Framework of Quality Measurement in CASE Tool Based Software Development

Dissertation

zur Erlangung des akademischen Grades

Doktoringenieur (Dr.-Ing.)

angenommen durch die Fakultät für Informatik
der Otto-von-Guericke-Universität Magdeburg

von: M.Sc. in DKE **Hashem Yazbek**
geb. am 17.10.1974 in Homs, Syrien

Gutachter:
Prof. Dr.-Ing. habil. Reiner Dumke
Prof. Dr.-Ing. habil. Georg Paul
Prof. Dr. Juan José Cuadrado-Gallego

Magdeburg, den 28.09.2010

Contents

Acknowledgement

I would like to thank my supervisor, Prof. Dr. Reiner Dumke for the help throughout this work. Especially his patience and advice that he provided over the years.

A special thanks goes out to Prof. Georg Paul and Prof. Juan J. Cuadrado-Gallego for their efforts in reviewing and providing their expert opinions on the thesis at hand.

Magdeburg, September 2010
Hashem Yazbek

List of Figures

List of Tables

List of Abbreviations

AOP	Aspect-Oriented Programming
ANSI	American National Standards Institute
AOSE	Agent-Oriented Software Engineering
API	Application Programming Interface
ASG	Allen Systems Group
BPMN	Business Process Modeling Notation
BPEL	Business Process Execution Language
CASE	Computer-Aided Software Engineering
CAME	Computer Assisted software Measurement and Evaluation
CBSE	Component-Based Software Engineering
CBO	Coupling Between Objects
CFP	Cosmic Function Point
CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integration
COCOMO	Constructive Cost Model
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
COSMIC	the Common Software Measurement International Consortium
COTS	Commercial, Off-The-Shelf
CSV	Comma-Separated Values
CWM	Common Warehouse Metamodel
C&K	Chidamber and Kemerer
DACS	Data & Analysis Center for Software

DAML	DARPA Agent Markup Language
DBMS	Database Management System
DBS	Database System
DCE	Distributed Computing Environment
DDL	Data Definition Language
DML	Data Manipulating Language
DIT	Depth of Inheritance Tree
DPDS	DACS Productivity Center
DTD	Document Type Definition
EAI	Enterprise Application Integration
EBD	Event-Based Design
EF	Experience Factory
ERP	Enterprise Resource Planning
ETL	Extract Transform Load
FOD	Feature-Oriented Design
FP	Function Point
GQM	Goal-Question-Metric
GUI	Graphical User Interface
HTML	Hyper Text Markup Language
ICASE	Integrated Computer Aided Software Engineering
ICT	Information and Communication Technology
IDL	Interactive Data Language
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical & Electronics Engineers

IFPUG	International Function Point Users Group
IPD-CMM	Integrated Product Development CMM
IPSE	Integrated Project Support Environment
ISBSG	International Software Benchmarking Standard Group
ISO	International Standardization Organization
IT	Information Technology
J2EE	Java Platform, Enterprise Edition
KDSI	Kilo Delivered Source Instructions
LCOM	Lack of Cohesion in Methods
LOC	Lines of Code
MAS	Multi-Agent System
MP	Measurement Process
NASA	National Aeronautics & Space Administration
NOC	Number Of Children
ODMG	Object Data Management Group
OIM	Open Information Model
OLAP	Online Analytical Processing
OMG	Object Management Group
OMT	Object Modeling Technique
OOAD	Object-Oriented Analysis and Design
OOMO	Object-Oriented Measurement Ontology
OOSE	Object-Oriented Software Engineering
OWL	Ontology Web Language
PDF	Portable Document Formant

PNG	Portable Network Graphics
PSM	Practical Software Measurement
QIP	Quality Improvement Paradigm
QoS	Quality of Service
QuaD²	Quality Driven Design
QMP	Quality Model
RFC	Response For a Class
ROI	Return On Investment
RTF	Rich Text Format
SAM	Structured Analysis Methods
SANTA	Solution Architecture for N-Tier Applications
SD	Software Development Process
SDK	Software Development Kit
SDP	Software Development Project
SE	Software Engineering
SECM	Software Engineering Capability Model
SEMS	Software Engineering Measurement System
SLED	Software Lifecycle Empirical/Experience Database
SMDB	Software Measurement Data-Base
SML@b	Software Measurement Laboratory
SMP	Software Measurement Program
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SOMDB	Service-Oriented Measurement Database

SOSE	Service-Oriented Software Engineering
SP	Software Product
SPARC	Standards Planning and Requirements Committee
SPC	Statistical Process Control
SPICE	Software Process Improvement and Capability Determination
SQL	Structured Query Language
SR	Software Development Resources
TIM	Type Information Model
TLB	Type Libraries
UDDI	Universal Description, Discovery and Integration
UML	Unified Modeling Language
UREP	Universal Repository
URI	Unified Resource Identifier
W3C	Word Wide Web Committee
WBSE	Web-Based Software Engineering
WMC	Weighted Methods per Class
WS	Web Service
WS-CDL	Web Service Choreography Description Language
WSDL	Web Service Description Language
XML	eXtensible Markup Language
XSD	XML Schema Definition

1 Introduction

1.1 Motivation

The role of software in the daily life has an increasing effect in the information society worldwide. The quality of software is an essential aspect of the reliability and safety of our life in all domains such as working, living, travelling, shopping etc. The same is true for software development process itself as well. The quality of this process decides over success or failure in the human related fields characterized above.

Among the kernel elements of such software processes are CASE tools as computer-aided software engineering tools. The quality related ingredients of these tools play an important role in ensuring quality of software products.

Many process models and quality assurance (QA) approaches exist (such as CMMI, SPICE, ITIL etc.) to evaluate current situations and to improve these situations in a better direction. These approaches attempt to consider all IT aspects and summarize the essential experience mostly in a verbal manner. Typically, the role of CASE tools is implicitly considered in these process descriptions. Hence, the developer has no clear suggestions which kind of quality assurance based on direct software measurement should be applied or should be necessary in the different system development context.

Otherwise, CASE-based software development is one of the great important components in the software process in the different paradigms and methodologies. Quality techniques and methods in CASE tools are basically for the system quality. This thesis considers the quality assurance in the CASE-based software development explicitly founded on the exact determination of the quality situation as software measurement and evaluation.

1.2 Research Questions

CASE tools based software development is one of the engineering characteristics in the field of software engineering. The experience-based measurement and evaluation determine the other essential side of engineering and helps to motivate, consider and implement quality aspects in a quality assurance manner.

Therefore, the following research questions in the domain of CASE-based software development can be identified:

- What are the general software process characteristics and which process levels can be built in order to evaluate and ensure software quality assurance?
- How the CASE tool component is embedded in the software process methodologies, phases and artefacts?
- What are the quality assurance levels in the software process that are determined by CASE tools?
- Which kind of improvement would be achieved by different metrics supported in CASE tools?

- How can it be determined what the next step of process improvement in the CASE-based software development is?

Finding meaningful answers of the most of theses questions is the intention of this thesis and leads to framework as a measurement based concept for the determination of the QA level of the CASE-based software development and for the suggestion of the (next steps) QA improvements of this software process basics.

1.3 Thesis Structure

In this first chapter we give a short introduction of the necessity and meaningfulness of software quality determination in the CASE tool based software development area.

The general basics of software development processes, their kinds of modelling, process evaluations and process oriented (improvement) standards and approaches would be considered in the chapter two.

In the third chapter we describe the software engineering field of CASE tools. Especially the CASE-based processes are characterized and their activities are shown with examples.

The different kinds of software process evaluations including the CASE-based development are summarized in chapter four. Here we consider the variants of measurement and evaluation levels such as nomination (with criteria and indicators), classification (using ordinal based evaluation) and measurement using metrics and measures for the software development phases and artefacts.

In the fifth chapter we describe our framework in order to evaluate the quality measurement and assurance level in a given practical situation. Furthermore, we define a measurement level determination that can be used in order to identify quality measurement improvements explicitly.

The application of our framework and the validation are the contents of the sixth chapter in this thesis.

Finally, some conclusions and next steps of investigations are discussed in the seventh and final chapter.

2 Software Process Descriptions and Models

2.1 Software Process Characteristics

The software process is one of the central components in the software engineering field of research, practice and application. Especially, the managerial foundations play an essential role in the nature of software processes. The following figure 1 shows some categories of managerial foundations of software engineering defined by Wang ([Wang 2000], see also [Boehm 2000b] and [Royce 2005]).

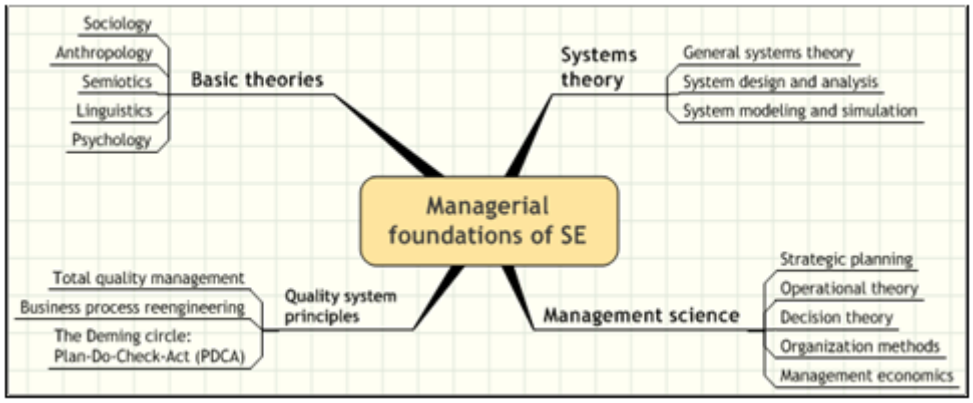


Figure 1: Managerial foundations of software engineering

In following we will give some definitions in order to clarify the management and controlling background of the software processes considered in this Thesis.

The first (basic) definition of software processes was presented by Wang [Wang 2000] and characterizes the general software engineering process.

> *"The software engineering process is a set of sequential practices that are functionally coherent and reusable for software engineering organization, implementation, and management. It is usually referred to as the **software process**, or simply the process."*

An appropriate method for software process handling consists of creating and applying *process models*. Different implications for this kind of abstraction are shown in the following figure 2 based on [Deek 2005].

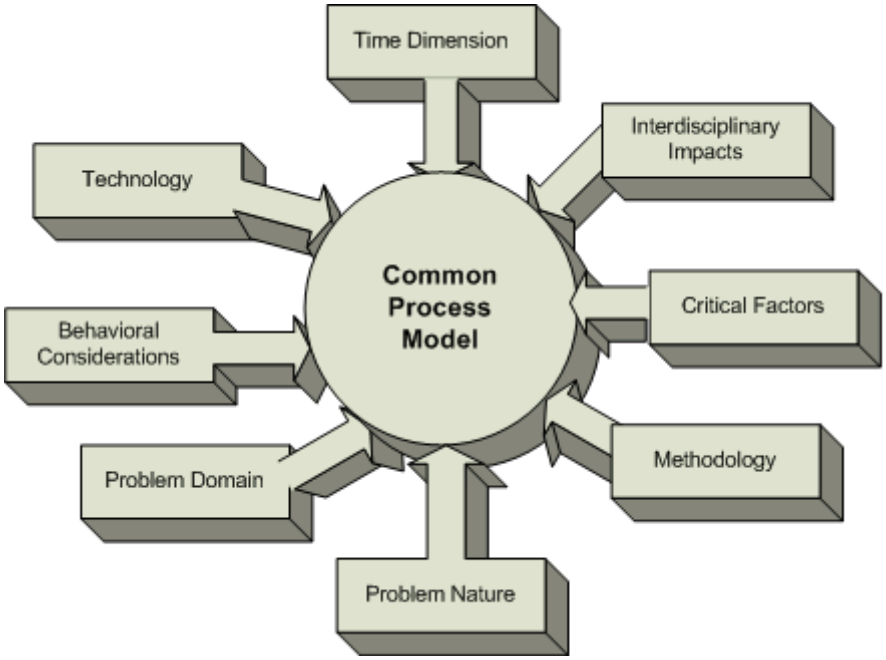


Figure 2: Context diagram for software process models

Software engineering processes exist in different kinds of context such as different technologies or systems like multimedia software engineering [Chang 2000] or Web engineering [Dumke 2003]. Software processes include a set of involvements which forms the special characteristics and directions of such operationalities. Therefore, we will use some appropriate definitions by Wang and King [Wang 2000].

> *"A **practice** is an activity or a state in a software engineering process that carries out a specific task of the process."*
>
> *"A **process** is a set of sequential practices (or base process activities (BPAs)) which are functionally coherent and reusable for software project organization, implementation, and management."*
>
> *"A **process category** is a set of processes that are functionally coherent and reusable in an aspect of software engineering."*
>
> *"A **process subsystem** is a set of process categories that are functionally coherent and reusable in a main part of software engineering.*
>
> *"A **process system** is an entire set of structured software processes described by a process model."*

Considering the different process domains, we can establish the following kinds of processes cited from [Wang 2000].

> *"A **domain of a process model** is a set of ranges of functional coverage that a process model specifies at different levels of the process taxonomy."*

*"**Organization processes** are processes that belong to a top level administrative process subsystem, which are practiced above project level within a software development organization."*

*"**Development processes** are processes that belong to a technical process subsystem, which regulate the development activities in software design, implementation, and maintenance."*

*"**Management processes** are processes that belong to a supporting process subsystem, which control the development processes by means of resource, staff, schedule, and quality."*

Note that the software process could change in a dynamic environment itself. Therefore, a so-called *software engineering process group (SEPG)* must be established in order to maintain the change management. A SEPG (see [Kandt 2006]): "obtains support from all levels of management, facilitates process assessments, helps line managers define and set expectations for software processes, maintains collaborative working relationships with practitioners, arranges for software process improvement training, monitors and reports on the progress of specific software process improvements efforts, creates and maintains process definitions and a process database, and consults with projects on software development processes."

In order to characterize the different approaches and structures of process models we will use the helpful definitions by Wang [Wang 2000] as given below:

*"A **process model** is a process of a model system that describes process organization, categorization, hierarchy, interrelationship, and tailor-ability."*

*"An **empirical process model** is a model that defines an organized and benchmarked software process system and best practices captured and elicited from the software industry."*

*"A **formal process model** is a model that describes the structure and methodology of a software process system with an algorithmic approach or by an abstractive process description language."*

*"A **descriptive process model** is a model that describes 'what to do' according to a certain software process system."*

*"A **prescriptive process model** is a model that describes 'how to do' according to a certain software process systems."*

Especially, the software process as defined by *NASA Software Engineering Laboratory* consists of a series of phases [Donzelli 2006]:

- *Requirements:* requirements changes, requirement increments

- *Specification:* specification changes, specification increments, specification correction reports

- *High-level design:* high-level design changes, high-level design increments, high-level design correction reports

- *Low-level design:* low-level design changes, low-level design increments, low-level design corrections reports

- *Code:* code changes, code increments, code correction reports

- *System-tested code :* system-tested code changes, system-tested code increments, system-tested code corrections reports

- *Acceptance-tested code:* acceptance-tested code changes, acceptance-tested code increments (final SW product)

In general we can establish the following four categories of processes in the software development ([Kulpa 2003], [SEI 2002]): the project management processes, the process management processes, the engineering processes, and the support processes. Based on process models like the CMMI we can evaluate main activities shown in the Figure 3.

