	Imagemaps and hyperlinks
Static structure diagrams	Static structure diagrams	Imagemaps and hyperlinks
Textual forms	Textual forms	Hyperlinks

4.3.4 Customising instance

Before the guide was built, some of the properties of the guide were already customised in the layout definition phase. These included information within the HTML templates of the dynamic, static, and textual parts, which could also be enhanced with scripts executed at build phase. Also, the inclusion of metrics to measure the process was customised before building the guide. At this phase the information contained in the textual part, i.e. the detailed entity properties, can be customised with a web-interface created at the build time. This information can include, for example, guidelines or procedures related to a entity in this specific instance of the process model.

4.3.5 Administrator guide

The administrator guide is a special version of the model that can be built similarly to a regular process instance. The guide allows the modification of the information stored in the process database by providing a web-interface to the database. Only the entity properties can be modified through the administrator guide – modifying the behaviour

or relationships of the entities is not necessary since the information is extracted directly from the SSDs and ADs. Modifications made with the administrator guide are related to the process model and thus affect all the instances based on the model. The administrator guide is an easy and fast way to insert and update the detailed process information in the database. Examples of the screens of an administrator guide are shown in Appendix 3. The screens of the administrator guide are similar to the screens of a regular instance with the exception that most of the fields are editable.

4.4 Summary of the environment architecture

In this subchapter a summary of the environment combining the process modelling and EPG creation is presented. Also, a view to the underlying technical architecture of the environment is given. The summary of the environment is given in the form of a description of the steps performed in the EPG creation in the example case:

1. setting objectives and scope,
2. designing conceptual schema,
3. selection of languages and tools,
4. process information elicitation,
5. model creation,
    (a) drawing or modifying the SSDs and ADs with TCM,
    (b) building a process database from the TCM source files,
    (c) building an administrator guide from the TCM source files,
    (d) inserting general process information into the database with the administrator guide web-interface,
6. EPG creation,
    (a) planning information content,

(b) customising guide layout and metrics,

(c) building the EPG from the TCM source files and process database, and

(d) customising the EPG details with the web-interface.

The first three steps were executed only during the creation of the environment and need not to be executed in the future unless changes to the schema are needed. The following two steps may be executed when changes to the model are needed, and the last step whenever an instance of the model has to be created. Small changes to the model and all the instances based on it can be made with the administrator guide at any time, while larger changes require building the whole model anew.

The architecture of the environment is illustrated in Figure 8 where the people, tools, and the underlying architecture of the environment are shown interconnected with arrows representing the information flow. The architecture can be compared to the process cycle presented in Figure 2 to illustrate how the different parts of the architecture correspond to the sections of the process cycle. The three sectors *A*, *B*, and *C* in the cycle are also presented with three vertical sections in Figure 8 with their respective performing roles and tools. The information flows presented in the center of the process cycle – feedback, process descriptions, and instantiated process – are presented in the architecture information flows.

Although process feedback is shown as flowing straight from a role to another, the EPG itself can be used as the means for the performers giving process feedback to the process managers, as described in the subsection concentrating on customising the EPG layout. In this case the process feedback is stored in the process instance database.

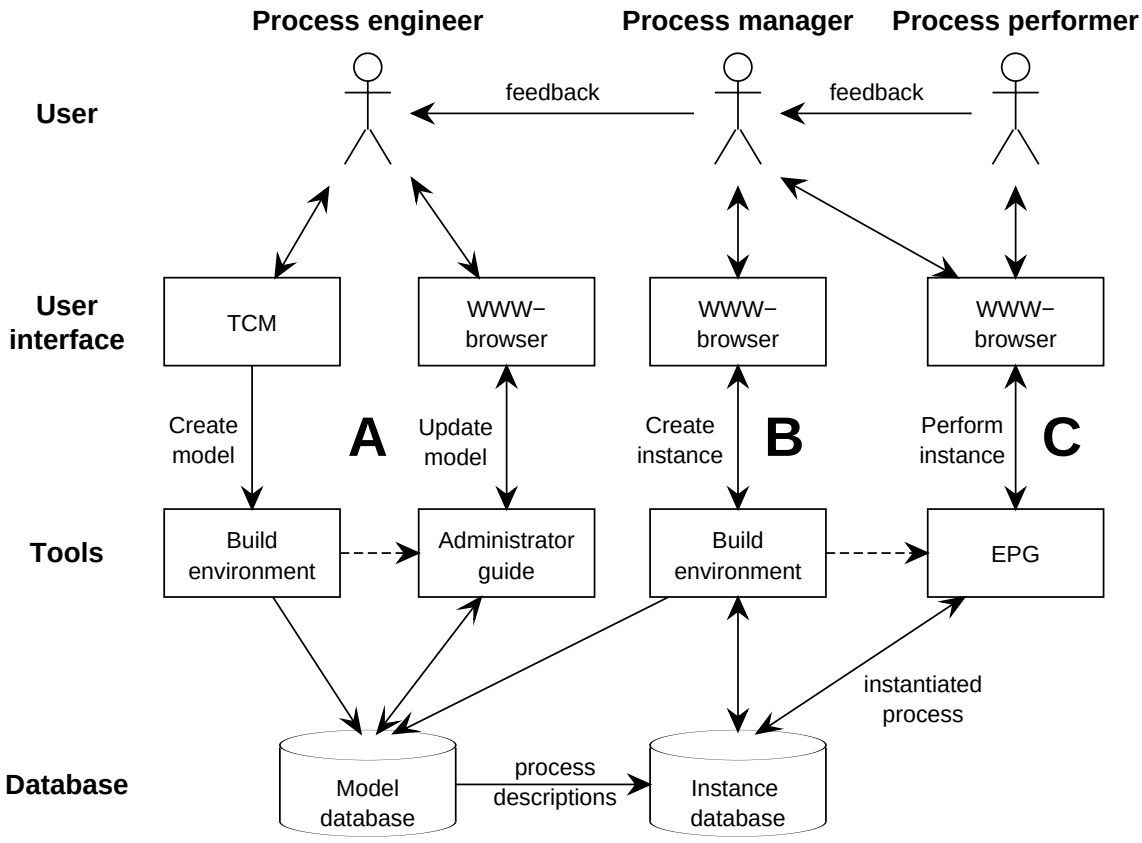


Figure 8: EPG environment architecture.

5 EPG SUPPORT FOR PROCESS IMPROVEMENT AND ENACTMENT

In this chapter the usage of the EPGs created in the environment described in previous chapter is investigated in a telecommunications software company. The usage is examined from the viewpoints of the objectives set to the EPGs in the process modelling part, namely the support for process performers and the support for SPI activities. The objective was to explore how the EPGs and the environment could support the existing SPI and process performance activities in the example company. The structure of the chapter follows the structure of the process improvement model presented in Figure 3 as an example of the generic steps used in CPI. The possible uses of the environment and the EPGs in the individual steps of the cycle are discussed in the context of the target organisation.

5.1 Evaluate process

In this phase the main objectives are understanding the software process, evaluating the process using qualitative and quantitative information, and find process improvement opportunities [6]. The recommended means for understanding the process is to create a descriptive model [6]. In the company a previous process model existed, which was considered to be outdated and too rigid for the current needs of the company. Although descriptions had been developed for most individual parts of the software process, there was a need for a more holistic view of the processes in the company. The individual processes were documented with semiformal textual forms or graphical notations and a standard formal language to describe processes did not exist.