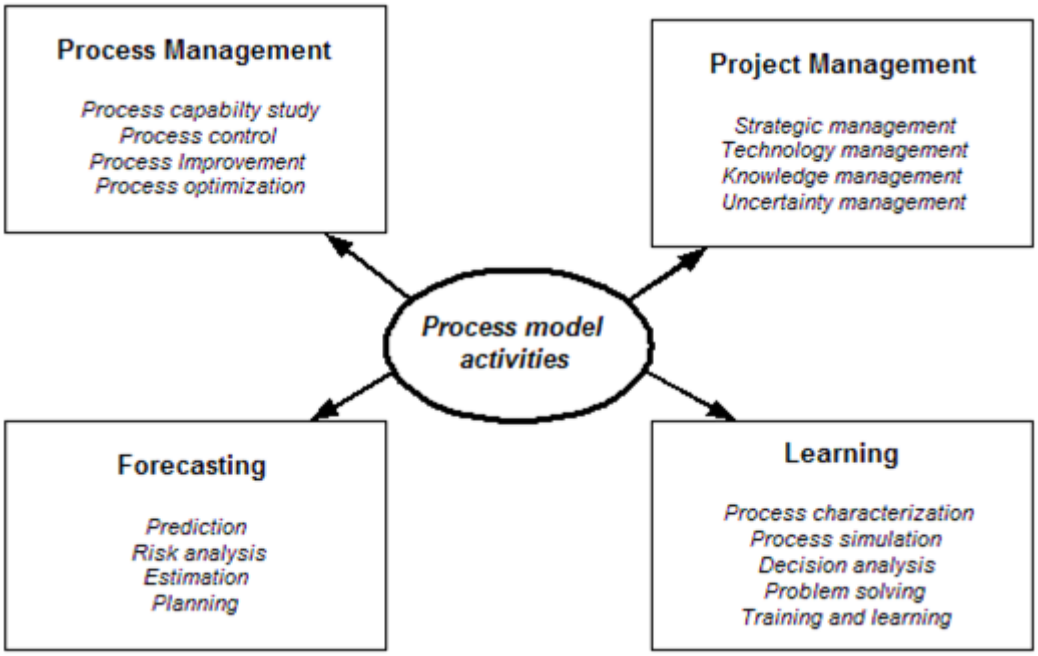


Figure 3: Activities supporting by process models

Finally we will cite some definitions which are helpful in order to prepare some intentions or model for software process measurement and evaluations (also chosen from [Wang 2000]).

> *"**Software process establishment** is a systematic procedure to select and implement a process system by model tailoring, extension, and/or adaptation techniques."*

*"**Software process assessment (SPA)** is a systematic procedure to investigate the existence, adequacy, and performance of an implemented process system against a model, standard, or benchmark."*

*"**Process capability determination** is a systematic procedure to derive a capability level for a process, and/or organization based on the evidence of existence, adequacy, and performance of the required practices defined in a software engineering process system."*

*"**Software process improvement (SPI)** is a systematic procedure to improve the performance of an existing process system by changing the current processes or updating new processes in order to correct or avoid problems identified in the old process system by means of a process assessment."*

Based on these aspects of evaluation are defined the following concepts, methods and models of process evaluations (see [Wang 2000]).

*"A **generic model of the software development organization** is a high-level process model of an organization which is designed to regulate the functionality and interactions between the roles of developers, managers, and customers by a software engineering process system."*

*"A **process reference model** is an established, validated, and proven software engineering process model that consists of a comprehensive set of software processes and reflects the benchmarked best practices in the software industry."*

*"A **process capability model (PCM)** is a measurement scale of software process capability for quantitatively evaluating the existence, adequacy, effectiveness, and compatibility of a process."*

*"A **process capability scope** is an aggregation of all the performing ratings, such as existence, adequacy, and effectiveness, of the practice which belong to the process."*

*"A **project process capability scope** is an aggregation of all process capability levels of processes conducted in a project."*

*"An **organization process capability scope** is an aggregation of the process capability levels from a number of sampled projects carried out in a software development organization."*

*"A **process capability determination model** is an operational model that specifies how to apply the process capability scales to measure a given process system described by a process model."*

*"A **process improvement model (PIM)** is an operational model that provides guidance for improving a process system's capability by changing, updating, or*

enhancing existing processes based on the findings provided in a process assessment."

*"A **model-based process improvement model** is an operational model that describes process improvement based on model- or standard-based assessment results."*

*"A **benchmark-based process improvement model** is an operational model that describes process improvement methods based on benchmark-based assessment results."*

A general software process improvement cycle is defined by Lepasaar et al. [Lepasaar 2001] in the following manner:

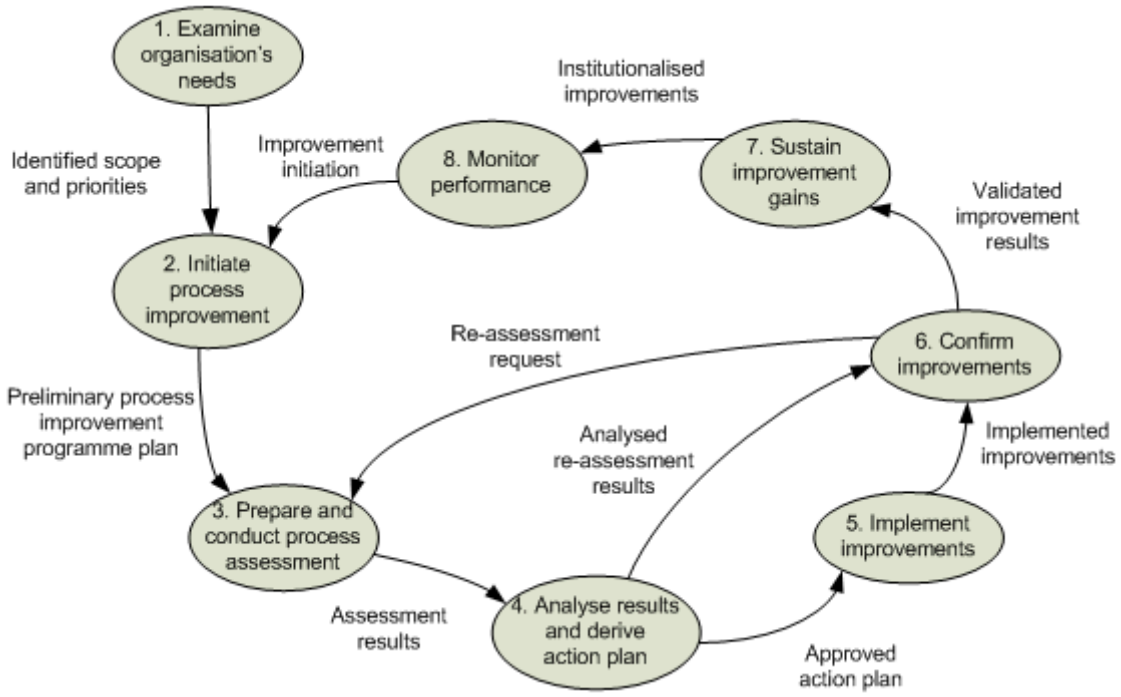


Figure 4: The software process improvement cycle by Lepasaar et al.

In this thesis we will characterize a ***software project*** as an *instance of a software process*. Hence, we must consider the detailed aspects of project management in the process domain also. Typical project management phases are project definition, project planning, and project control which involves the process measurement, communication and the corrective actions [Verzuh 2005].

2.2 Software Process Modelling

The main intention of software engineering is to create/produce software products with a high quality for the customers [Dumke 2005]. A software system or ***software product*** SP is

developed by the software process SD and is based on the supporting resources SR. At first, we will define the software product as a (software) system:

$$SP = (M_{SP}, R_{SP}) = (\{programs, documentations\}, R_{SP}) \quad (2.1)$$

where the two sets are divided in the following elements or components (without achieving completeness)

$programs \subseteq \{sourceCode, objectCode, template, macro, library, script, plugIn, setup, demo\}$

$documentations = \{userManual, referenceManual, developmentDocumentation\}$

and R_{SP} describes the set of the relations over the SP elements.

The given subsets could be described in following

$developmentDocumentation = \{documentationElements, productRequirements, productSpecification, productDesign, implementationDescription\}$

$documentationElements \subseteq \{model, chart, architecture, diagram, estimation, review, audit, verificationScript, testCase, testScript, pseudoCode, extensionDescription, qualityReport\}$

$productRequirements = systemRequirement \subseteq \{functionalRequirements, qualityRequirements, platformRequirements, processRequirements\}$

$functionalRequirements \subseteq \{execution, mapping, information, construction, controlling, communication, learning, resolution, cooperation, coordination\}$[1]

$qualityRequirements \subseteq \{functionality, reliability, efficiency, usability, maintainability, portability\}$[2]

$platformRequirements \subseteq \{systemSoftware, hardwareComponent, hardwareInfrastructure,peripheralDevice, host\}$

$processRequirements \subseteq \{developmentMethod, resources, cost, timeline, milestone, criticalPath, developmentManagement, lifecycleModel\}$

Here, we can define a software product as a software system as following ([Chung 2000], [Dumke 2003], [Horn 2002], [Maciaszek 2001], [Marciniak 1994], [Mikkelsen 1997])

$SE\text{-}SoftwareSystems \subseteq \{informationSystem, constructionSystem, embeddedSystem, communicationSystem, distributedSystem, knowledgeBasedSystem\}$ (2.2)

[1] The kind of the functional requirements depends on the kind of the software system which we characterize later.
[2] This set of quality characteristics is related to the ISO 9126 product quality standard.

Relations involving general aspects of software products are [Messerschmitt 2003]: *software is different, software is ubiquitous, software makes our environment interactive, software is important, software is about people, software can be better, software industry is undergoing radical changes, creating software is social, software is sophisticated and complex,* and *software can be tamed.* We can derive some of the examples of the relations in R_{SP} as given next:

- The process of the software testing on some software product components:

$$r_{SP}^{(test)} \in R_{SP}: sourceCode \times verificationScript \times testScript \rightarrow testDescription$$

- The elements of the product design considering the necessary components:

$$r_{SP}^{(design)} \in R_{SP}: architecture \times review \times template \times library \times pseudoCode \rightarrow productDesign$$

- A special kind of a programming technique could be defined as following:

$$r_{SP}^{(programmingTechnique)} \in R_{SP}: template \times macro \rightarrow sourceCode$$

- The process of the software implementation could be defined as following:

$$r_{SP}^{(implementation)} \in R_{SP}: coding \times unitTest \times integrationTest \rightarrow implementation$$

The following figure by [Messerschmitt 2003] shows different roles of technology in software applications or products.

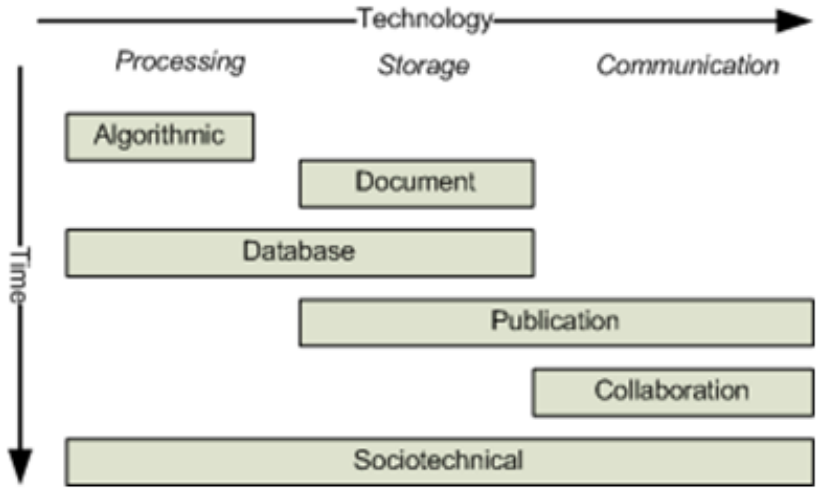


Figure 5: Components of the software product

The following figure summarizes the components and elements of the software product described in the text above.

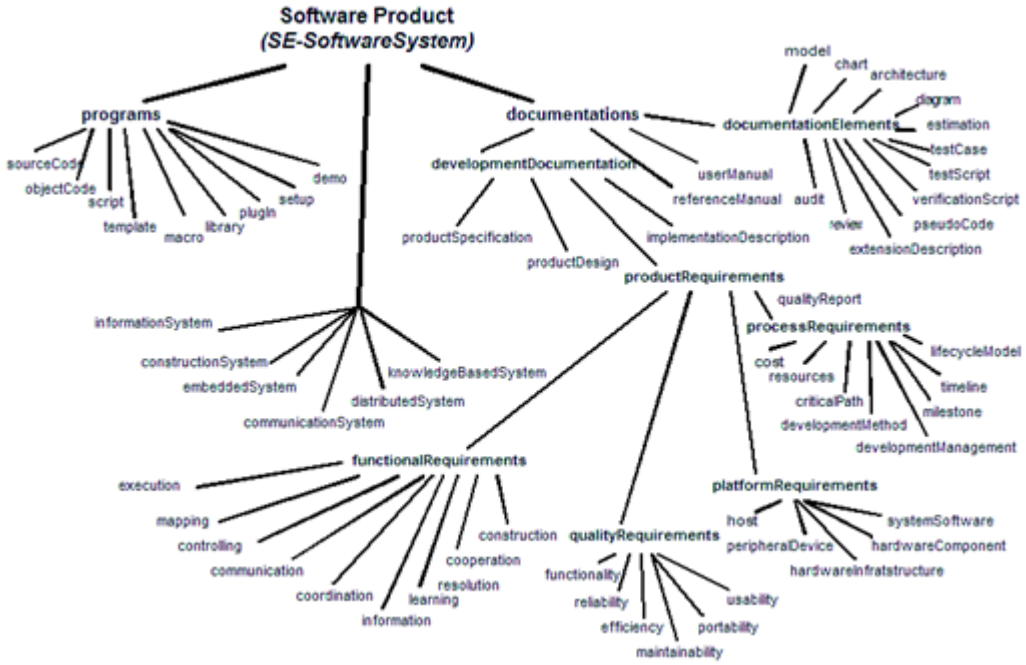


Figure 6: Components of the software product

Now, we will define the ***software development process*** SD itself (note, that the concrete software process is known as *software project*). So, we can define the software process SD as following (including the essential details of every development component)

$$SD = (M_{SD}, R_{SD}) =$$
$$(\{developmentMethods, lifecycle, softwareManagement\} \cup M_{SR}, R_{SD}) \qquad (2.3)$$

$$developmentMethods \subseteq \{formalMethods, informalMethods\} = SE\text{-}Methods$$

$$formalMethods \in \{CSP, LOTOS, SDL, VDM, Z\}$$

We can see a plenty of "classical" informal development methods as structured methods (*SAM)*. Actually, the informal methods are based on the objects (*OOSE),* the components (*CBSE),* or the agents (*AOSE).* Therefore, we can define

$$informalMethods \in \{SAM, OOSE, CBSE, AOSE\}$$

and especially

$$SAM \in \{SA/SD, Jackson, Warnier, HIPO\}$$

$$OOSE \in \{UML, OMT, OOD, OOSE, RDD, Fusion, HOOD, OOSA\}$$

$$CBSE \in \{DCOM, EJB, CURE, B\text{-}COTS, SanFrancisco\}$$

$$AOSE \in \{AAII, AUML, DESIRE, IMPACT, MAS, MaSE, MASSIVE, SODA\}$$

The life cycle aspects could be explained by the following descriptions

lifecycle = {lifecyclePhase, lifecycleModel}

lifecyclePhase $\in$ *{problemDefinition*[3]*, requirementAnalysis, specification, design, implementation, acceptanceTest, delivering}*

lifecycleModel $\in$ *{waterfallModel, Vmodel, evolutionaryDevelopment, prototyping, incrementalDevelopment, spiralModel, …, winWinModel}*

Finally, the software management component of the M_{SD} could be described in the following manner

softwareManagement = developmentManagement $\subseteq$ *{projectManagement, qualityManagement, configurationManagement }*

Note that the software development process could be addressed as a special kind of a software system. Hence, we can make the following characterization

$$\boldsymbol{SD}_{informationSystem} \neq \boldsymbol{SD}_{embeddedSystem} \neq \boldsymbol{SD}_{distributedSystem} \neq \boldsymbol{SD}_{knowledgeBased\ System} \quad (2.4)$$

Furthermore, some of the examples of the relations in R_{SD} could be derived in the following manner

- The process of building an appropriate life cycle model:

$$r_{SD}^{(lifecycle)} \in R_{SD}: lifecyclePhase_{i_1} \times \ldots \times lifecyclePhase_{i_n} \rightarrow lifecycleModel$$

- The defining of software development based on the waterfall model:

$$r_{SD}^{(waterfall)} \in R_{SD}: problemDefinition \times specification \times design \times implementation \times acceptanceTest \rightarrow waterfallModel$$

- The definition of software development based on the V model:

$$r_{SD}^{(V\,mod\,el)} \in R_{SD}: (problemDefinition, softwareApplication) \times (specification, acceptanceTest) \times (design, integrationTest), \times (coding, unitTest) \rightarrow Vmodel$$

- The characterization of the tool-based software development based on UML:

$$r_{SD}^{(UMLdev)} \in R_{SD}: UML \times developmentEnvironment_{UML} \times systemOfMeasures_{UML} \times experience_{UML} \times standard_{UML} \rightarrow developmentInfrastructure_{UML}$$

These descriptions lead us to the following general model of the software engineering considering the three dimensions of the software methodology, the software technology and the related application domains or kinds of systems.

[3] Problem definition is a verbal form of the defined system or product requirements.

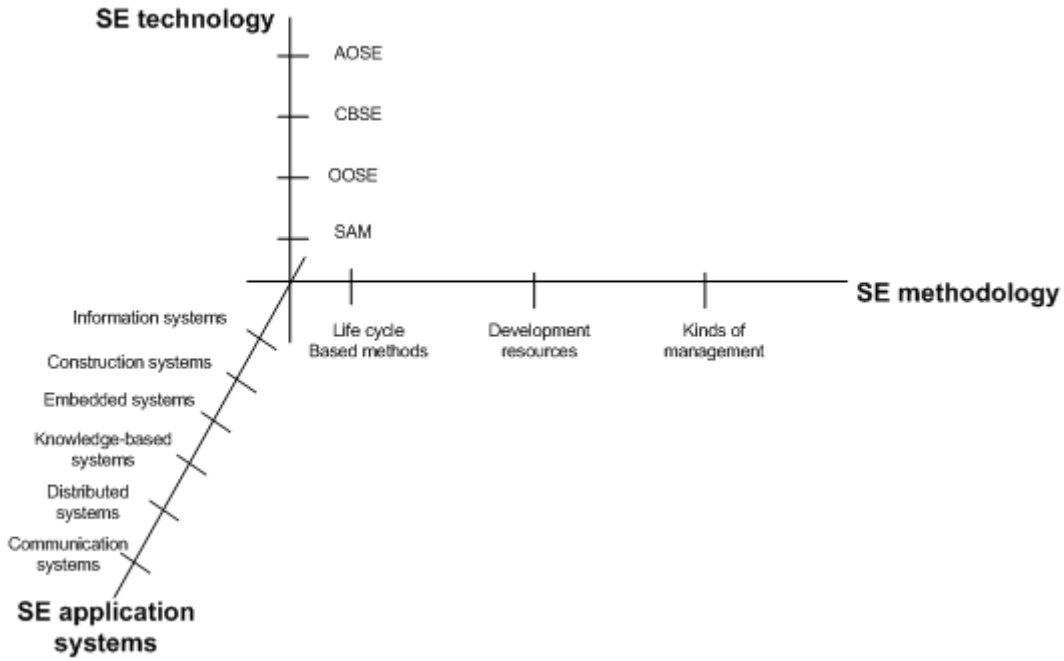


Figure 7: Dimensions of the software engineering

Finally, the components and aspects of the software development process are shown in the following Figure 8.

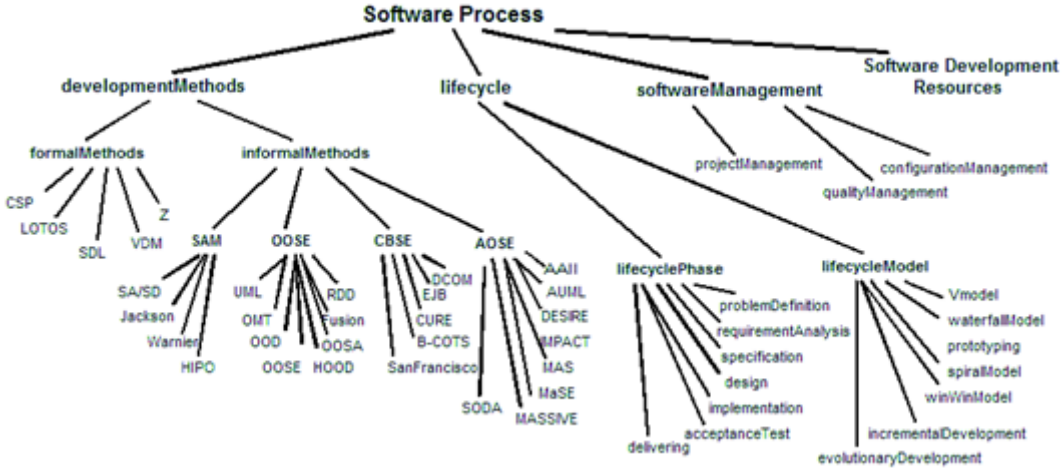


Figure 8: Components of the software process

In order to develop a software product we need resources such as developers, CASE tools and variants of hardware. Therefore, we define the ***software development resources*** SR as following

$$SR = (M_{SR}, R_{SR}) = (\{personnelResources, softwareResources, platformResources\}, R_{SR}) \quad (2.5)$$

where the software resources play a dual role in the software development: as a part of the final system (as COTS or software components) and as the support for the development (as CASE or integrated CASE as ICASE). We continue our definition as follows

$$softwareResources = \{COTS\} \cup \{ICASE\}$$

$$ICASE = CASE \cup CARE \cup CAME$$

where *CARE* stands for computer-aided reengineering and *CAME* means computer-assisted measurement and evaluation tools. Considering the WWW aspects and possibilities for software development infrastructures based on CASE environments, the set of CASE tools could be divided as following

$$CASE_{infrastructure} = \{ (\{UpperCASE\} \cup \{LowerCASE\})_{environment} \}$$

Further, we can define

$$UpperCASE = \{modellingTool, searchTool, documentationTool, diagramTool, simulationTool, benchmarkingTool, communicationTool\}$$

$$LowerCASE = \{assetLibrary, programmingEnvironment, programGenerator, compiler, debugger, analysisTool, configurationTool\}$$

Especially, we can describe the following incomplete list of personnel resources as

$$personnelResources = \{analyst, designer, developer, acquisitor, reviewer, programmer, tester, administrator, qualityEngineer, systemProgrammer, chiefProgrammer, customer\}$$

$$SE\text{-}Communities = \{personnelDevelopmentResources, ITadministration, softwareUser, computerSociety\}$$

Accordingly, some of the examples of the relations in R_{SR} could be derived in the following manner

- The process of building an appropriate development environment:
  $r_{SR}^{(devEnv)} \in R_{SR}: ICASE \times platformResources \rightarrow developmentEnvironment$

- The defining of software developer teams for the agile development:
  $r_{SR}^{(agile)} \in R_{SR}: programmer \times programmer \times customer \rightarrow agileDevelopmentTeam$

Now, we summarize different elements and components of the resources as the basics of the software development and maintenance in the following figure.

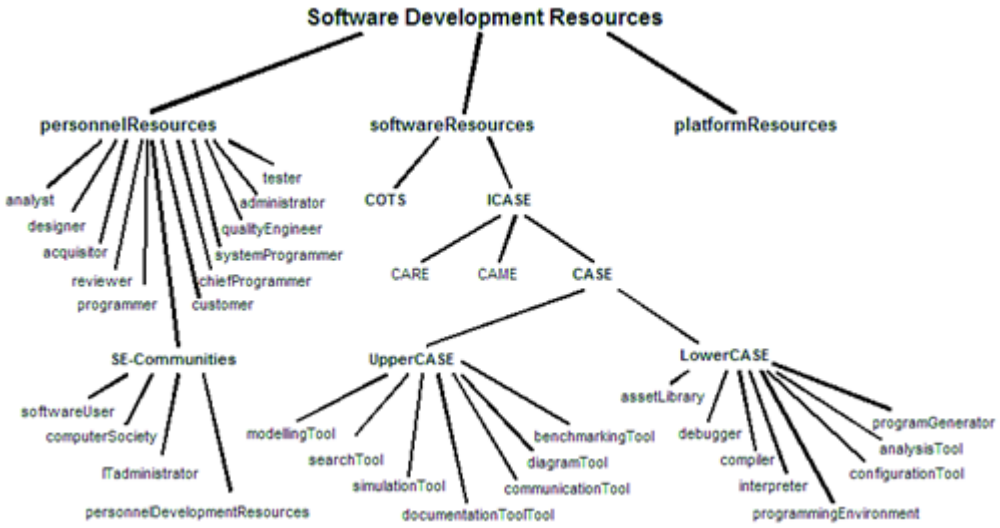


Figure 9: Components of the software development resources

The different aspects and characteristics of the ***software maintenance*** are summarized by the following formulas [April 2005]

$$\boldsymbol{SM} = (M_{SM}, R_{SM}) = (\{maintenanceTasks, maintenanceResources\} \cup SP, R_{SM}) \qquad (2.6)$$

where

maintenanceTasks = {extension, adaptation, correction, improvement, prevention}

maintenanceResources = ICASE ∪ {maintenancePersonnel, maintenancePlatform}

maintenancePersonnel = {maintainer, analyst, developer, customer, user}

Accordingly, some of the examples of the relations in R_{SM} could be derived in the following manner

- The process of building the extension activity of the maintenance:

$$r_{SM}^{(extension)} \in R_{SM}: \boldsymbol{SP} \times functionalRequirements \rightarrow \boldsymbol{SP}^{(extended)}$$

- The defining of software correction:

$$r_{SM}^{(correction)} \in R_{SM}: \boldsymbol{SP} \times qualityRequirements \rightarrow \boldsymbol{SP}^{(corrected)}$$

- The defining of software adaptation:

$$r_{SM}^{(adaptation)} \in R_{SM}: \boldsymbol{SP} \times platformRequirements \rightarrow \boldsymbol{SP}^{(adapted)}$$

- The defining of software improvement:

$$r_{SM}^{(perform)} \in R_{SM}: \boldsymbol{SP} \times performanceRequirements \rightarrow \boldsymbol{SP}^{(improved)}$$

- The defining of software prevention:

$$r_{SM}^{(prevention)} \in R_{SM} : \boldsymbol{SP} \times preventionRequirements \rightarrow \boldsymbol{SP}^{(modified)}$$

- The characterization of a special kind of software maintenance as remote maintenance:

$$r_{SM}^{(remoteMaint)} \in R_{SM} : ICASE_{remote} \times maintenanceTasks \times maintenancePersonnel \rightarrow \text{remoteMaintenance}$$

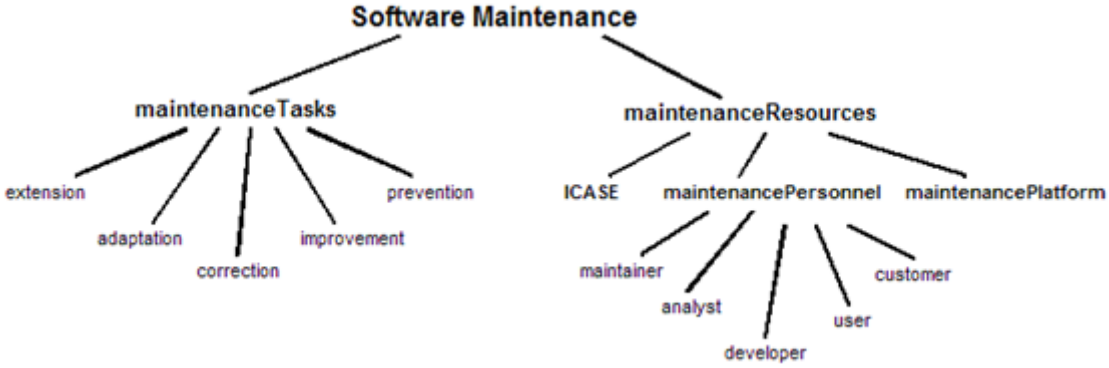


Figure 10: Components of the software maintenance

After the software development, the software product goes in two directions: first (in the original sense of a software product) to the ***software application*** $\boldsymbol{SA}$**,** and second in the software maintenance $\boldsymbol{SM}$. We define the different aspects in the following

$$\boldsymbol{SA} = (M_{SA}, R_{SA}) = (\{applicationTasks, applicationResources, applicationDomain\} \cup M_{SP}, R_{SA}) \quad (2.7)$$

where

$$applicationTask \in \{delivering, operation, migration, conversion, replacement\}$$

$$applicationResources = \{applicationPlatform, applicationPersonnel, applicationDocuments\}$$

$$applicationPersonnel \subseteq \{customer, user, operator, administrator, consultant, trainer\}$$

$$applicationDocument \subseteq \{userManual, trainingGuideline, acquisitionPlan, setup, damageDocument, troubleReport\}$$

$$applicationDomain \subseteq \{organisationalDocument, law, contract, directive, rightDocument\}$$

Based on these definitions, some of the examples of the relations in R_{SA} could be derived in the following manner

- The process of the first introduction of the software product as deliveration:

$$r_{SA}^{(deliver)} \in R_{SA}: \boldsymbol{SP} \times trainer \times applicationPersonnel \times applicationPlatform \rightarrow deliveration$$

- The defining of software migration based on essential requirements:

$$r_{SA}^{(migration)} \in R_{SA}: productExtension \times \boldsymbol{SP} \times migrationPersonnel \rightarrow migration$$

- The characterization of software operation:

$$r_{SA}^{(operation)} \in R_{SA}: applicationPersonnel \times applicationPlatform \times \boldsymbol{SP} \times user \rightarrow operation$$

- The defining of the outsourcing of the software operation by extern IT contractors:

$$r_{SA}^{(outsourcing)} \in R_{SA}: systemInputs \times contractors \times systemFeedback \rightarrow outsourcing$$

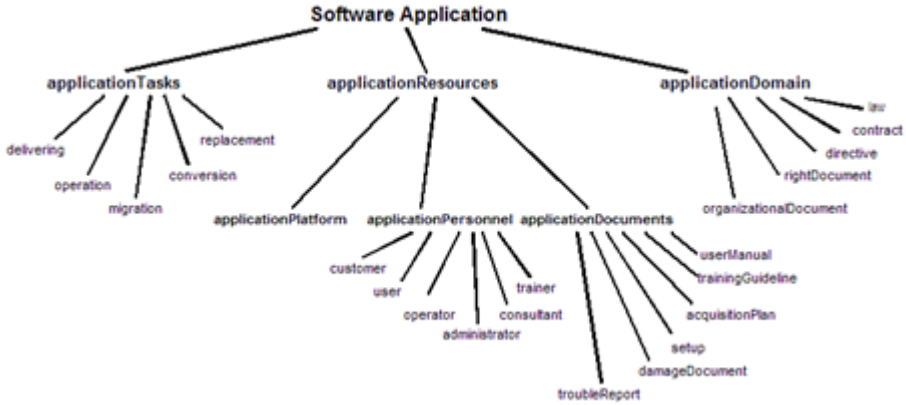


Figure 11: Components of the software product application

This formal concept demonstrates the wide area of the software process artefacts and involvements which must be considered in order to analyse, measure, evaluate, improve and control software development and maintenance.

2.3 Software Process Improvement and Evaluation Approaches

Examples of software process improvement standards and approaches are summarized as following (described in [Emam 1998], [Garcia 2005], [Royce 1998] and [Wang 2000])

- ***ISO 9001:2000*** as a standard for process assessment and certification comparable to other business areas and industries.
- ***TickIT*** inform the developer about the actual quality issues and best practices considering the process improvement.
- ***ISO 12207*** defines the software life cycle processes for a general point of view and involves the process quality implicitly.
- ***ISO 15504*** is also known as SPICE (Software Process Improvement and Capability Determination) and was described shortly later in this Thesis.
- ***Bootstrap*** process evaluation is based on the assessment process, the process model (including the evaluation as *incomplete, performed, managed, established, predictable* and *optimising*), the questionnaires and the scoring, rating and result presentation .

- ***SEI-CMMI*** is the well-known capability maturity model which *integrated* some of other process improvement standards and approaches (see below).
- ***Trillium*** is a Canadian initiative for software process improvement and provides to initiate and guide a continuous improvement program.
- ***EFQM*** as European Foundation of Quality Management considers soft factors like customer satisfaction, policy and strategy, business results, motivation, and leading in order to evaluate the process effectiveness and success.

The following semantic network shows some classical approaches in the software process evaluation without any comments [Ferguson 1998] or [Ebert 2007].