The software process had been evaluated through external SPICE assessments and smaller scale internal assessments and the resulting qualitative information had been used for evaluating the current state of the process and planning further process improvement. In comparison with other Finnish software companies the results of the assessments of the target company were well above average. Also some quantitative information for process analysis had been available, such as defect counts, but the need for more project spe-

cific quantitative data for process analysis was acknowledged. While the process assessments were the main source for process improvement opportunities, also other important sources, such as process performers, customers, and management, were used to identify possible targets for improvement.

Increasing process understanding in the company was one of the main objectives of the EPGs created in the environment. Especially the process engineers designing and improving the processes require through understanding about the individual processes and the software process as a whole. The administrator guide containing the whole process tree and the underlying model provides a holistic view to the processes of the company at a needed level of detail. In a sense, an the EPG environment provided the glue to bind the individual processes together and to describe their interconnections. While the existing process documentation was converted to a common notation, also the original documents were attached to appropriate places in the process framework. Thus, the administrator guide and other process instances work also as a central location from where all process documentation in the company can be accessed and maintained. The process engineers can access and update specific parts of the software process and all their related documentation by browsing the administrator guide. Once the underlying descriptive model of the process was created and kept up-to-date, separate modelling phases are not needed in subsequent SPI cycles [6].

The design of the EPG environment allows the insertion of measurements at any given activities, artifacts, or roles for specific process instances. As an example, the performance information of an activity in a project was described in the previous chapter and illustrated in Appendix 2. Project specific EPG instances of the process model with the needed metrics can be created and published in the company intranet as a tool to gather information about individual processes or projects. Measurements submitted in an EPG through a web-browser are inserted to the instance database, from where they can be fetched for further analysis. Thus, the EPG instances provide the means for acquiring quantitative information about processes, which is presented as the preferred basis for process evaluation [6].

A process representation does not suggest any process improvement opportunities in itself, but an EPG can be used as a vehicle for gathering process improvement ideas from

performers or management. The ideas submitted through an EPG instance in any project are attached to a specific process or entity and stored to a database. This enables the EPG to function as a repository for process improvement proposals, which is presented as a useful source of ideas for SPI [6].

5.2 Develop process

The develop process phase includes setting the goals, planning, and executing of the process development [6]. The goal setting or process development planning are not directly affected by the EPG environment. Although the availability of a larger amount of quantitative information from processes may enable the setting of more specific goals and constraints for process development, the current procedures used in planning process development in the company are not affected. Existing, formal PAL to help in identifying and understanding reusable process assets was not available, and the main sources for process alternatives in the company were the various existing documentation, process performers, and literature as also suggested in [6]. The process alternatives were evaluated mainly based on qualitative information from assessing the process, and a single standard method for describing the chosen process did not exist. Usually the process was documented with a textual document and semiformal diagrams, such as flowcharts.

The EPG environment supports process reuse in different processes and instances through the linking of different parts of the process tree as described in the previous chapter. This allows the reuse of proven processes or subprocesses in process development, and facilitates the search for alternatives in existing processes. The process engineers may browse the process model through the administrator guide in searching suitable candidates for process reuse. Also, if the found alternatives have been used in a previous project, quantitative information may be available to aid in their evaluation. The processes are described with the UML notations and forms using the tools presented in the previous chapter, which the process engineers have to be familiar with. Additional and existing documentation in any format can be attached with hyperlinks to any part of the process model.

Verifying, validating, and possible prototyping the process are also tasks belonging to this phase [6]. Since the process model is described in a more formal manner than the processes in the company before, more attention has to be paid to the verification of the processes to keep the model consistent. This requires syntactical and semantical analysis of the individual processes and the model as a whole by the process engineers. EPG instances can be created to support and provide feedback channels for the usual reviews and possible prototyping of processes during validation phase.

The next step in the spiral model presented by Kellner et al. [6] is documenting and packaging the process material to an appropriate form depending on the subsequent use of the process. In the EPG environment this would mean the instantiation and customising of an EPG instance to satisfy needs of its planned use.

5.3 Install process

At this phase the process developed in the previous steps is introduced to the organisation by establishing the supporting infrastructure and performing detailed project planning for the projects where the process is to be installed [6]. Staff and other resources for supporting the process performance in projects have to be assigned regardless of the tools and process models used. The advantage of the EPG environment may present in this phase is in the training of the staff for the new process. Instances of the processes with common look and feel can be created for training purposes, thus providing a vehicle for process training that the performers to be trained are familiar with. The process descriptions in the company had previously been trained with differing notations and tools, and there was not a single standard way for presenting the processes to performers. Naturally, the notations used in the EPG have to be trained in the company to people unfamiliar with UML, but since all the processes are presented with the same notations the training could be organised with reasonably small effort.

Project specific planning for process use may involve tailoring the process to the specific project needs, which can be done by process instance customising. EPG instances can be created for projects and customised according the needs of the project.

5.4 Performance and monitoring

Since the EPG environment was to provide support for both SPI and process performers, this subchapter is divided in to two parts respectively. First, the support for process performers is discussed and afterwards the support for SPI by monitoring the use of the process is investigated.

5.4.1 Process performance

The main motivation for EPGs has been the support for process performers and their use for this purpose has been discussed [6, 9, 10]. In this subchapter the EPG support for process performers is discussed in the context of the described EPG environment and the target company. As mentioned earlier, the company had existing process documentation albeit in various formats and locations in the company intranet. Existing employees were considered to have a good understanding about the parts of the software process directly affecting them, and the process documentation was rarely used as a reference. The documentation was mainly used when the processes needed to be communicated and changes were needed. However, the need for increasing the understanding of the interconnections of the individual processes and the software process as a whole was acknowledged. Especially new employees needed process documentation that could be easily accessed and understood. Also, tailored project specific process documentation was uncommon and consequently process event information, as mentioned in [9], was not captured.

The EPG environment provides an easily navigable, centralised location for process documentation that is accessible for performers of the process through a web-browser. The processes are described with a common notation, even though documentation in other formats and notations can also be attached. Due to the tree-like structure of the description, the processes can be viewed at an individual or at a higher level providing a view to the surrounding or connected processes of any given process. This enables the process performers to have a broader look of the software process. The EPGs can be used in communicating and visualising changes to the processes with a common notation. Additional information, such as links to templates and other internal or external documentation, can

be included in the guides especially to help new employees in familiarising themselves with the company's software process. The creation of process instances enables the process performers to track their work and store process information by inserting data.

5.4.2 Monitor process use

This phase is part of the SPIE focused on monitoring the actual use of the process [6], described also in the previous subchapter. Basic issues to monitor in process performance include process understanding, adherence to the process definition, problems during execution, measurement recording, and recording process variations. Additionally, improvement suggestions and feedback should be collected, and assistance for process performers and managers should be offered throughout this stage of the process [6].

General process understanding in the company had been measured by internal assessments concentrating on software process. The method used was Finnish KYKY model, which is a self-assessment method based on the ISO9001, CMM, and SPICE developed for fast evaluation of processes. The results of the assessments indicated that the company rated among the top companies in Finland in understanding the process-oriented way of work. Monitoring adherence to the defined processes and recording process variations in single projects had not been systematically practised. Monitoring the recording of measurements was possible through a web-browser when specific measurements, such as defect counts, were collected to the intranet by the process performers. Acquiring feedback about the process from the performers and process managers was accomplished with traditional ways of communication, without any specific methods and tools.