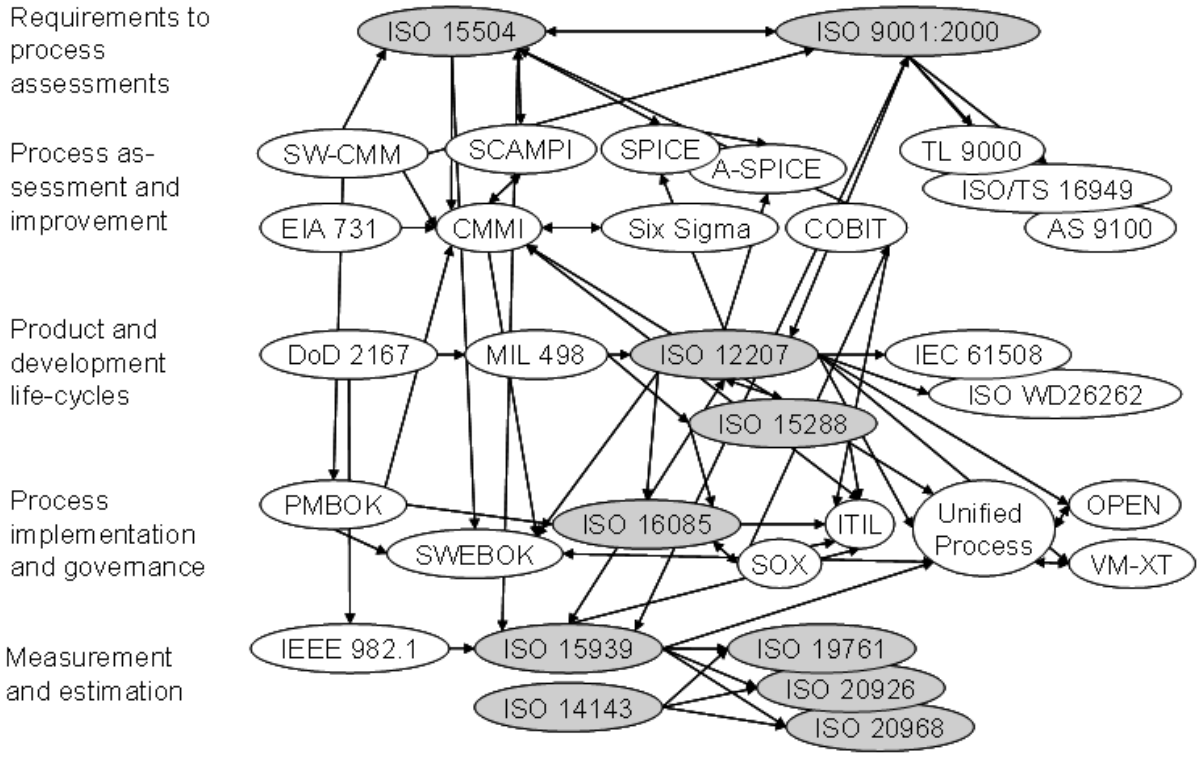


Figure 12: Dependencies of software process evaluation methods and standards

2.3.1 General Maturity Models

Based on the idea of process improvement, a lot of *maturity models (MM)* were defined and implemented in order to classify different aspects of software products, processes and resources. Some of these maturity evaluation approaches are described in the following table (see [April 2005] and [Braungarten 2005])

Model	Description	Model	Description
PEMM	Performance Engineering MM	**CM3**	Configuration Management MM
TMM	Testing Maturity Model	**ACMM**	IT Architecture Capability MM
ITS-CMM	IT Service Capability MM	**OMMM**	Outsourcing Management MM

iCMM	Integrated CMM	**PM2**	Project Management Process Model
TCMM	Trusted CMM	**IMM**	Internet MM
SSE-CMM	System Security Engineering CMM	**IMM**	Information MM
OPM3	Organizational Project Management MM	**PMMM**	Program Management MM
OMM	Operations MM	**PMMM**	Project Management MM
M-CMM	Measurement MM	**IPMM**	Information Process MM
SAMM	Self-Assessment MM	**CPMM**	Change Proficiency MM
UMM	Usability MM	**ASTMM**	Automated Software Testing MM
ECM2	E-Learning CMM	**LM3**	Learning Management MM
WSMM	Web Services MM	**ISM3**	Information Security Management MM
eGMM	e-Government MM	**TMM**	Team MM
EVM3	Earned Value Management MM	**SRE-MM**	Software Reliability Engineering MM
WMM	Website MM	**EDMMM**	Enterprise Data Management MM
DMMM	Data Management MM	**S3MM**	Software Maintenance MM

Table 1: Chosen maturity models

The following figure summarizes some of these maturity models and chosen improvement models in a layer structure of software process evaluation.

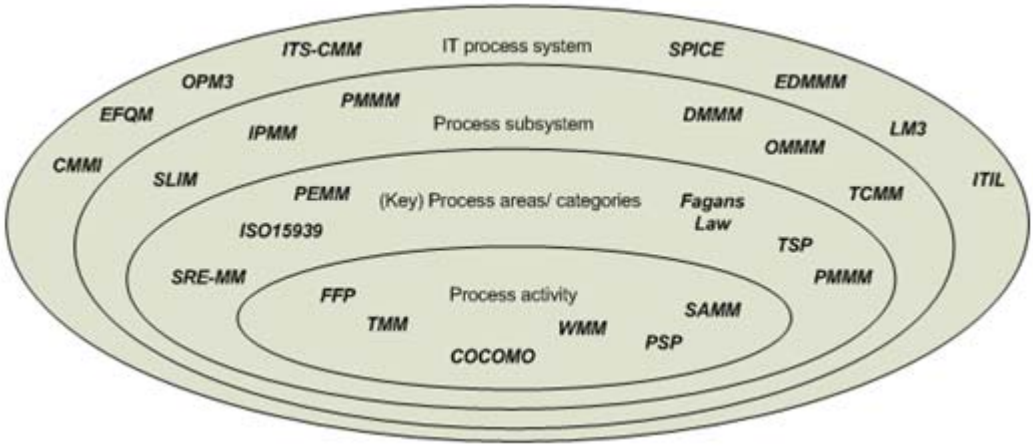


Figure 13: Overview of chosen process maturity and improvement models

In following we will consider some of the essential approaches of software process evaluation and improvement.

2.3.2 The CMMI Approach

CMMI stands for ***Capability Maturity Model Integration*** and is an initiative for changing the general intention of an *assessment view* based on the "classical" CMM or ISO 9000 to an *improvement view* integrating the System Engineering CMM (SE-CMM), the Software Acquisition Capability Maturity Model (SA-CMM), the Integrated Product Development Team Model (IDP-CMM), the System Engineering Capability Assessment Model (SECAM), the Systems Engineering Capability Model (SECM), and basic ideas of the new versions of the ISO 9001 and 15504 [Chrissis 2003]. The CMMI is structured in the five maturity levels, the considered process areas, the specific goals (SG) and generic goals (GG), the common features and the specific practices (SP) and generic practices (GP). The *process areas* are defined as follows [Kulpa 2003]:

> *"The Process Area is a group of practices or activities performed collectively to achieve a specific objective."*

Such objectives could be the part of requirements management at the level 2, the requirements development at the maturity level 3 or the quantitative project management at the level 4. The difference between the "specific" and the "general" goals, practices or process area is the reasoning in the special aspects or areas which are considered in opposition to the general IT or company-wide analysis or improvement. There are four common features:

- The commitment to perform (CO)
- The ability to perform (AB)
- The directing implementation (DI)
- The verifying implementation (VE).

The CO is shown through senior management commitment, the AB is shown through the training personnel, the DI is demonstrated by managing configurations, and the VE is demonstrated via objectively evaluating adherence and by reviewing status with higher-level management. The following Figure 14 shows the general relationships between the different components of the CMMI approach.

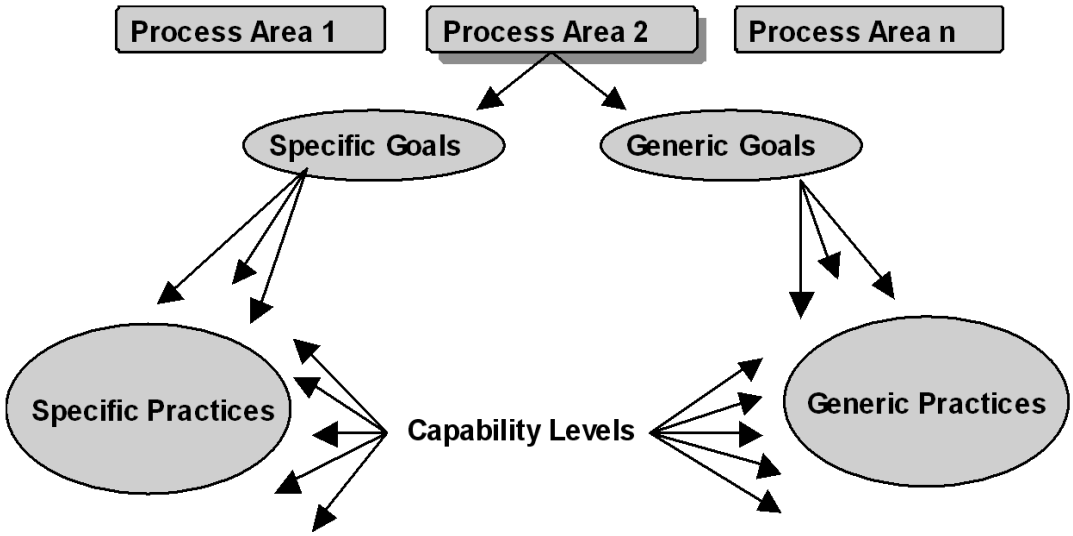


Figure 14: The CMMI model components

The CMMI gives us some guidance as to what is a required component, an expected component, and simply informative. There are six capability levels (but five maturity levels), designated by the numbers 0 through 5 [SEI 2002], including the following process areas:

0. *Incomplete:* -
1. *Performed:* best practices;
2. *Managed:* requirements management, project planning, project monitoring and control, supplier agreement management, **measurement and analysis**, process and product quality assurance;
3. *Defined:* requirements development, technical solution, product integration, **verification**, **validation**, organizational process focus, organizational process definition, organizational training, integrated project management, risk management, integrated teaming, integrated supplier management, **decision analysis and resolution**, organizational environment for integration;
4. *Quantitatively Managed:* organizational process performance, **quantitative project management**;
5. *Optimizing:* organizational innovation and deployment, **causal analysis and resolution**.

Addressing the basics of the project management CMMI considers the following components for the management of the IT processes [SEI 2002]:

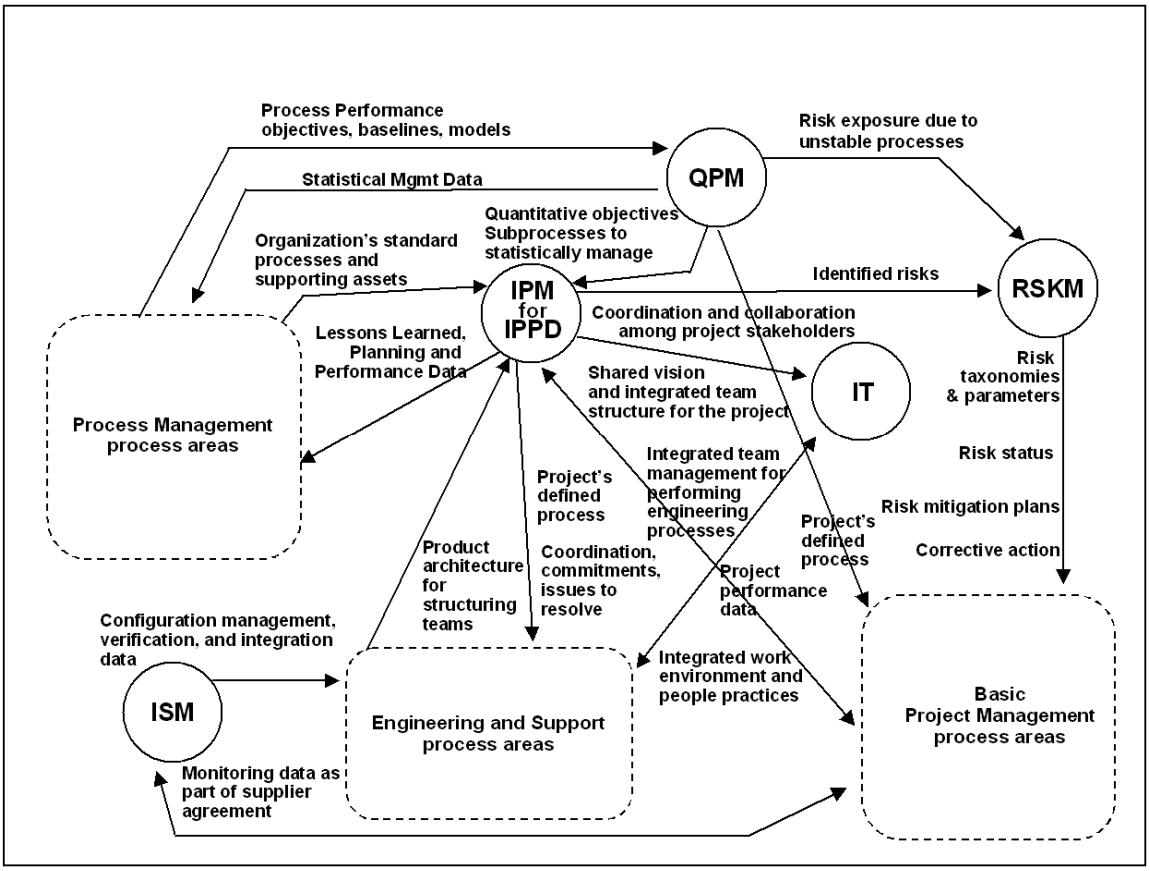


Figure 15: The CMMI project management process areas

where QPM stands for Quantitative Project Management, IPM for Integrated Project Management, IPPD for Integrated Product and Process Development, RSKM for risk management, and ISM for Integrated Supplier Management.

In order to manage the software process quantitatively, CMMI defines a set of example metrics. Some of these appropriate software measurement intentions are [SEI 2002]:

- *Examples of quality and process performance attributes for which needs and priorities might be identified include the following:* Functionality, Reliability, Maintainability, Usability, Duration, Predictability, Timeliness, and Accuracy;

- *Examples of quality attributes for which objectives might be written include the following:* Mean time between failures, Critical resource utilization, Number and severity of defects in the released product, Number and severity of customer complaints concerning the provided service;

- *Examples of process performance attributes for which objectives might be written include the following:* Percentage of defects removed by product verification activities (perhaps by type of verification, such as peer reviews and testing), Defect escape rates, Number and density of defects (by severity) found during the first year following product delivery (or start of service), Cycle time, Percentage of rework time;

- *Examples of sources for objectives include the following:* Requirements, Organization's quality and process-performance objectives, Customer's quality and process-performance objectives Business objectives, Discussions with customers and potential customers, Market surveys;

- *Examples of sources for criteria used in selecting sub processes include the following:* Customer requirements related to quality and process performance, Quality and process-performance objectives established by the customer, Quality and process-performance objectives established by the organization, Organization's performance baselines and models, Stable performance of the sub process on other projects, Laws and regulations;

- *Examples of product and process attributes include the following:* Defect density, Cycle time, Test coverage;

- *Example sources of the risks include the following:* Inadequate stability and capability data in the organization's measurement repository, Sub processes having inadequate performance or capability, Suppliers not achieving their quality and process-performance objectives, Lack of visibility into supplier capability, Inaccuracies in the organization's process performance models for predicting future performance, Deficiencies in predicted process performance (estimated progress), Other identified risks associated with identified deficiencies;

- *Examples of actions that can be taken to address deficiencies in achieving the project's objectives include the following:* Changing quality or process performance objectives so that they are within the expected range of the project's defined process, Improving the implementation of the project's defined process so as to reduce its normal variability (reducing variability may bring the project's performance within the objectives without having to move the mean), Adopting new sub processes and technologies that have the potential for satisfying the objectives and managing the associated risks, Identifying the risk and risk mitigation strategies for the deficiencies, Terminating the project;

- *Examples of sub process measures include the following:* Requirements volatility, Ratios of estimated to measured values of the planning parameters (e.g., size, cost, and schedule), Coverage and efficiency of peer reviews, Test coverage and efficiency, Effectiveness of training (e.g., percent of planned training completed and test scores), Reliability, Percentage of the total defects inserted or found in the different phases of the project life cycle Percentage of the total effort expended in the different phases of the project life cycle;

- *Sources of anomalous patterns of variation may include the following:* Lack of process compliance, Undistinguished influences of multiple underlying sub processes on the data, Ordering or timing of activities within the sub process, Uncontrolled inputs to the sub process, Environmental changes during sub process execution, Schedule pressure, Inappropriate sampling or grouping of data;

- *Examples of criteria for determining whether data are comparable include the following:* Product lines, Application domain, Work product and task attributes (e.g., size of product), Size of project;

- *Examples of where the natural bounds are calculated include the following:* Control charts, Confidence intervals (for parameters of distributions), Prediction intervals (for future outcomes);

- *Examples of techniques for analyzing the reasons for special causes of variation include the following:* Cause-and-effect (fishbone) diagrams, Designed experiments, Control charts (applied to sub process inputs or to lower level sub processes), Sub grouping (analyzing the same data segregated into smaller groups based on an understanding of how the sub process was implemented facilitates isolation of special causes);

- *Examples of when the natural bounds may need to be recalculated include the following:* There are incremental improvements to the sub process, New tools are deployed for the sub process, A new sub process is deployed, The collected measures suggest that the sub process mean has permanently shifted or the sub process variation has permanently changed;

- *Examples of actions that can be taken when a selected sub process' performance does not satisfy its objectives include the following:* Changing quality and process-performance objectives so that they are within the sub process' process capability, Improving the implementation of the existing sub process so as to reduce its normal variability (reducing variability may bring the natural bounds within the objectives without having to move the mean), Adopting new process elements and sub processes and technologies that have the potential for satisfying the objectives and managing the associated risks, Identifying risks and risk mitigation strategies for each sub process' process capability deficiency;

- *Examples of other resources provided include the following tools:* System dynamics models, Automated test-coverage analyzers, Statistical process and quality control packages, Statistical analysis packages

- *Examples of training topics include the following:* Process modelling and analysis, Process measurement data selection, definition, and collection;

- *Examples of work products placed under configuration management include the following:* Sub processes to be included in the project's defined process, Operational definitions of the measures, their collection points in the sub processes, and how the integrity of the measures will be determined, Collected measures;

- *Examples of activities for stakeholder involvement include the following:* Establishing project objectives, Resolving issues among the project's quality and process-performance objectives, Appraising performance of the selected sub

processes, Identifying and managing the risks in achieving the project's quality and process-performance objectives, Identifying what corrective action should be taken;

- *Examples of measures used in monitoring and controlling include the following:* Profile of sub processes under statistical management (e.g., number planned to be under statistical management, number currently being statistically managed, and number that are statistically stable), Number of special causes of variation identified;

- *Examples of activities reviewed include the following:* Quantitatively managing the project using quality and process-performance objectives, Statistically managing selected sub processes within the project's defined process;

- *Examples of work products reviewed include the following:* Sub processes to be included in the project's defined process Operational definitions of the measures, Collected measures;

Based on these quantifications CMMI defines: "A `managed process` is a performed process that is planned and executed in accordance with policy; employs skilled people having adequate resources to produce controlled outputs; involves relevant stakeholders; is monitored, controlled, and reviewed; and is evaluated for adherence to its process description".

2.3.3 The SPICE Approach

The *Software Process Improvement and Capability dEtermination (SPICE)* is defined as an ISO/IEC standard TR 15504 [Emam 1998]. The SPICE process model considers the following process activities

- *Customer – supplier:* acquire software product, establish contract, identify customer needs, perform joint audits and reviews, package, deliver and install software, support operation of software, provide customer service, assess customer satisfaction
- *Engineering:* develop system requirements, develop software requirements, develop software design, implement software design, integrate and test software, integrate and test system, maintain system and software
- *Project:* plan project life cycle, establish project plan, build project teams, manage requirements, manage quality, manage risks, manage resources and schedule, manage subcontractors
- *Support:* develop documentation, perform configuration management, perform quality assurance, perform problem resolution, perform per reviews
- *Organization:* engineer the business, define the process, improve the process, perform training, enable reuse, provide software engineering environment, provide work facilities

Based in these process activities, SPICE defines the different *capability levels* such as incomplete, performed, managed, established, predictable, and optimizing. The principles of the process assessment of SPICE are given in the following semantic network [SPICE 2006].

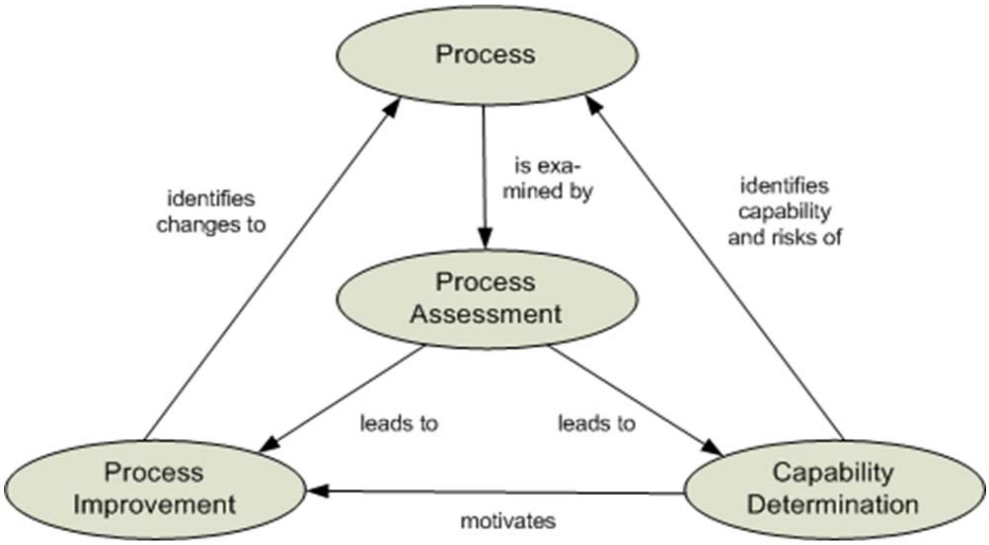


Figure 16: The SPICE process assessment model

The SPICE using and evaluation process is based on different documents: *concepts and introductory guide, guide for use in process improvement, guide for use in determining supplier process capability, qualification and training of assessors, rating processes, guide to conducting assessment, construction, selection and use of assessment instruments and tools, a model for process management.*

2.3.4 The Six Sigma Approach

Sigma (σ) stands for standard deviation of anything. The Six Sigma approach in the software development field was considered an interval (six: three at both sides) which keeps a 99.9 percent correctness as absence of any ***defects*** [Tayntor 2003]. The following table shows the defect percentage depending upon the different sigma levels.

Sigma level	Percent correct	#defects per million opportunities
3	93.3193	66807
4	99.3790	6210
5	99.9767	233
6	99.99966	3.4

Table 2: Characteristics of different sigma levels

The cornel process of the Six Sigma approach includes/uses five phases referred to as the *DMAIC* model which means

1. *Define* the problem and identify what is important (define the problem, form a team, establish a project charter develop a project plan, identify the customers, identify key outputs, identify and prioritize customer requirements, document the current process).

2. *Measure* the current process (determine what to measure, conduct the measurements, calculate the current sigma level, determine the process capability, benchmark the process leaders).
3. *Analyze* what is wrong and potential solutions (determine what cause the variation, brainstorm ideas for process improvements, determine which improvements would have the greatest impact on meeting customer requirements, develop a proposed process map, and assess the risk associated with the revised process).
4. *Improve* the process by implementing solutions (gain approval for the proposed changes, finalize the implementation plan, implement the approved changes).
5. *Control* the improved process by ensuring that the changes are sustained (establish key metrics, develop the control strategy, celebrate and communicate success, implement the control plan, measure and communicate improvements).

The general aspects of the Six Sigma approach are shown in the following figure [Dumke 2005].

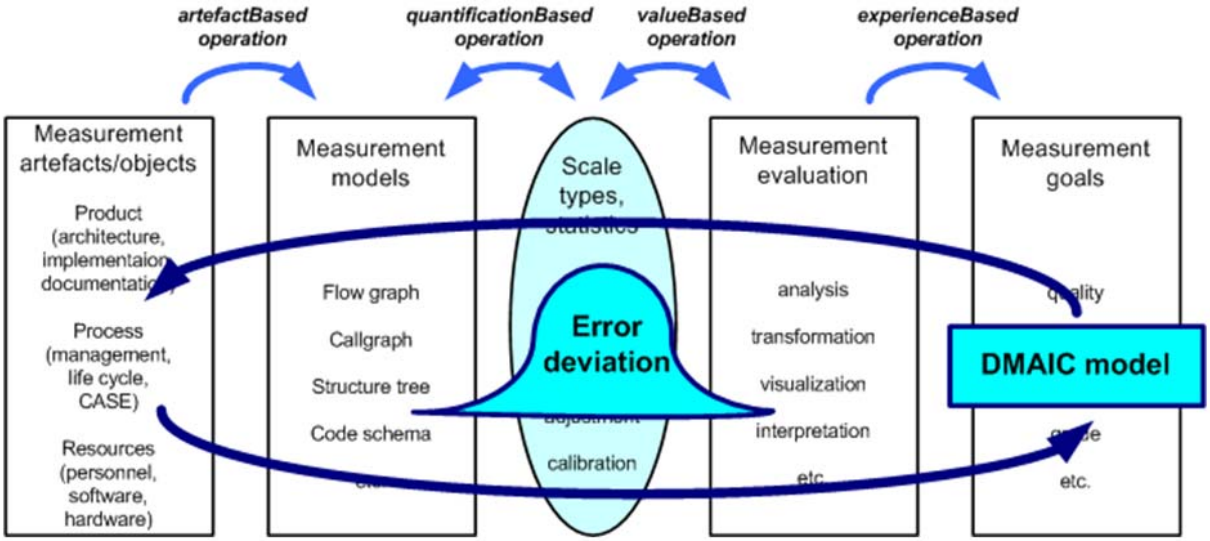


Figure 17: Basic characteristics of the Six Sigma approach

Furthermore, the Six Sigma approach is available for [Tayntor 2003] traditional software development life cycle, legacy systems, package software implementation, and outsourcing.

2.3.5 The ITIL Approach

ITIL (the *IT Infrastructure Library*) is a set of documents that are used to aid the implementation of a framework for *IT Service Management* ([ITIL 2006], [Johnson 2007]). This framework characterises how *Service Management* is applied within an organisation. ITIL was originally created by the CCTA, a UK Government agency, it is now being adopted and used across the world as the de facto standard for best practice in the provision of IT Service.

ITIL is organized into a series of sets as a ***best practice approach***, which themselves are divided into eight main areas

1. *Service Support* is the practice of those disciplines that enable IT Services to be provided effectively (service-desk, incident management, problem management, change management, configuration management, release management).
2. *Service Delivery* covers the management of the IT services themselves (service level management, financial management for IT services, capacity management, service continuity management, availability management).
3. *Security Management* considers the installation and realization of a security level for the IT environment (trust, integrity, availability, customer requirements, risk analysis, authority, and authenticity).
4. *ICT Infrastructure Management* describes four management areas: design and planning, deployment, operations, technical support.
5. *Application Management* describes the service life cycle as requirements – design – build- deploy – operate – optimise.
6. *Planning to Implement Service Management* defines a guide in order to deploy the ITIL approach in a concrete IT environment.
7. *The Business Perspective* describes the relationships of the IT to the customers and users.
8. *Software Asset Management* defines the processes and the life cycles for managing the software assets.

The following *triangle* characterizes the different relationships between the service management standards and ITIL.

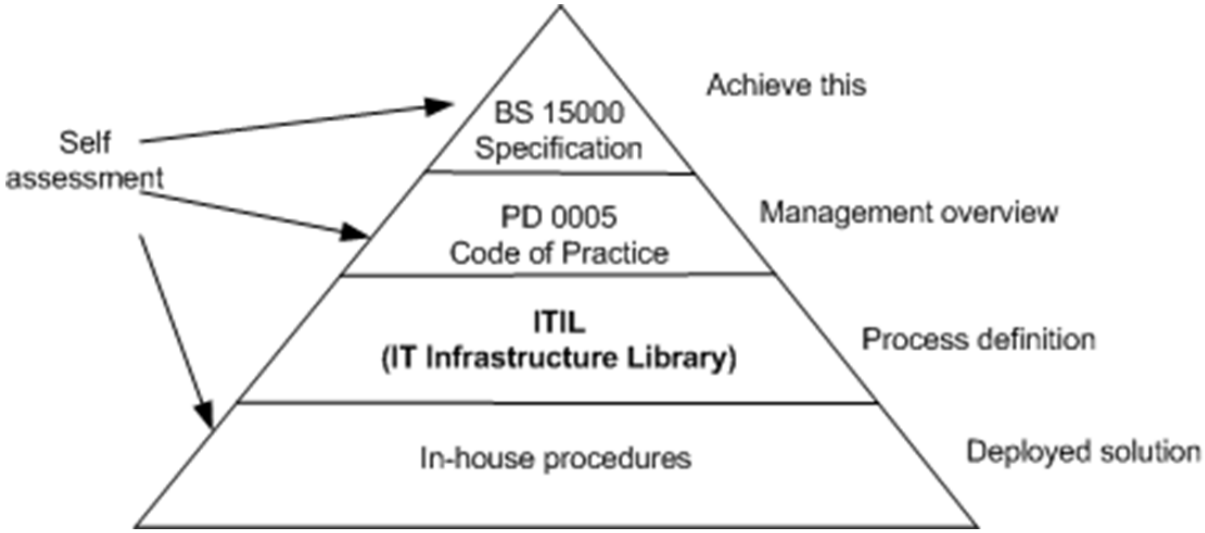


Figure 18: The relationship between the service standards and ITIL

where BS 15000 is the service management standard, ISO 20000 describes the specification for service management, and PD 0005 stands for code of practice for the IT service management (ITSM). Usually, the implementation of the ITIL approach is supported by any *ITIL toolkits*.

2.3.6 Further Software Process Evaluation

Further process measurement approaches are addressed to special process aspects or IT characteristics such as

- Assessment software ***processes for small companies*** [Wangenheim 2006] considering CMMI, ISO 9001 and SPICE and definition of a *Métododе Avaliacao de Processo de Software (MARES)* that includes that assessment phases *planning, contextualization, execution, monitoring and control, and post-mortem* which will be applied continuously.

- The ***agile process management*** could be described as follows (see [Augustine 2005] and [Boehm 2005])

    - The agile methods are lightweight processes that employ short iterative cycles, actively involve users to establish, prioritize, and verify requirements and rely on a team's tacit knowledge as opposed to documentation

    - The ability to manage and adapt to change

    - A view of organization's fluid, adaptive systems composed of intelligent people

    - Recognition of the limits of external control in establishing order

    - An overall humanistic problem-solving approach (considers all members to be skilled and valuable stakeholders in team management, relies on the collective ability of autonomous teams as the basic problem-solving mechanism, minimizes up-front planning, stressing instead adaptability to changing conditions)

- ***Management issues of internet/Web systems*** [Walter 2006] which defines the priority of management aspects as

    1. *protecting information about consumers,*
    2. *holistic thinking of company activities,*
    3. *linking internet strategic planning with corporate strategic planning,*
    4. *aligning internet development projects with corporate strategies,*
    5. *prioritizing company's internet objectives,*
    6. *providing adequate reassurance to consumers that information is fully protected,*
    7. *recruiting trained internet personnel,*
    8. *intranets remain security problems,*
    9. *retaining trained internet personnel,*
    10. *making company logistics system compatible with the internet,*
    11. *providing data privacy and data security to costumer companies,*

12. *providing adequate firewall,s*
13. *the site objectives requires definition,*
14. *recognizing potential benefits available from the internet,*
15. *intellectual property rights have become a major concern,*
16. *internet personnel should be strategists,*
17. *making WWW sites user friendly,*
18. *costs/benefits analyses fir internet systems are difficult,*
19. *keeping up to the dynamism of the internet-based marketplace,*
20. *providing quality customer service through interne systems,*
21. *the speed of change makes internet technology forecasting difficult,*
22. *integrating internet systems across multiple sites within a company,*
23. *linking internet systems to other internet systems*
24. *distribution channel conflicts inhibit more widespread use of e-commerce,*
25. *developing new cots/benefits analysis methodologies to evaluate internet project,*
26. *competitors may be leaping ahead.*

- ***Product line project management*** [Clements 2005] is based on the product line development phases as *core asset development, product development,* and *management.* This management involves the typical project input as products requirements, product line scope, core assets, and a production plan. The following figure shows the "What to Build" pattern used in this management area.