The EPGs provide a feedback channel for the process performers where the feedback can be targeted to a specific entity on the process and stored to a database for further processing. For example, problems and improvement suggestions can be inserted directly into the related artifact or activity in a process instance that is affected by the feedback. Also, since the measurements are integrated in the EPG instances themselves, they can be monitored real time by extracting information from the instance database. Not only the insertion of measurements can be tracked but also the state of the process itself can be monitored within the possibilities offered by the specified measurements. Inspecting

key activity performance times, for example, may provide useful information about the whole process. Process engineers can offer assistance for process performers by inserting additional documentation to any entity in the EPG instance via the administrator guide.

6 DISCUSSION AND CONCLUSIONS

The purpose of this work was to develop an environment for creating EPGs about the software process to support software process improvement and enactment. The motivation for the work was based on the assumption that a properly designed and implemented EPG can be a valuable tool in all phases of SPI activities as well as process enactment. The usability of the EPGs created in the environment for supporting SPI was investigated by modelling the software process of a telecommunications software company and studying how the current SPI efforts and process performance in the company could be supported by the EPGs.

The main goal of this work was to design and develop the EPG environment itself. The EPGs have not yet been experimented with in large-scale practical SPI or project activities, although the possibilities for practical implementations have been discussed with the process personnel in the target company. The results of these discussions about the use of the environment and the EPGs in supporting SPI and process performance in the company were presented in Chapter 5. Although the majority of the discussions about the environment and the use of the EPGs were promising, and the architecture was found to provide support for many aspects of software process engineering in the company, the real value of the work can only be evaluated through practical experience. The most useful feature offered by the environment from the process performers viewpoint was considered to be the use of the environment as an easily accessible, centralised source for process-related information. The most promising opportunities offered for the SPI efforts in the company was the support for process tracking, measurement, and feedback. Possible problems facing the practical use of the environment were thought to be the necessary training and motivation of the process performers and personnel. Even though this work was focused on the the actual process model and tools, they are only small part of taking advantage of process technology. Neglecting the cultural and organisational issues, discussed for example in [3], will result in ineffective process use.

Kellner et al. [9] discuss EPGs for offering guidance to process participants and present an initial prototype. They propose a number challenges for future EPG development, including dynamic creation of the EPGs from a process model database, easier layout

managing, searchable EPGs, and configuration management. In the environment presented in this work many of those issues were addressed: the EPGs were created from a model database, the layout of the guides could be managed with one HTML template, and queries to the underlying process database could be inserted in the guides. Although several versions of process models and EPGs could exist in the environment, configuration management is a complex research area in itself and can possibly be handled best by specialised, external tools. Instantiation and customisation of the EPGs have also been presented as possible areas for future improvement. Tailored personal or group copies with various possibilities for interactivity, such as inserting personal annotations and collecting data from process participants, are examples of the opportunities discussed for future EPGs [10, 9]. In this work, these aspects of EPGs are approached by offering a possibility to create and customise process instances, and to insert the elements needed for interaction directly in them.

The combined process modelling tool and EPG generator, presented by Becker-Kornstaedt et al. [11, 28], is an approach combining the development of process models and EPG creation with a recently added support for process participant annotations [27]. The main goal of the approach is to provide support for the process performers [28], and the use of the architecture in SPI context has not been extensively discussed. In this work a differing approach to EPG development and usage was taken due to the SPI focus. For example, methods for project-specific EPG creation and the introduction of metrics in the instances were discussed and implemented. However, the basic requirements for EPGs offering support for process performers and SPI are essentially the same, namely increasing process understanding and improving communication about the process within and between different user groups, although the details of the communication may be different.

The feedback from the initial study in the suitability of an EPG for supporting SPI activities in the target company was promising, and work should continue with experimenting with the use of the EPGs in practise. The usability and possible advantages of the environment and the EPGs cannot be determined without validating them with actual users. In future, experiments with the EPGs in practise are needed to fully assess the usability of the environment. If the environment and the EPGs prove to be a helpful tool in software development context, the model may be extended to cover also other processes in an organisation, and thus evolve towards an organisation-wide process guide.

7 SUMMARY

The quality of software has received considerable attention in the last couple of decades both in academia and industry. The attention is largely based on the assumption that the quality of the software is highly dependent on the quality of the process used to produce it. As a consequence, the software process and its improvement has been a focus of most of the attempts in improving software quality. Unfortunately, software processes are complex entities and their understanding and performance is not a trivial task.

In order to fully understand a process, a description of the process has to exist. Whereas a single individual may understand and perform a process flawlessly without explicit process description other than an image in his mind, processes affecting more than one person encounter problems. Images in the minds of two people about a process will rarely be exactly same, and while both of them *think* they are performing the same process, they probably are not. To form a common understanding and agreement of a process, the process has to be communicated. Process engineers developing processes, process managers introducing them into their projects, and process performers carrying out the software process all need a description of the process to perform their part and communicate the process to other people.

Software process improvement is no exception. Something that is not unambiguously understood and can not be communicated is impossible to improve. The successful introduction of the various software improvement and assessment paradigms, such as SPICE or CMM, requires explicitly defined and described processes from a software developing organisation. Process improvement, if anything, involves intense communication between different parties and thorough understanding of the process itself.

Most of the attempts on describing the software process have been focused on highly sophisticated and detailed process definitions that are described with textual “process programming languages” or complex graphical models. While these representations are justified by the need to be precise and they are required for complex simulations and automation, often the real need for process descriptions is process understanding and communication. Not only are these traditional process models often difficult to understand

and update by the process performers, managers, and engineers, but their development and maintenance can be time-consuming and expensive. For the purposes of communicating and understanding a process, the method and medium used to represent the process information also plays an important role. Traditional, paper-based process documentation places numerous limitations for effective process maintenance and communication.

In this work an environment for creating process definitions and representations for the purposes of supporting process improvement and performance was described. The main requirements for the representations created in the environment were understandability and their ease of use, without sacrificing the necessary accuracy and preciseness of the description. In the environment the processes are modeled based on a solid conceptual schema with a tool using graphical UML notations. The individual processes are modeled in separate diagrams with various levels of abstraction to form tree-like structures of the static and dynamic aspects of the software process. Useful process information is extracted from the diagrams and inserted to a database using automated scripts. The process model, consisting of the diagrams and the database, can be modified and updated through a web-interface. Instances, or *electronic process guides*, of the process model can be created, customised, and published using a web-browser. The process instances take advantage of current web-technologies by providing an effective and usable interface for the navigation and modification of instance-related information. Also, process instance creation for specific projects enables the insertion of project-specific measurements and provides a communication channel for supporting software process improvement activities.

The environment was used to model the software process of a telecommunications software company. The support offered by the process model instances for the software process performers and improvement was investigated in the company by exploring the possibilities for their use. In the initial investigations the environment and the EPG instances were found to offer wide support for process improvement and performance in the company. However, further experiments with the environment and the instances are needed to assess their true usability – a proper assessment cannot be done without results and feedback from real users in concrete projects.

REFERENCES

[1] Fuggetta A. Software process: a roadmap. In Finkelstein A, editor. The future of software engineering. ACM Press, 2000.

[2] Gibson R. Software process modeling: theory, results and commentary. Proceedings of the 31st Hawaii International Conference on System Sciences; 1998 Jan 6-9; Kohala Coast HI, USA; IEEE Computer Society Press, 1998.

[3] Zahran S. Software process improvement. Harlow, England: Addison-Wesley, 1998.

[4] Bandinelli S, Fuggetta A, Lavazza L, Loi M, Picco GP. Modeling and improving an industrial software process. IEEE Transactions on Software Engineering 1995; 21(5):440-454.

[5] Osterweil LJ. Software processes are software too, revisited: an invited talk on the most influential paper of ICSE 9. Proceedings of the International Conference on Software Engineering; 1997 May 17-23; Boston MA, USA: ACM Press, 1997.

[6] Kellner MI, Briand L, Over JW. A method for designing, defining, and evolving software process. Proceedings of the 4th International Conference on the Software Process; 1996 Dec 2-6; Brighton, UK: IEEE Computer Society Press, 1996.

[7] Becker U, Hamann D, Verlage M. Descriptive modeling of software processes. Proceedings of the 3rd Conference on Software Process Improvement; 1997 Dec; Barcelona, Spain. 1997.

[8] Heinemann GT, Botsford JE, Caldiera G, Kaiser GE, Kellner MI, Madhavji NH. Emerging technology supporting the process cycle: the process reuse study at IBM CAS. IBM Systems journal 1994; 33(3):501-529.

[9] Kellner MI, Becker-Kornstaedt U, Riddle WE, Tomal J, Verlage M. Process guides: effective guidance for process participants. Proceedings of the 5th International Conference on the Software Process; 1998 Jun 14-17; New Jersey, USA: The International Software Process Association Press, 1998.

[10] Becker-Kornstaedt U, Verlage M. The V-Modell guide: experience with a web-based approach for process support. Proceedings of the 9th International Workshop on

Software Technology and Engineering Practice; 1999 Aug 30-Sep 2; Pittsburgh PA, USA: IEEE Computer Society Press, 1999.

[11] Becker-Kornstaedt U, Scott L, Zettel J. Process engineering with Spearmint^{TM}/EPG. Proceedings of the 22nd International Conference on Software Engineering; 2000 Jun 4-11; Limerick, Ireland: IEEE Computer Society Press, 2000.

[12] Cambridge dictionaries online [online] 2001 [cited 2001 Nov 5]. Available from: http://dictionary.cambridge.org/.

[13] Feiler PH, Humphrey WS. Software process development and enactment: concepts and definitions. Proceedings of the 2nd International Conference on the Software Process; 1993: IEEE Computer Society Press, 1993.

[14] Pressman RS. Software engineering: a practitioner's approach. 4th ed. Cambridge, England: McGraw-Hill, 1997.

[15] Dandekar A, Perry DE, Votta LG. A study in process simplification. Proceedings of the 4th International Conference on the Software Process; 1996 Dec 2-6; Brighton, UK: IEEE Computer Society Press, 1996.

[16] Rossi S, Marttiin P. Adoption of integrated process and product support for software engineering in SP Jyväskylä. Proceedings of the International Conference on Software Methods and Tools; 2000 Nov 6-10; Wollongong, Australia: IEEE Computer Society Press, 2000.

[17] Madhavji, NH. The process cycle. Software engineering journal 1991; 6(5):234-242.

[18] McFeeley, B. 1996. IDEAL^{SM}: a user's guide for software process improvement. Technical report. Pittsburgh PA, USA: Software Engineering Institute, Carnegie Mellon University; 1996 Feb. Report No.: CMU/SEI-96-HB-001.

[19] Varkoi TK, Mäkinen TK. Case study of CMM and SPICE comparison in software process assessment. Proceedings of the International Conference on Engineering and Technology Management; 1999 May 3-5; San Juan, Puerto Rico: IEEE, 1998.

[20] International Organization for Standardization (ISO), International Electrotechnical Commission (IEC). ISO/IEC TR 15504-2:1998(E). Technical report. ISO/IEC; 1998.

[21] Armitage JW, Kellner MI. A conceptual schema for process definitions and models. Proceedings of the 3rd International Conference on the Software Process; 1994: IEEE Computer Society Press, 1994.

[22] Webby R, Becker U. Towards a logical schema integrating software process modeling and software measurement. ICSE97 Workshop on Process Modelling and Empirical Studies of Software Evolution; 1997 May 17-23; Boston MA, USA; IEEE Computer Society Press, 1997.

[23] Franch X, Ribó JM. PROMENADE: A modular approach to software process modelling and enaction. Research report. Barcelona, Spain: Software Department, Technical University of Catalonia; 1999. Report No.: LSI-99-13-R.

[24] Franch X, Ribó JM. Some reflexions in the modelling of software processes. Proceedings of the International Process Technology Workshop; 1999 Sep; Grenoble, France. 1999.

[25] Sutton SM Jr., Osterweil LJ. The design of a next-generation process language. Proceedings of the Joint 6th European Software Engineering Conference and the 5th ACM SIGSOFT Symposium on the foundation of Software Engineering; 1997 Sep 22-25; Zürich, Switzerland: Springer, 1997.

[26] World wide web consortium [online] 2001 [cited 2001 Nov 5]. Available from: http://www.w3.org/.

[27] Becker-Kornstaedt U, Reinert R. Using annotations in an electronic process handbook to systematically incorporate experience into processes. Technical report. Kaiserslautern, Germany: Fraunhofer Institute for Experimental Software Engineering; 2001 Jun. Report no.: 041.01/E

[28] Becker-Kornstaedt U, Hamann D, Kempkens R, Rösch P, Verlage M, Webby R, Zettel J. The Spearmint approach to process definition and process guidance. Technical report. Kaiserslautern, Germany: Fraunhofer Institute for Experimental Software Engineering; 1998 Jun. Report no.: 035.98/E.

[29] Zamli KZ, Lee PA.Taxonomy of process modeling languages. Proceedings of the ACS/IEEE International Conference on Computer Systems and Applications; 2001 Jun 25-29; Beirut, Lebanon. 2001.

[30] Bröckers A, Lott CM, Rombach HD, Verlage M. MVP-L language report (version 2). Technical Report 265/95. Kaiserslautern, Germany: Department of Computer Sciences, University of Kaiserslautern; 1995.

[31] National Institute of Standards and Technology (US). Integration definition for function modeling (IDEF0). Federal Information Processing Standards Publication 183; 1993.

[32] Object Management Group. OMG Unified modeling language specification (Version 1.3). Object Management Group; 2000.

[33] Franch X, Ribó JM. Using UML for modelling the static part of a software process. Proceedings of the Second Unified Language Conference (UML'99); 1999 Oct; Fort Collins COL, USA: LNCS 1723, 1999.

[34] Schader M, Korthaus A. Modeling business processes as part of the BOOSTER approach to business object-oriented system development based on UML. Proceedings of the 2nd International Enterprise Distributed Object Computing Workshop; 1998 Nov 2-5; San Diego CA, USA; 1998.