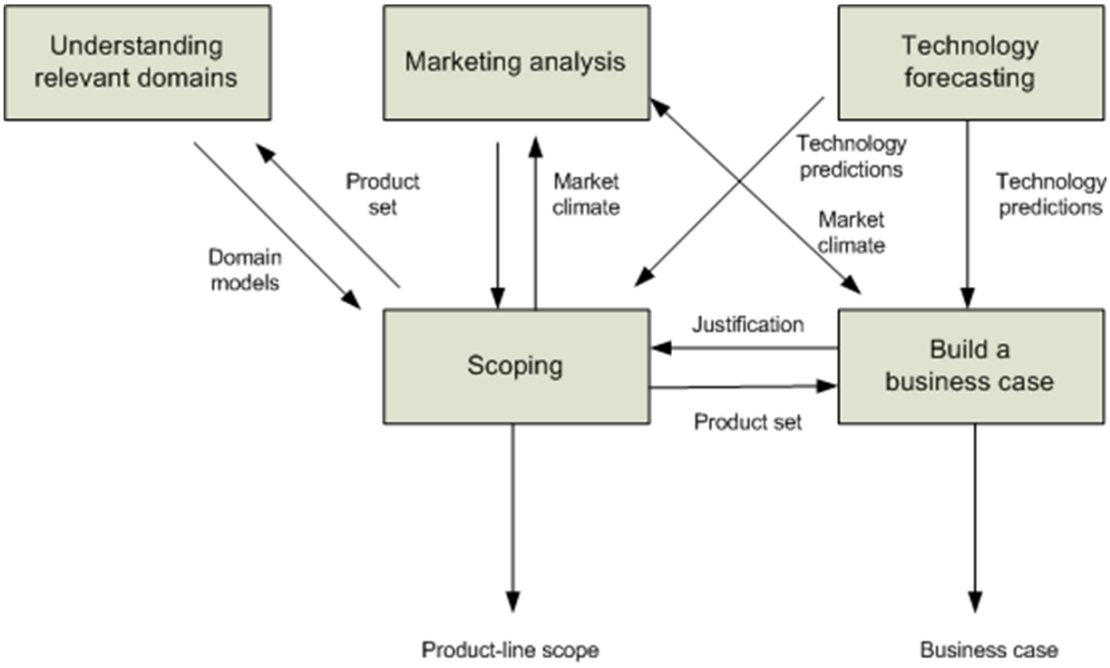


Figure 19: The "What to Build" pattern for product line project management

- ***Personal Software Process (PSP)*** considers the quality of the IT personnel themselves by analysis, evaluation and improvement of their activities [Humphrey 2000]. The following figure shows the essential steps of the PSP.

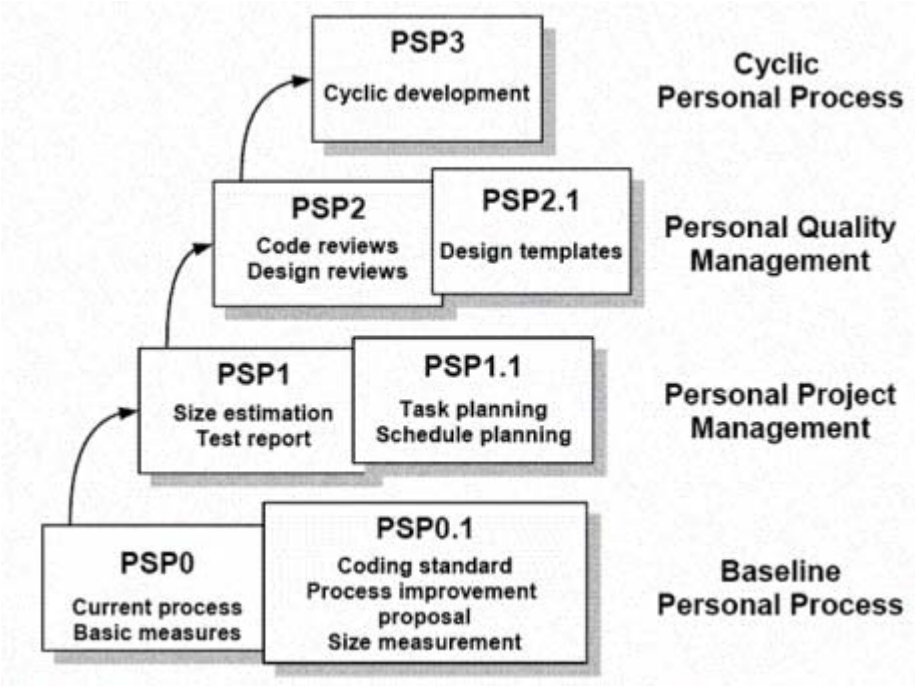


Figure 20: The PSP approach

Based on the *Telemetry project* from Johnson et al. [Johnson 2005] Ullwer has defined and implemented a background measurement and repository in order to automate the PSP using the so called *Hackystat* technology for Open Office [Ullwer 2006].

Currently, an experience in the industrial area is available and shows the relevance of this approach ([Kamatar 2000], [Zhong 2000]).

2.4 Summary

This chapter describes the software engineering field in general and motivates the quality assurance aspects considering the software process respectively. For our thesis approach we can establish that

- Software processes can be defined in different levels using experiences, considering development paradigms and determine process activities, categories and (sub) systems
- The main area of software processes is the development on software systems/products using software resources such as personnel, COTS, hardware and (basically) CASE tools
- Different approaches of software process descriptions such as CMMI or ITIL helps to improve the process but define the quality assurance implicitly and in a verbal manner mostly

In order to evaluate the software processes involving CASE tools we describe the CASE based development in the next chapter.

3 CASE-Based Software Development

3.1 CASE Tools

In the chapter before (formula (2.5)) we have CASE (Computer-Aided Software Engineering) characterizes as CASE tools mainly summarizing as Integrated CASE or ICASE as

$$ICASE = CASE \cup CARE \cup CAME \tag{3.1}$$

where *CARE* stands for computer-aided reengineering and *CAME* means computer-assisted measurement and evaluation tools. A simple example of a CASE tools architecture is given in the following figure.

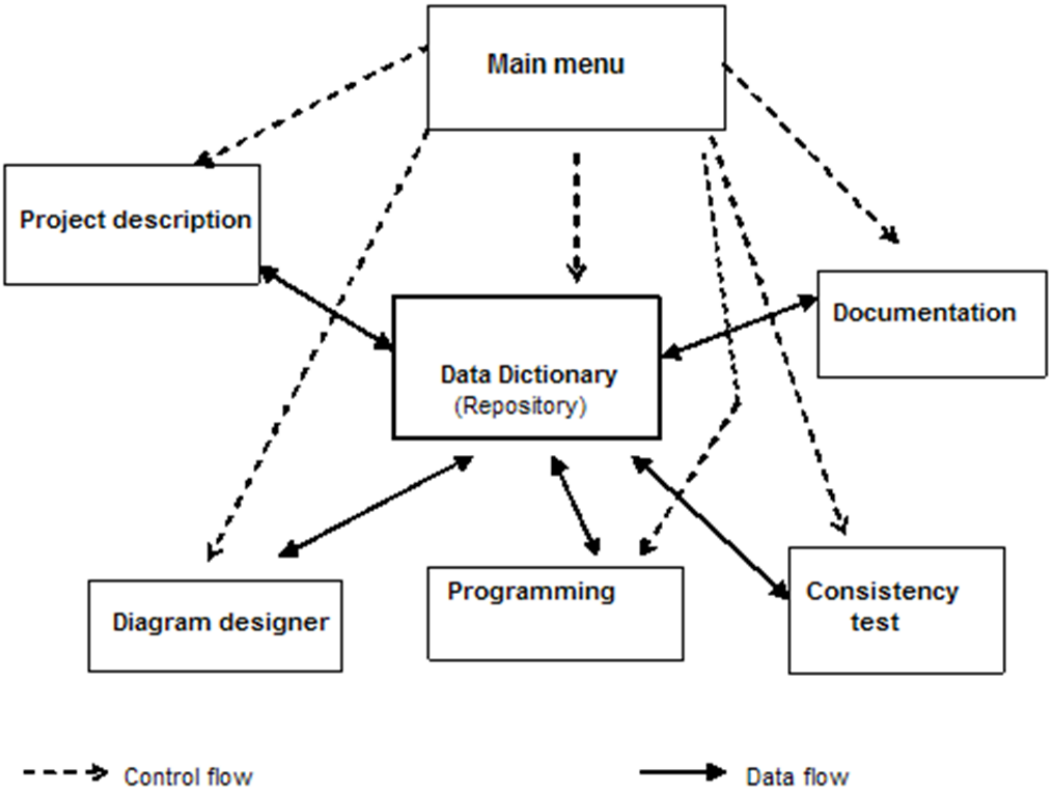


Figure 21: Simplified CASE tool architecture

CASE could be used in different software development phases and software maintenance activities. Adapting formula (2.3) for different development paradigm we can identify CASE tools as

$$\{CASE_{developmentMethods}, CASE_{lifecycle}, CASE_{softwareManagement}\} \tag{3.2}$$

and

$$\{CASE_{problemDefinition}, CASE_{requirementAnalysis}, CASE_{specification}, CASE_{design}, CASE_{implementation}, CASE_{acceptanceTest}, CASE_{delivering}\} \tag{3.3}$$

and

$$\{CASE_{SAM}, CASE_{OOSE}, CASE_{CBSE}, CASE_{AOSE}\} \quad (3.4)$$

Considering formula (2.4) we can classify different CASE tools for the development of different software systems such as

$$\{CASE_{informationSystem}, CASE_{constructionSystem}, CASE_{embeddedSystem}, CASE_{communicationSystem}, CASE_{distributedSystem}, CASE_{knowledgeBasedSystem}\} \quad (3.5)$$

Relating to formula (2.6) we can identify CASE tools in the software maintenance area as

$$\{CASE_{extension}, CASE_{adaptation}, CASE_{correction}, CASE_{improvement}, CASE_{prevention}\} \quad (3.6)$$

A simple example of a CASE Tool architecture is given in the following figure characterizing the Eclipse tool.

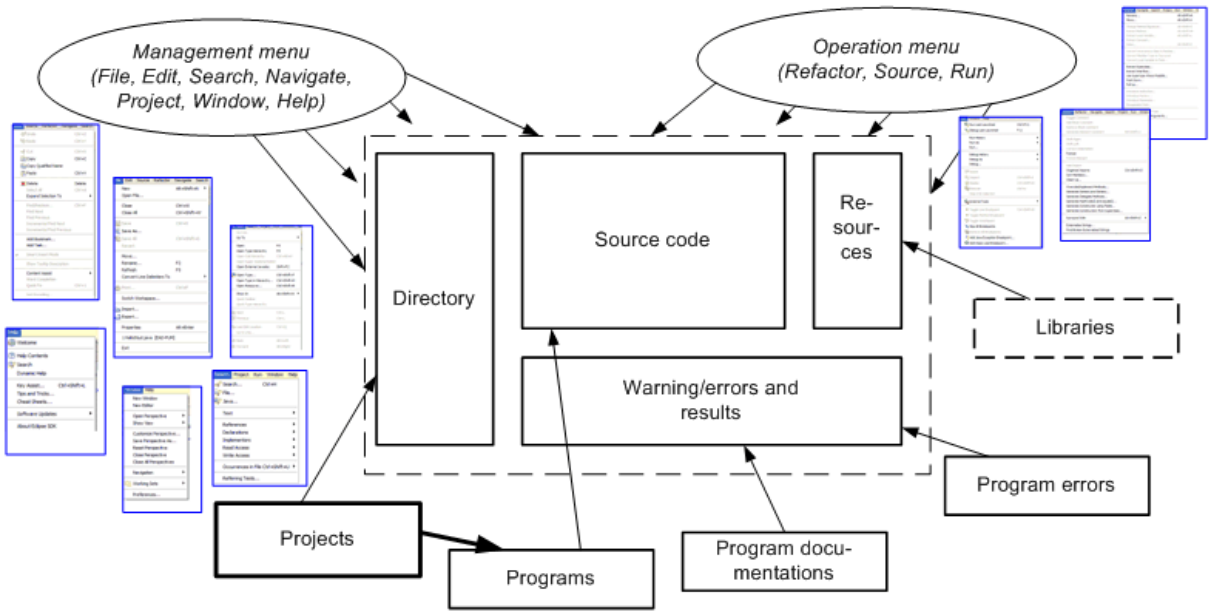


Figure 22: Eclipse tool GUI architecture

CASE tools could be built as software development infrastructures based on CASE environments as following

$$CASE_{infrastructure} = \{ (\{UpperCASE\} \cup \{LowerCASE\})_{environment} \} \quad (3.7)$$

Therefore, we had defined

$$UpperCASE = \{modellingTool, searchTool, documentationTool, diagramTool, simulationTool, benchmarkingTool, communicationTool\}$$

$$LowerCASE = \{assetLibrary, programmingEnvironment, programGenerator, compiler, debugger, analysisTool, configurationTool\}$$

Kinds of CASE tool integration as *presentation, controlling, platform* and *data integration* are shown in the following figure by [Wasserman 1990].

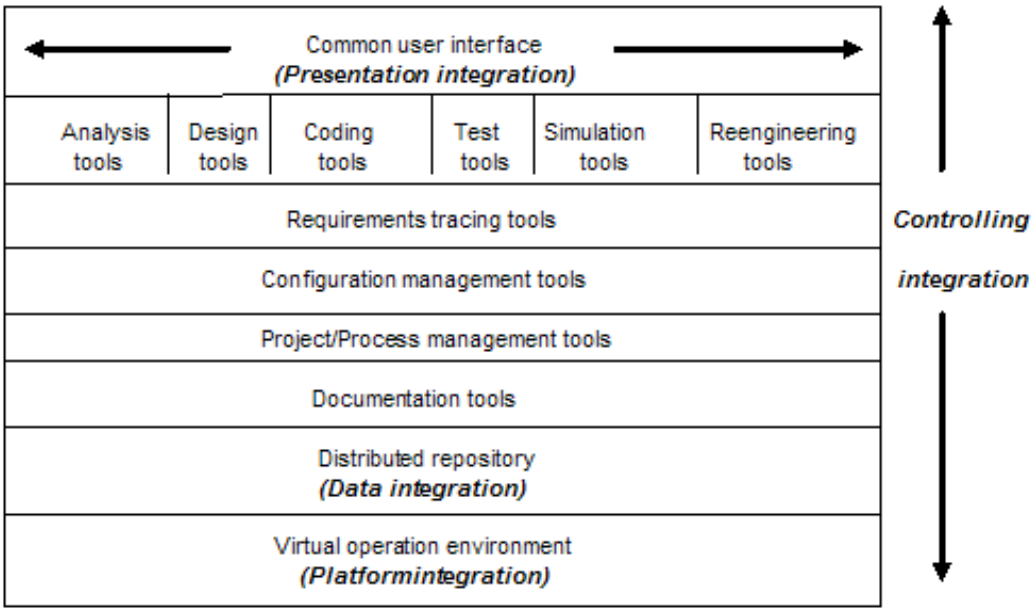


Figure 23: Four essential types of CASE tool integration

Finally, CASE tools could be identified for different paradigms, system and development support characterized in the following figure.

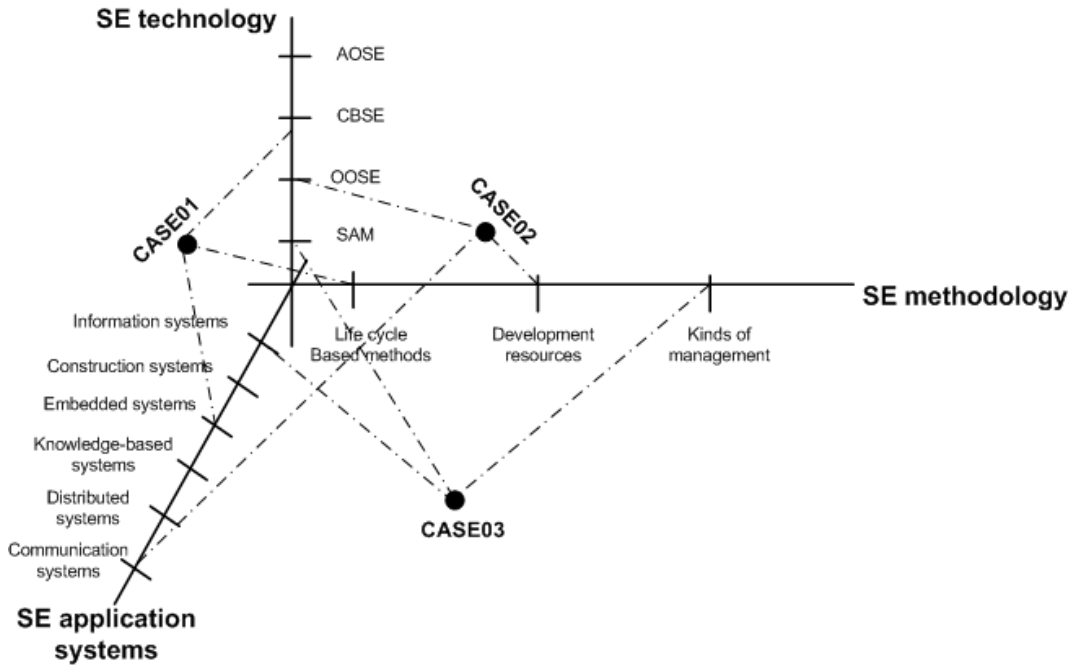


Figure 24: CASE tools in the dimensions of the software engineering

This figure demonstrates the large “space” of CASE tools variants and categories in the software development processes.