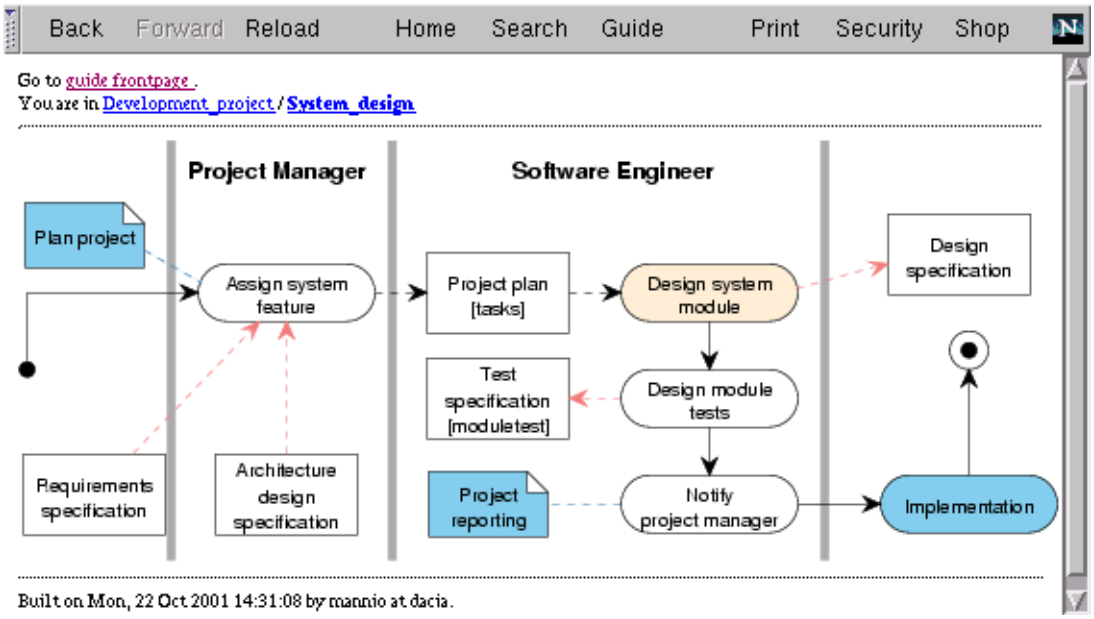
[35] Object Management Group. Software process engineering management: the unified process model (UPM). Initial submission. OMG document number ad/2000-05-05; 2000.

[36] Pons C, Giandini R, Baum G. Dependency relations between models in the Unified Process. Proceedings of the 10th International Workshop on Software Specification and Design; 2000 Nov 5-7; San Diego CA, USA; IEEE Computer Society Press, 2000.

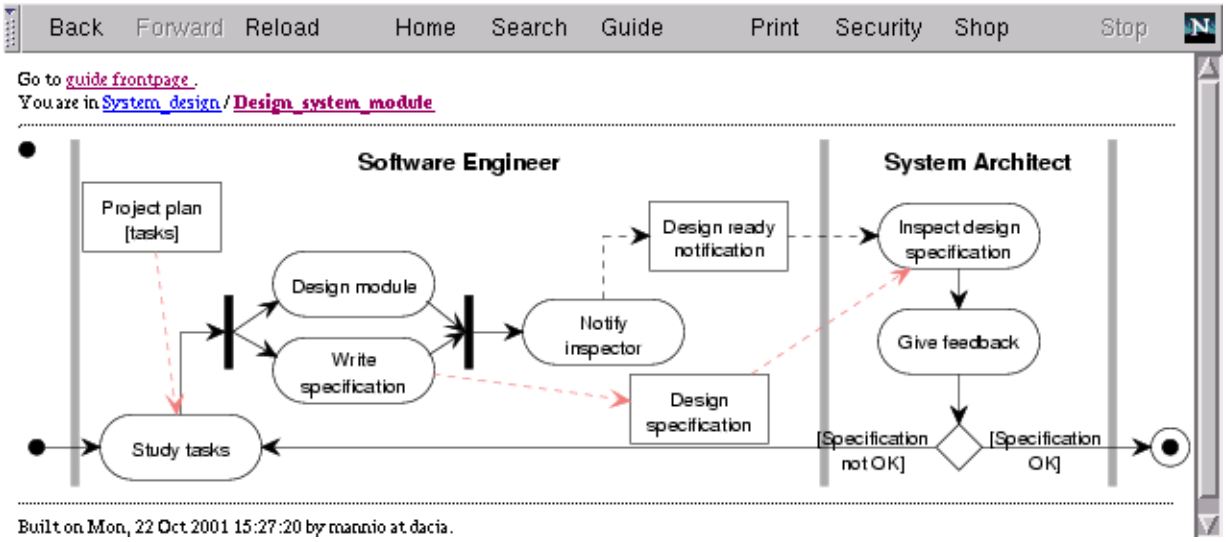
[37] Toolkit for conceptual modeling [online] 2001 [cited 2001 Nov 5]. Available from: http://wwwhome.cs.utwente.nl/~tcm.

Appendix 1. Example screens of the dynamic and static part

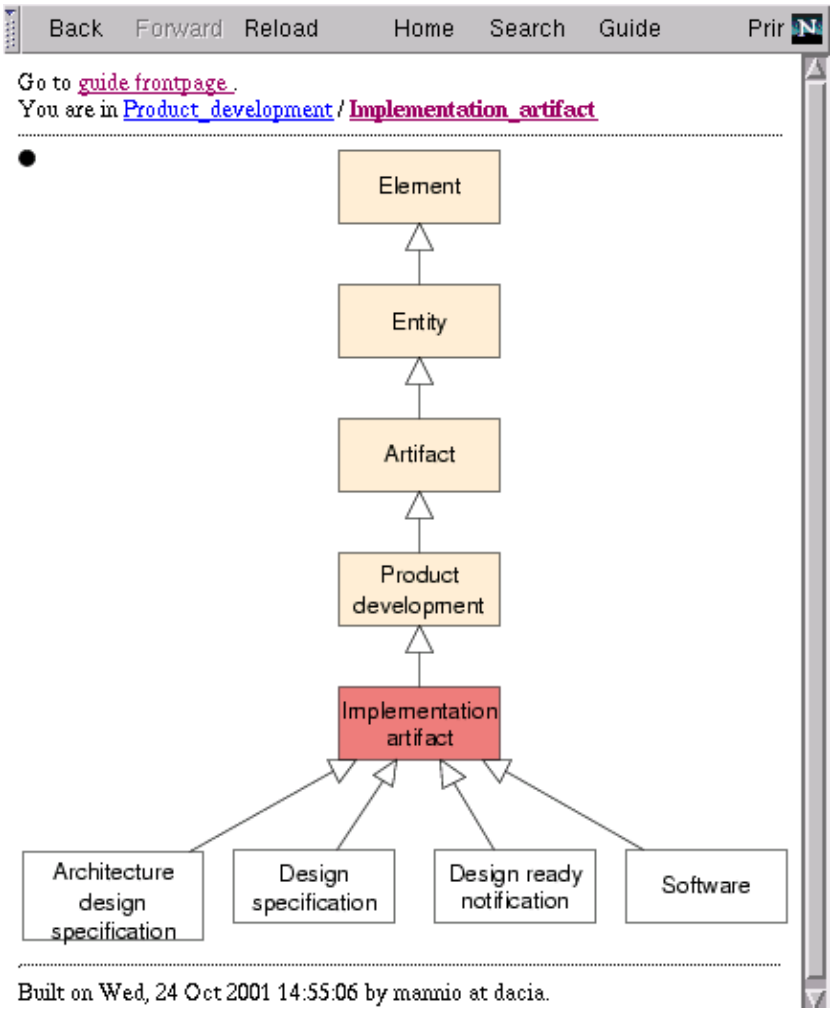
Dynamic view of the activity *System Design*