3.2 CASE-Based Processes

Using our process descriptions in the chapter before we can adapt these definitions in a CASE tool based manner as following (based on [Wang 2000]).

> ***CASE tool based development processes*** *are processes that belong to a technical process subsystem, which regulate the development activities in software design, implementation, and maintenance using tools for supporting, performing and managing these process activities.*
>
> *A* ***CASE tool based empirical process model*** *is a model that defines an organized and benchmarked software process system and best practices captured and elicited from the software industry involved in the used and adapted CASE tools.*

Furthermore, in order to analyze and improve the development processes we adapt the following definitions of kinds of process evaluations.

> ***CASE-based software process establishment*** *is a systematic procedure to select and implement a process system by model tailoring, extension, and/or adaptation techniques and appropriate tools.*
>
> ***CASE-based software process assessment (CSPA)*** *is a systematic procedure to investigate the existence, adequacy, and performance of an implemented and tool-based process system against a model, standard, or benchmark.*
>
> ***CASE-based process capability determination*** *is a systematic procedure to derive a capability level for a process, and/or organization based on the evidence of existence, adequacy, and performance of the required practices defined in a CASE tool based software engineering process system.*
>
> ***CASE-based software process improvement (CSPI)*** *is a systematic procedure to improve the performance of an existing CASE tool based process system by changing the current processes or updating new processes in order to correct or avoid problems identified in the old process system by means of a process assessment.*

Otherwise existing standards and approaches for software process evaluation considers the CASE tool application in an implicit but meaningful manner from an automation point of view.

The following semantic network showing in the chapter before can be used in order to characterize these intentional aspects.

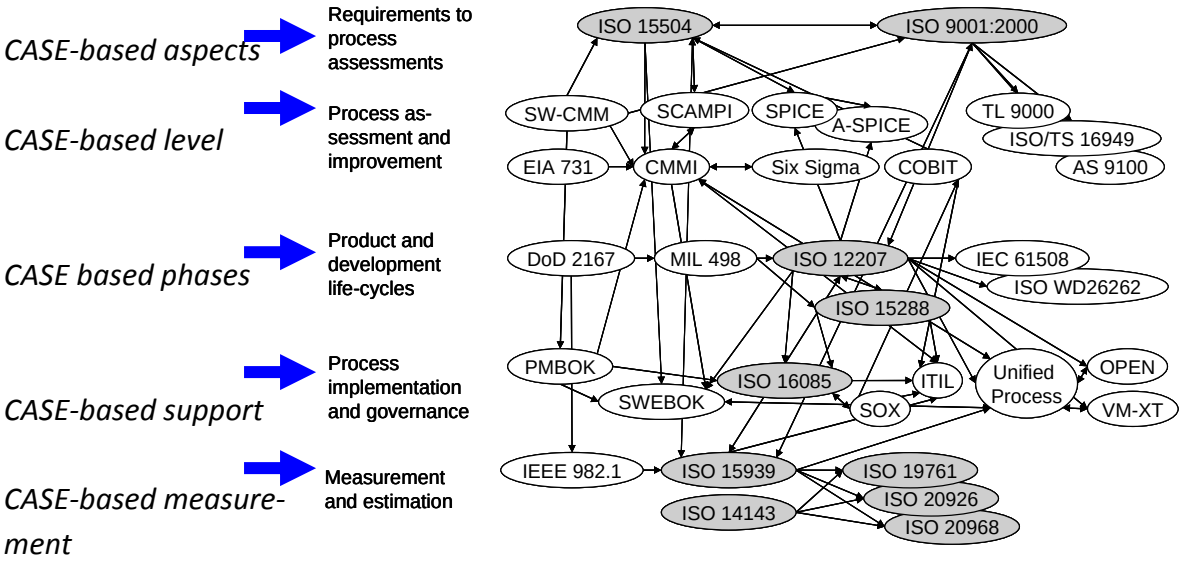


Figure 25: CASE and software process evaluation methods and standards

The following adapted figure summarizes the mainly CASE-based area of the maturity models and chosen improvement models of software process evaluation.

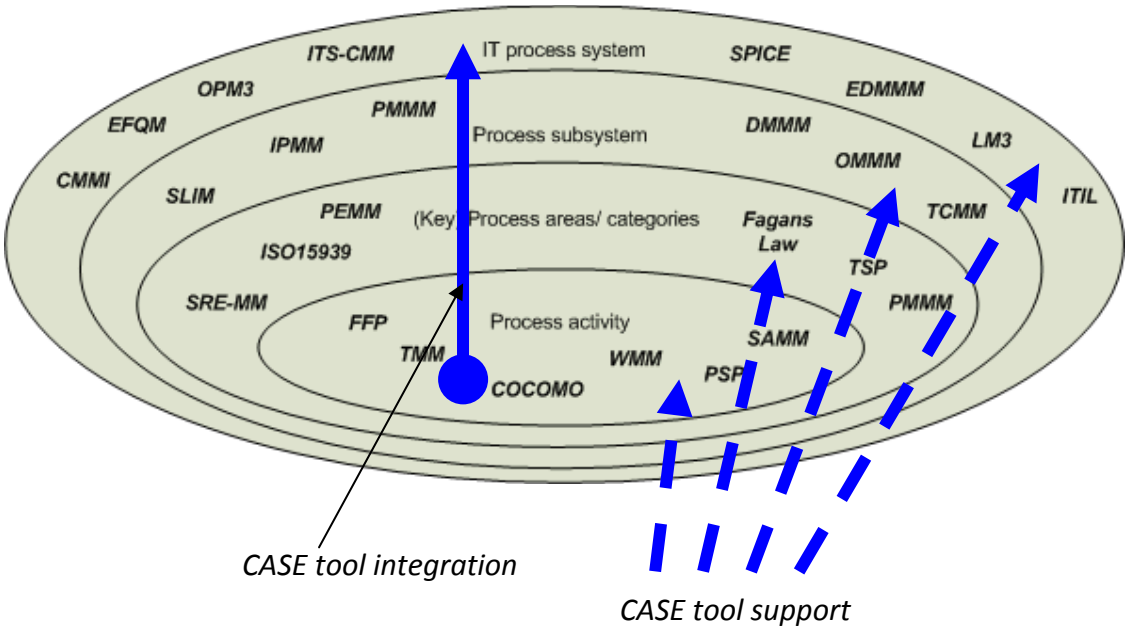


Figure 26: Overview of chosen process maturity and improvement models

Chosen CMMI-based software measurement intentions in order to manage the software process quantitatively considering CASE-based development are [SEI 2002]:

- ***CASE tool based support*** *for the examples of quality and process performance attributes for which needs and priorities might be identified include the following:* Functionality, Reliability, Maintainability, Usability, Duration, Predictability, Timeliness, and Accuracy;

- ***CASE tool based determination*** *of the examples of quality attributes for which objectives might be written include the following:* Mean time between failures, Critical resource utilization, Number and severity of defects in the released product, Number and severity of customer complaints concerning the provided service;

- ***CASE tool based support*** *for the examples of process performance attributes for which objectives might be written include the following:* Percentage of defects removed by product verification activities (perhaps by type of verification, such as peer reviews and testing), Defect escape rates, Number and density of defects (by severity) found during the first year following product delivery (or start of service), Cycle time, Percentage of rework time;

- ***CASE tool based support*** *for the examples of sources for objectives include the following:* Requirements, Organization's quality and process-performance objectives, Customer's quality and process-performance objectives Business objectives, Discussions with customers and potential customers, Market surveys;

- ***CASE tool based support*** *for the examples of sources for criteria used in selecting sub processes include the following:* Customer requirements related to quality and process performance, Quality and process-performance objectives established by the customer, Quality and process-performance objectives established by the organization, Organization's performance baselines and models, Stable performance of the sub process on other projects, Laws and regulations;

- ***CASE tool based determination*** *of the examples of product and process attributes include the following:* Defect density, Cycle time, Test coverage;

- ***CASE tool based support*** *for the example sources of the risks include the following:* Inadequate stability and capability data in the organization's measurement repository, Sub processes having inadequate performance or capability, Suppliers not achieving their quality and process-performance objectives, Lack of visibility into supplier capability, Inaccuracies in the organization's process performance models for predicting future performance, Deficiencies in predicted process performance (estimated progress), Other identified risks associated with identified deficiencies;

- ***CASE tool based determination*** *of the examples of sub process measures include the following:* Requirements volatility, Ratios of estimated to measured values of the planning parameters (e.g., size, cost, and schedule), Coverage and efficiency of peer reviews, Test coverage and efficiency, Effectiveness of training (e.g., percent of planned training completed and test scores), Reliability, Percentage of the total defects inserted or found in the different phases of the project life cycle Percentage of the total effort expended in the different phases of the project life cycle;

- ***CASE tool based support*** *for the examples of criteria for determining whether data are comparable include the following:* Product lines, Application domain, Work product and task attributes (e.g., size of product), Size of project;

- ***CASE tool based determination*** *of the examples of where the natural bounds are calculated include the following:* Control charts, Confidence intervals (for parameters of distributions), Prediction intervals (for future outcomes);

- ***CASE tool based support*** *for the examples of techniques for analyzing the reasons for special causes of variation include the following:* Cause-and-effect (fishbone) diagrams, Designed experiments, Control charts (applied to sub process inputs or to lower level sub processes), Sub grouping (analyzing the same data segregated into smaller groups based on an understanding of how the sub process was implemented facilitates isolation of special causes);

- ***CASE tool based support*** *for the examples of actions that can be taken when a selected sub process' performance does not satisfy its objectives include the following:* Changing quality and process-performance objectives so that they are within the sub process' process capability, Improving the implementation of the existing sub process so as to reduce its normal variability (reducing variability may bring the natural bounds within the objectives without having to move the mean), Adopting new process elements and sub processes and technologies that have the potential for satisfying the objectives and managing the associated risks, Identifying risks and risk mitigation strategies for each sub process' process capability deficiency;

- ***CASE tool based determination*** *of the examples of other resources provided include the following tools:* System dynamics models, Automated test-coverage analyzers, Statistical process and quality control packages, Statistical analysis packages

- ***CASE tool based determination*** *of the examples of training topics include the following:* Process modelling and analysis, Process measurement data selection, definition, and collection;

- ***CASE tool based determination*** *of the examples of work products placed under configuration management include the following:* Sub processes to be included in the project's defined process, Operational definitions of the measures, their collection points in the sub processes, and how the integrity of the measures will be determined, Collected measures;

- ***CASE tool based support*** *for the examples of activities for stakeholder involvement include the following:* Establishing project objectives, Resolving issues among the project's quality and process-performance objectives, Appraising performance of the selected sub processes, Identifying and managing the risks in achieving the project's quality and process-performance objectives, Identifying what corrective action should be taken;

- ***CASE tool based determination*** *of the examples of measures used in monitoring and controlling include the following:* Profile of sub processes under statistical management (e.g., number planned to be under statistical management, number currently being statistically managed, and number that are statistically stable), Number of special causes of variation identified;

- ***CASE tool based support*** *for the examples of activities reviewed include the following:* Quantitatively managing the project using quality and process-performance objectives, Statistically managing selected sub processes within the project's defined process;

Finally, the best practice approach of ITIL considers the essential CASE-based aspects as "*Service Support* is the practice of those disciplines that enable IT Services to be provided effectively (service-desk, incident management, problem management, change management, configuration management, release management)" [Johnson 2007].

3.3 Summary

The third chapter has described the CASE-based foundations of software process supports and implementations. For our thesis approach we can establish that

- CASE tools can be found in all software engineering dimensions such as technologies (OOSE, CBSE etc.) kinds of systems (embedded, information, knowledge-based etc.) and development process aspects (lifecycle, management etc.)
- An effective use of CASE tool requires their consistent integration over the different process activities and phases
- CASE tools can be used as monolithic tools, tool sequences, tool compositions and as (Web) services respectively
- CASE tool could be integrated in the software development process as CASE-based software process assessment and CASE-based software process improvement
- Evaluation and improvement standards and approaches define the functional background of CASE tools and their meaningful and effective application as CASE tool based support and determination

The different kinds of software process analysis, evaluation and measurement involving CASE tools are described in the next chapter.

4 Software Process Measurement and Evaluation

Process metrics or measures are involved in software measurement processes and are based on process experiences. Therefore, we will define these activities and information basics at first. The ***measurement methods M*** could be classified as following [Dumke 2005]

$$M = \{artefactBasedOperation, quantificationBasedOperation, valueBasedOperation, experienceBasedOperation\} \quad (4.1)$$

where

$$artefactBasedOperation \subseteq \{modelling, measurement, experimentation, assessment\}$$

$$quantificationBasedOperation \subseteq \{transformation, regression, factorAnalysis, calibration\}$$

$$valueBasedOperation \subseteq \{unitTransformation, correlation, visualization, analysis, adjustment, prediction\}$$

$$experienceBasedOperation \subseteq \{trendAnalysis, expertise, estimation, simulation, interpretation, evaluation, application\}$$

The ***measurement experiences*** summarize the general aspects of the concrete measurement results in different forms of aggregation, correlation, interpretation and conclusion based on a context-dependent interpretation.

Note that the measurement experience is divided in the experiences of the measurement results and the (evaluated-based) experience of the measurement itself. In following we only consider the first aspect. Some kinds of measurement experience are ([Armour 2004], [Davis 1995], [Endres 2003], [Kenett 1999])

$$\boldsymbol{E} \subseteq \{analogies, axioms, correlations, criterions, intuitions, laws, lemmas, formulas, methodologies, principles, relations, ruleOfThumbs, theories\} \quad (4.2)$$

Some examples of these kinds of experience are (see also [Basili 2001], [Boehm 1989], [Dumke 2003], [Halstead 1977] and [Putnam 2003])

$$analogies \in \{analogicalEstimation, systemAnalogy, hardwareSoftwareAnalogy\}$$

$$criteria \in \{fulfilCondition, qualityAspect, minimality, maximality\}$$

$$laws \in \{BrooksLaw, DijkstraMillsWirthLaw, FagansLaw, GlassLaw, GraySerlinLaw, McIlroysLaw, MooresLaw, SimonsLaw\}$$

$$lemmas \in \{\text{'any system can be tuned'}, \text{'installability must be designed in'}, \text{'human-based methods can only be studied empirically'}\}$$

$$methodologies \in \{agileMethodology, cleanroomMethodology, empiricalBasedMethodology\}$$

principles ∈ {'don't set unrealistic deadlines', 'evalvate alternatives',
'manage by variance', 'regression test after every change'}

rulesOfThumb ∈ {'one dollar in development leads to two dollars maintenance',
'1 KLOC professional developed programs implies 3 errors',
'more than 99 percent of all executing computer instructions come from COTS', 'more than the half of the COTS features go unused'}

On the other hand, there are three different types of empirical strategies: *survey, case study* and *experiment* (see [Juristo 2003], [Kitchenham 1997]). In following we will cited some examples from the literature for these kinds of measurement and experience addressed to the software process.

4.1 Software Process Indicators and Criteria

Special indicators or criteria for *project management* are defined by [Lecky-Thompson 2005] in the following manner:

Specification project management: invoice generation, reporting, payment tracking, order processing, account maintenance, customer management, stock management, and tax return;

Promoting corporate quality: projecting quality (communicating quality, documentation, rewarding quality), managing quality (quality reviews, quality checklists, total quality management, quality circles), document quality (process description documents, benchmark reporting, badges);

Feedback techniques: reporting line (documenting the reporting line, the reporting line document, specification, design, implementation, integration), central communication (quality management, change management, quality measurement), supporting the reporting process (external documentation, motivation via improvement),

Client satisfaction: pre- and post-project surveys, planning for failure, poor quality requirements capture, poor quality implementation, managing client dissatisfaction, poor quality specifications.