Dynamic view of the activity *Design System Module*

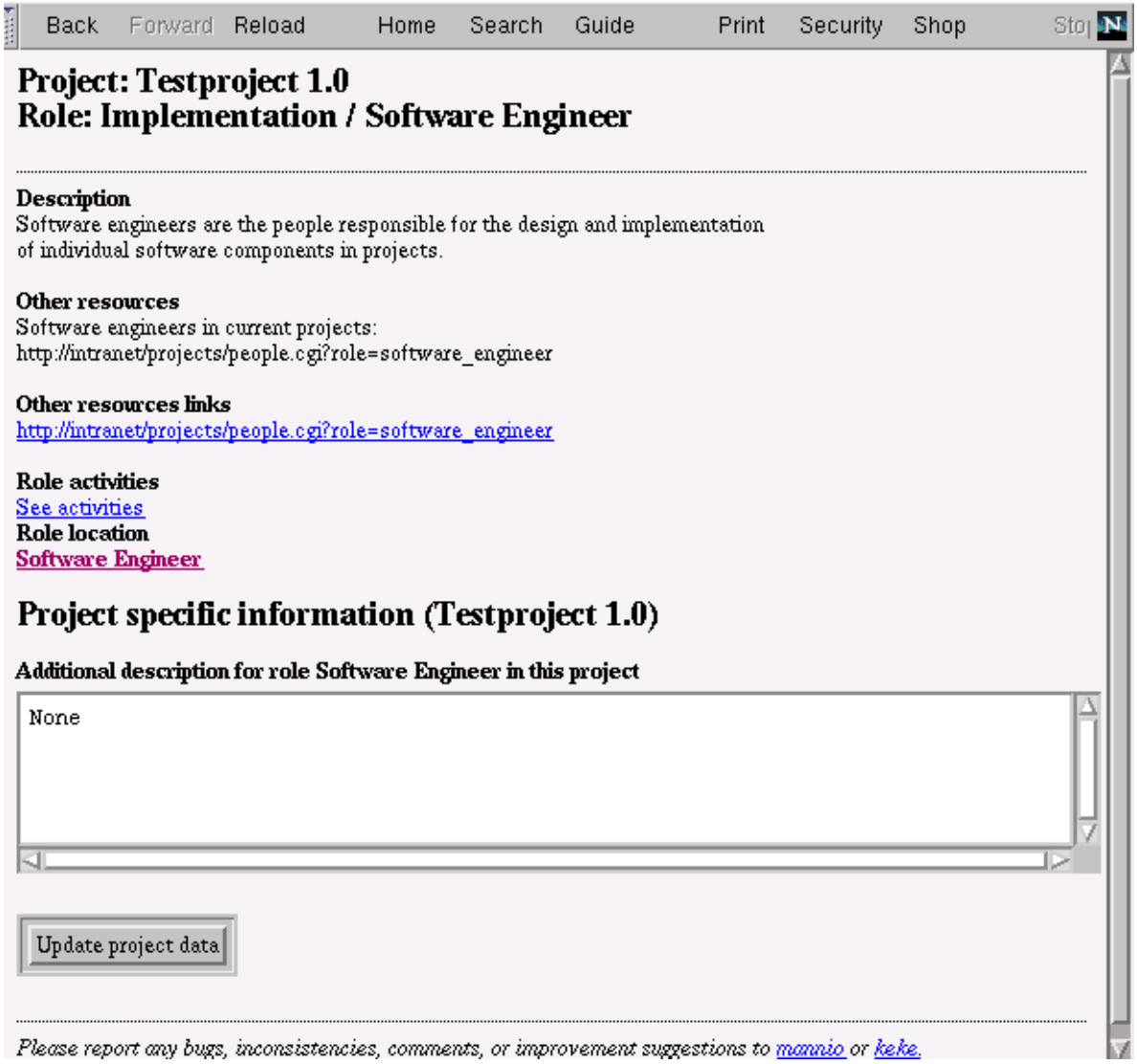


Static view of the *Design Specification* in *Implementation Artifact*

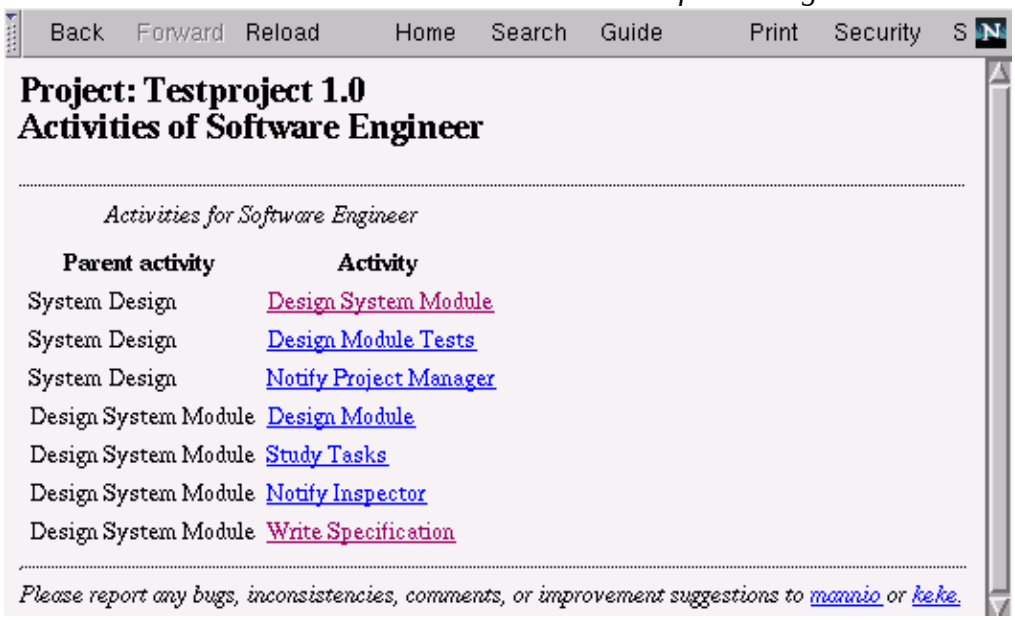


Appendix 2. Example screens of the textual part

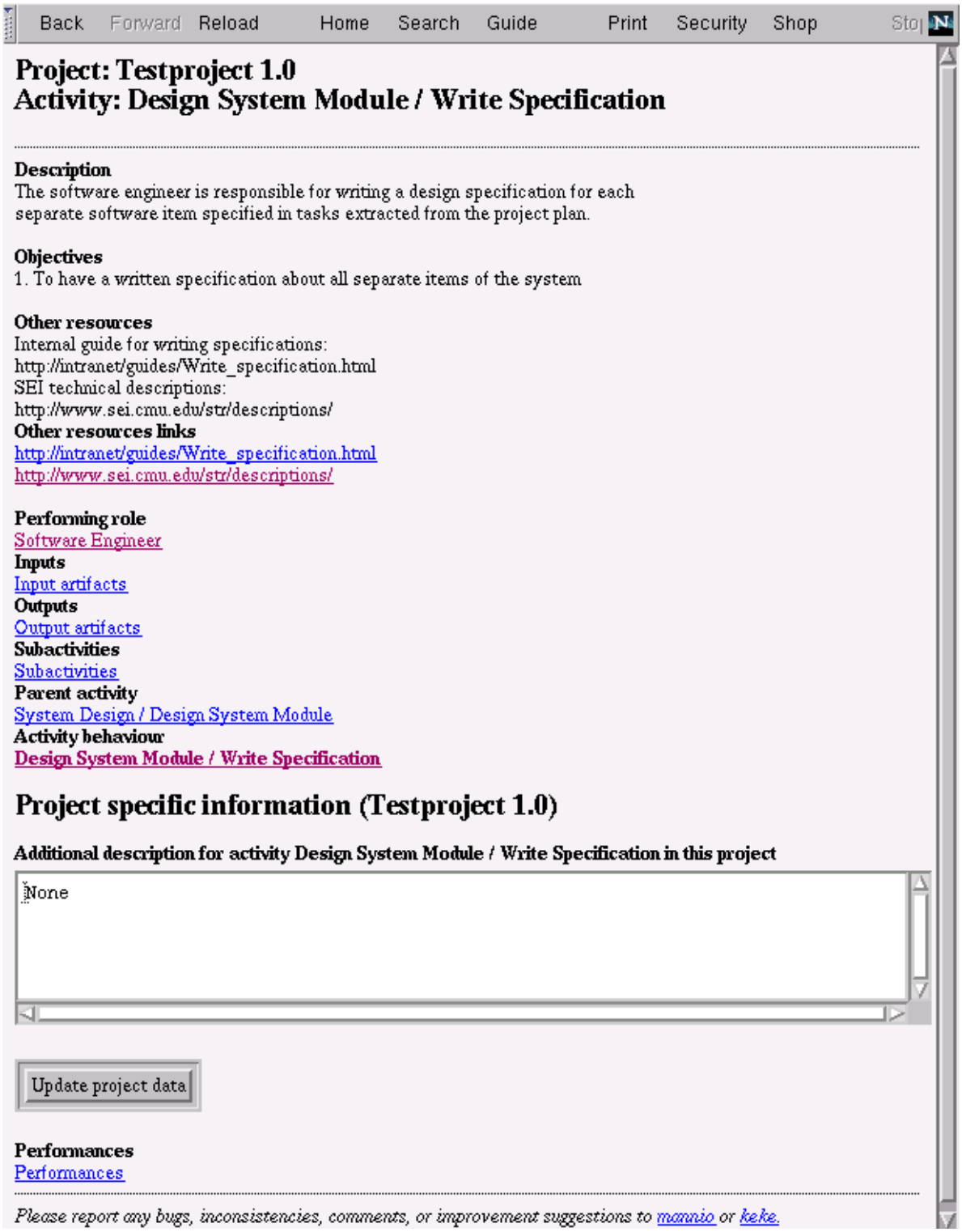
Textual view of the role *Software Engineer*



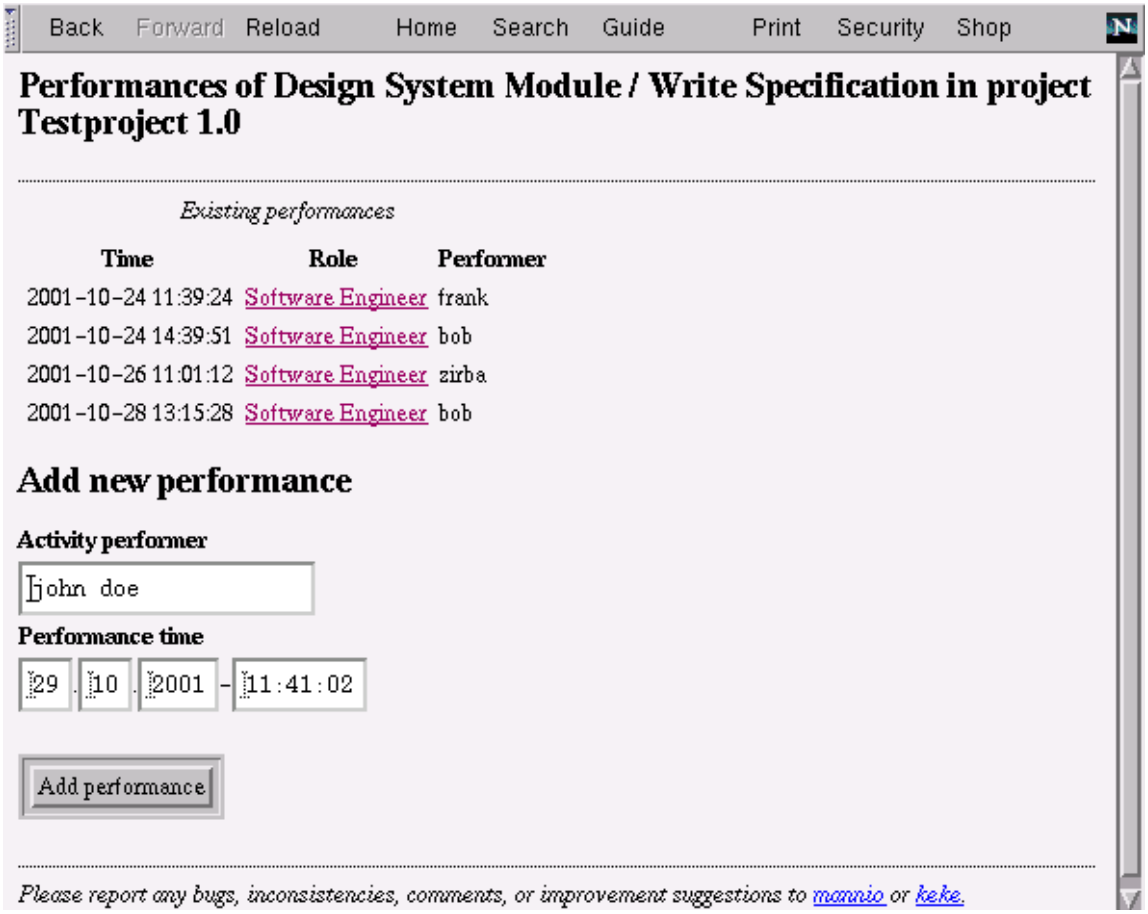
Textual view of the activities of the role *Software Engineer*



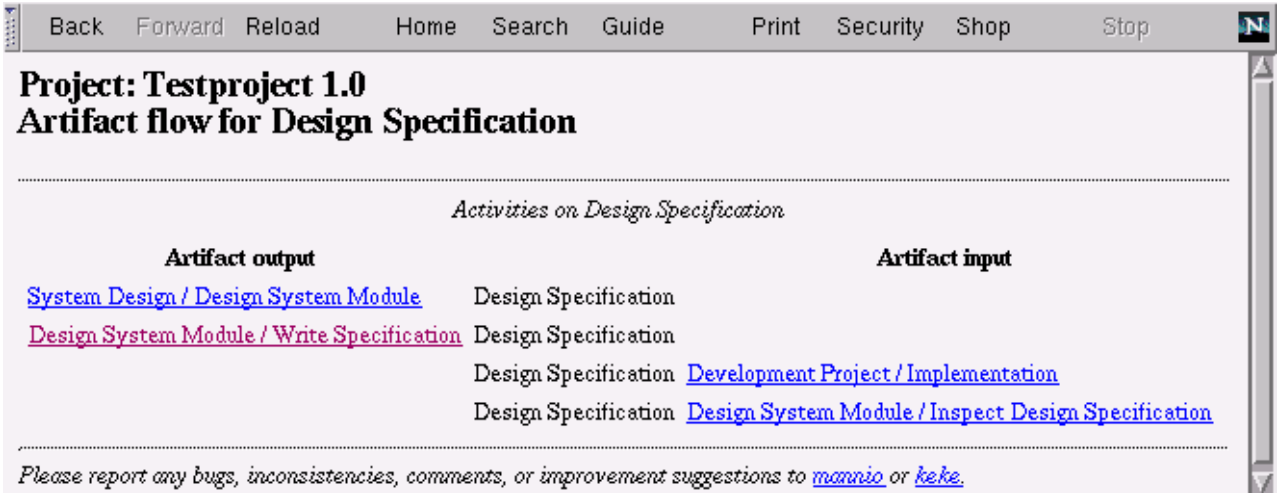
Textual view of the activity *Write Specification*



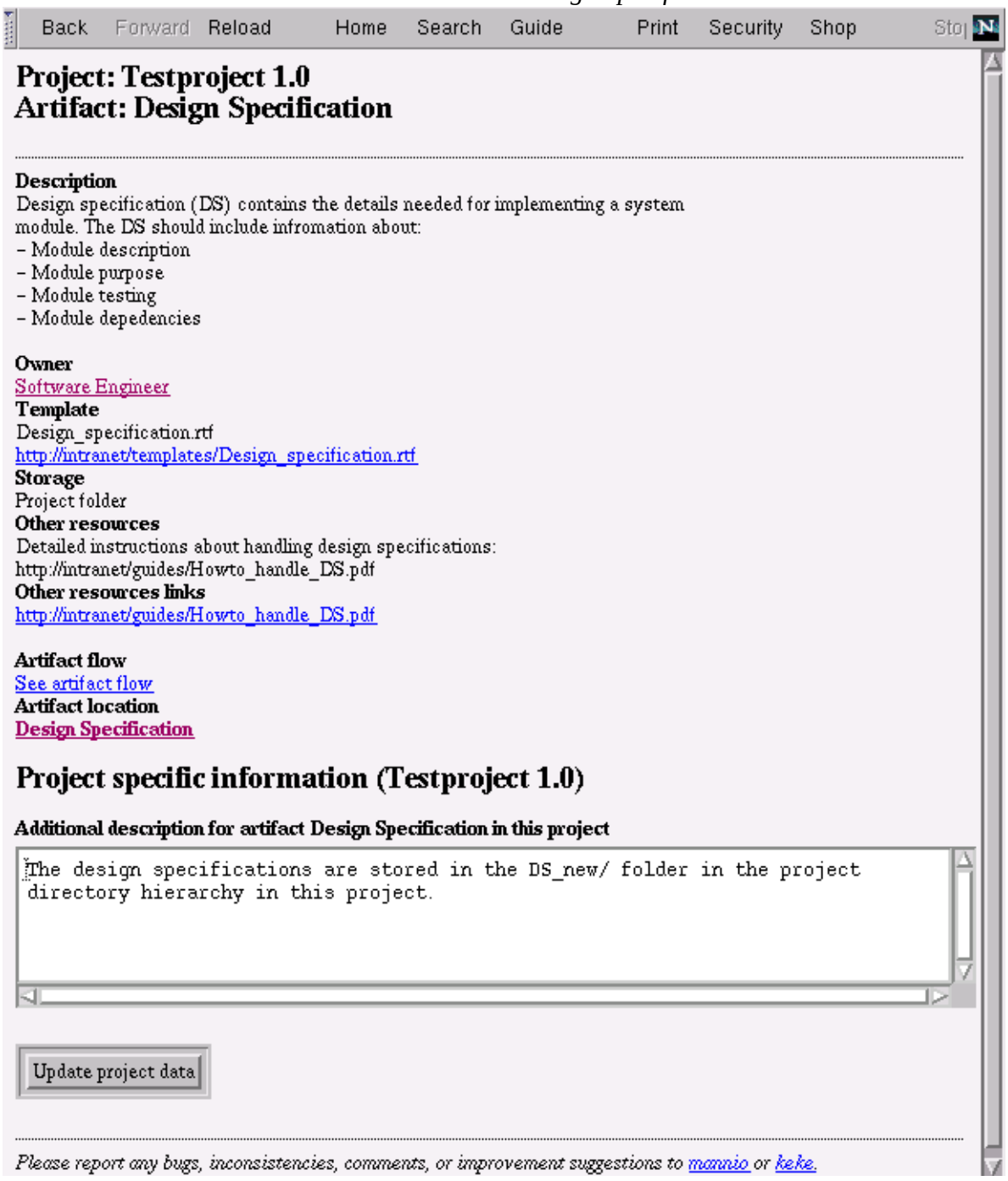
Textual view of the performances of the activity *Write Specification*



Textual view of the flow of the artifact *Design Specification*



Textual view of the artifact *Design Specification*



Back Forward Reload Home Search Guide Print Security Shop Stop

Project: Testproject 1.0
Artifact: Design Specification

Description
Design specification (DS) contains the details needed for implementing a system module. The DS should include infromation about:
- Module description
- Module purpose
- Module testing
- Module depedencies

Owner
Software Engineer
Template
Design_specification.rtf
http://intranet/templates/Design_specification.rtf
Storage
Project folder
Other resources
Detailed instructions about handling design specifications:
http://intranet/guides/Howto_handle_DS.pdf
Other resources links
http://intranet/guides/Howto_handle_DS.pdf

Artifact flow
See artifact flow
Artifact location
Design Specification

Project specific information (Testproject 1.0)

Additional description for artifact Design Specification in this project

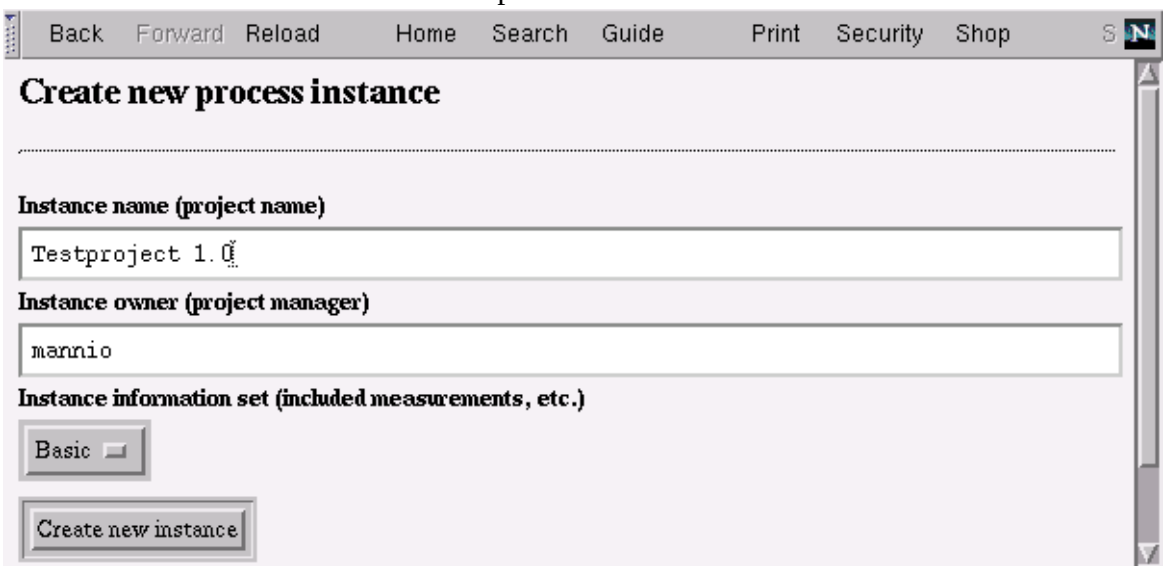
The design specifications are stored in the DS_new/ folder in the project directory hierarchy in this project.

Update project data

Please report any bugs, inconsistencies, comments, or improvement suggestions to mannio or keke.

Appendix 3. Example screens of the administrator guide

Create new process instance screen



Administrator textual view of the activity *Write Specification*