Another taxonomy of project management considers the special aspects of managing virtual teams [Haywood 1998]. These are

Virtual team characteristics: geographical separation of team members, skewed working hours, temporary or matrix reporting structures, multi-corporation or multi-organizational teams

Virtual team members: individual located at other corporate site, joint venture partners, telecommuters, consultants, third-party developers, vendors, suppliers, offshore development and manufacturing groups, satellite work groups, customers, clients;

Factors driving the prevalence of distributed teams: mergers, acquisitions, downsizing, outsourcing, technology, clean air laws, offshore development and manufacturing, technical specialization;

Manager's perspective of the advantages of a distributed team: access to a less expensive labor pool, reduced office space, greater utilization of employees, round-the-clock work force, greater access to technical experts, larger pool of possible job candidates;

Team member's perspective of the advantages of a distributed team: increased independence, less micro management, larger pool of jobs to choose from, greater flexibility, opportunity for travel;

Expectations to research: increased productivity, improved disaster recovery capabilities, increased employee satisfaction and retention, reduced office space requirements, environmental benefits, closer proximity to customers, increased flexibility, greater access to technical experts, larger pool of potential job candidates;

Manager's perspective of the challenges of distributed teams: team building, cultural issues, cost and complexity of technology, process and workflow;

Team member's perspective of the challenges of distributed teams: communication, technical support, recognition, inclusion vs. isolation, management resistance.

As ***key success factors for software process improvement*** (SPI) identify Lepasaar et al. [Lepasaar 2001] the following:

1. SPI related training;
2. External guidance of the SPI work;
3. Company's commitment to SPI activities;
4. External support for SPI activities;
5. Managements support for SPI;
6. SPI environment support for a sufficiently long period of time (external mentoring);
7. Availability of company's own resources;
8. Measurable targets set to SPI work.

Kandt gives a summarizing about different software quality drivers shown in the following table [Kandt 2006].

	Boehm's Ranking	Clark's Ranking	Neufelder's Ranking
1	Personnel/Team	Product complexity	Domain knowledge
2	Product complexity	Analyst capability	Non-programming managers
3	Required reliability	Programmer capability	Use of unit testing tools
4	Timing constraint	Constraint on execution time	Use of supported operating systems
5	Application experience	Personnel continuity	Testing of user documentation
6	Storage constraint	Required reliability	Use of automated tools
7	Modern programming practice	Documentation	Testing during each phase
8	Software tools	Multi-site development	Reviewed requirements
9	Virtual machine volatility	Application experience	Use of automated fracas
10	Virtual machine experience	Platform volatility	Use of simulation

Table 3: Software Quality Drivers

An overview about the essential indicators in order to characterize *defects* is given in [Emam 2005] as following:

- ***Defects and usage***: usage is a function of the number of end users using the product, the number of actual machines executing the product, the time since release
- ***Raw defect counts***: *defect density = (number of defects found)/size*
- ***Adjusted defect density:*** e. g. adjusted by comparisons to "standard company"
- ***Defect classification:*** as defect priorities, defect severities, classification by problem type

A stage model applying the defect analysis is shown in the following figure [Emam 2005].

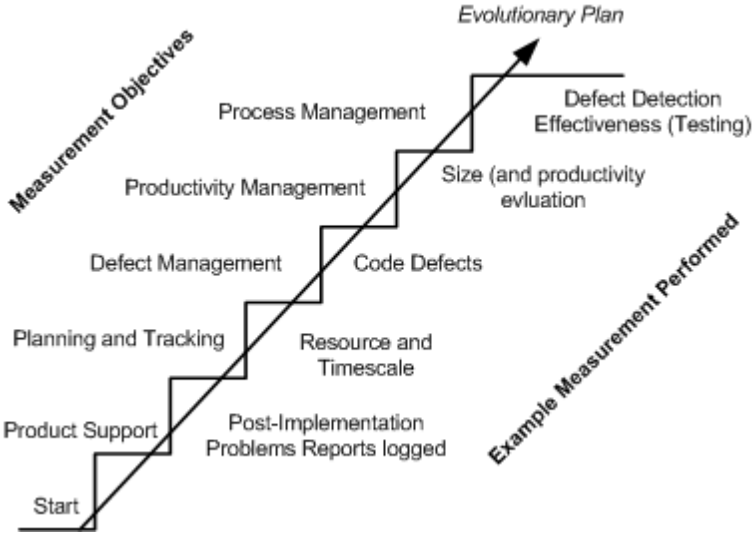


Figure 27: A model showing the stages of measurement that organizations typically go through

The IT controlling could be classified in the following sub processes defined by Gadatsch and Mayer [Gadatsch 2005]: *ADV controlling, DV controlling, EDV controlling, INF controlling, IV controlling, IS controlling, IT controlling.* Typical tools in order to support these processes are the *IT strategy, IT standards, IT portfolio management,* and *IT analysis and indicators.* A simple classification of IT indicators by [Gadatsch 2005] is

- *Absolute indicators*: as counting of anything,
- *Relative indicators*: as structural indicators, relational indicators, and index indicators.

Putnam and Meyers define the ***Five Core Metrics*** for software process analysis, improvement and controlling in the following manner [Putnam 2003]

1. quantity of function, usually measured in terms of ***size*** (such as source lines of code or function points), that ultimately execute on the computer
2. ***productivity,*** as expressed in terms of the functionality produced for the time and effort expended
3. ***time***, the duration of the project in calendar months
4. ***effort,*** the amount of work expended in person-months
5. ***reliability,*** as expressed in terms of defect rate (or its reciprocal, mean time effort)

The relationship of these core metrics are described by Putnam and Meyers as follows [Putnam 2003, p. 34]

> *"People, working at some level of **productivity,** produce a quantity of function or a **work product** at a level of **reliability** by the expenditure of **effort** over a **time** interval."*

Another relationship between the five core metrics defined by Putnam and Meyers characterizes a *first level* with the time and effort metric, a second level including the quality and productivity and a third (highest) level considering the function.

4.2 Software Process Laws

The following kinds of laws and hypothesis are cited from [Endres 2003]:

Fagan's law: *"Inspections significantly increase productivity, quality, and project stability"*. There are three kinds of inspection: design, code, and test inspection. They are applicable in the development of all information or knowledge intensive products. This form of inspection is wide spread throughout the industry today. Inspection also has a key role in the Capability Maturity Model (CMM). The benefit of inspections can be summarized as followed: they "create awareness for quality that is not achievable by any other method".

Porter-Votta law: *"Effectivness of inspections is fairly independent of its organizational form"*. A. Porter and L. Votta investigated the inspection process introduced by Fagan and came up with the following results: physical meetings are overestimated. It can be helpful while introducing the inspection process to new people. When education and experience are extant it is not that important anymore. Another point revealed was that it is not true that adding more persons to the inspection team increases the detection rate.

Hetzel-Myers law: *"A combination of different Verification and Validation methods outperforms any single method alone"*. W. Hetzel and G. Myers claim that it is better to use all three methods in combination to gain better results at the end. This is due to the fact that design, code and test inspection are not competitors.

Mills-Jones hypothesis: *"Quality entails productivity"*. It is also known as "the optimist's law" and can be seen as a variation of P. Cosby's proverb "quality is free". It is a very intuitive hypothesis: on the one hand, when the quality is high, less rework has to be done which results in better productivity. On the other hand, when quality is poor more rework has to be considered. Therefore productivity rate drops, as well.

Mays' hypothesis: *"Error prevention is better than error removal"*. No matter when an error is detected a certain amount of rework has to be done (this amount increases the later it is detected). Therefore it is better to prevent errors. To be able to do so, the circumstances of errors have to be investigated, identified and then removed. It is still a hypothesis because it is extremely difficult to prove.

Basili-Rombach hypothesis: *"Measurements require both goals and models"*. Metrics and measurement need goals and questions otherwise they do not have a meaning. It is also preferable to use a top-down approach when specifying the parameters. This leads to the Goal-Question-Metric (GQM) paradigm.

Conjecture a: *"Human-based methods can only be studied empirically"*. The human-based methods involve (human) judgement and depend on experience and motivation. This is why the results also depend on these different factors. To be able to understand and control those factors empirical studies are needed.

Conjecture b: *"Learning is best accelerated by a combination of controlled experiments and case studies"*. Observing software development helps the developers to learn. The case studies supply the project characteristics, (realistic) complexity, project pressure etc. The lack of cause and effect insights can be provided through controlled experiments.

Conjecture c: *"Empirical results are transferable only if abstracted and packaged with context"*. The information that has been gained needs to be transformed into knowledge with the context borne in mind. This can be achieved with the help of abstraction. It offers the opportunity to reuse the results. When the results are abstracted and packaged only two questions remain to be answered: "Do the results apply to this environment?" and "What are the risks of reusing these results?"

The following figure shows the variety of intentions of such laws. The detailed content of these laws is described in [Endres 03].

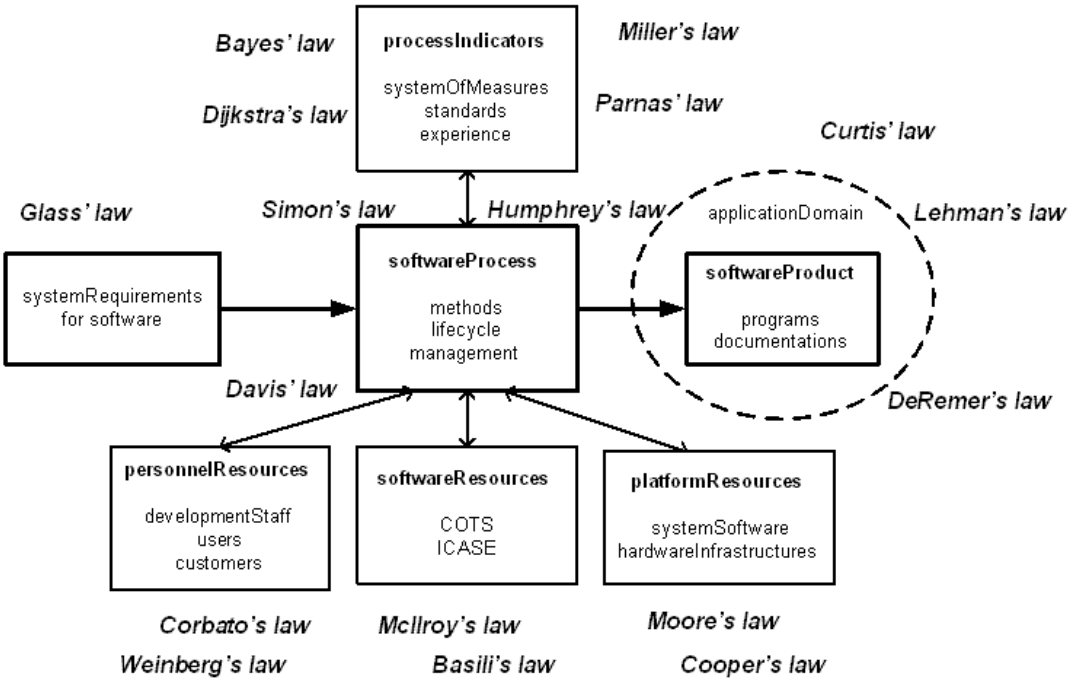


Figure 28: Intentions of chosen software engineering laws

4.3 Software Process Principles and Rules

Software value chains [Messerschmitt 2003]: „There are two value chains in software, in which participants in one chain add value sequentially to the others. The *supply chain* applies to the execution phase, tarts with the software vendor, and ends by providing valuable functionality and capability to the user. The *requirements value chain* applies to the software implementation phase, starts with the business and application ideas, gathers and adds functional and performance objectives from user, and finally end with a detailed set of requirements for implementation. Many innovations starts with software developers, who can better appreciate the technical possibilities but nevertheless require end-user involvement for their validation and refinement."

A ***cognitive structure of software evaluation*** is defined by [Wong 2001] shown in the following figure and consider the developer side as an essential software development resources.

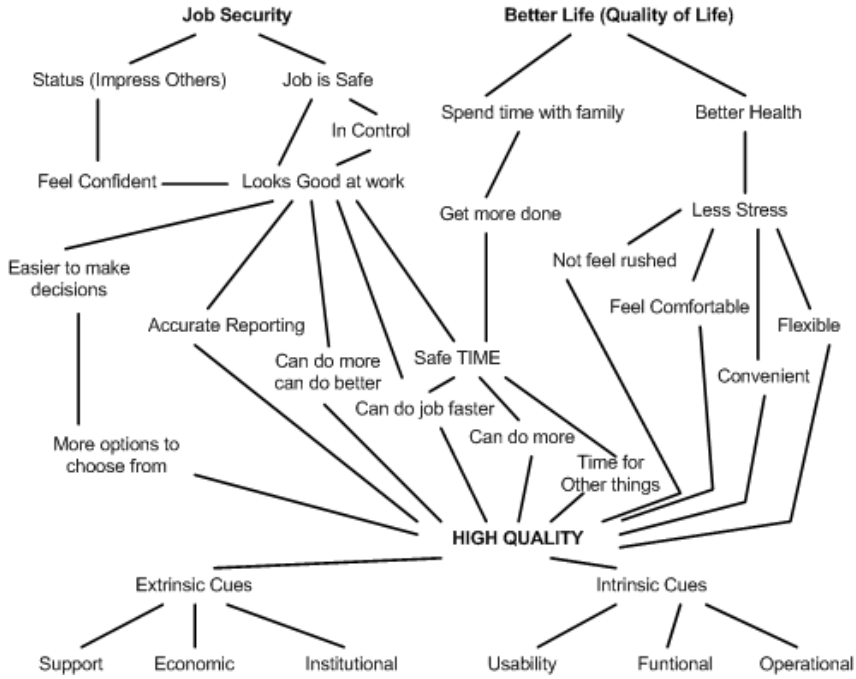


Figure 29: User's cognitive structure of software evaluation by Wong and Jeffery

The organizational and management-oriented activities of a ***lightweight process on extreme programming*** (LIPE) are defined by Zettel et al. [Zettel 2001] in the following manner:

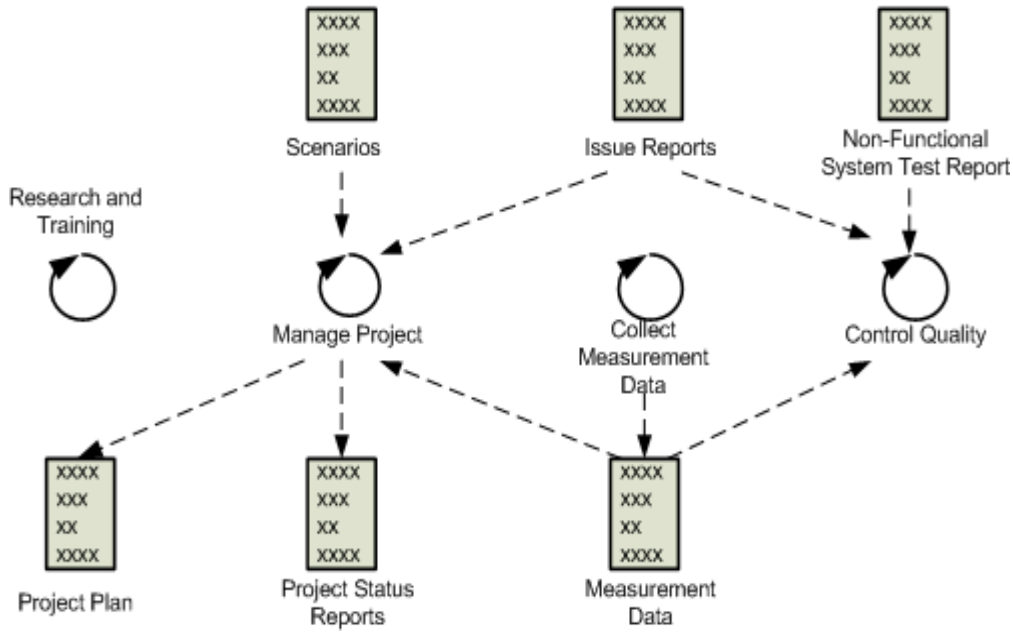


Figure 30: The LIPE activities and product flow among them by Zettel et al.

The typical issues of software evaluation in the IT area are shown in the following figure defined by [Ebert 2004]

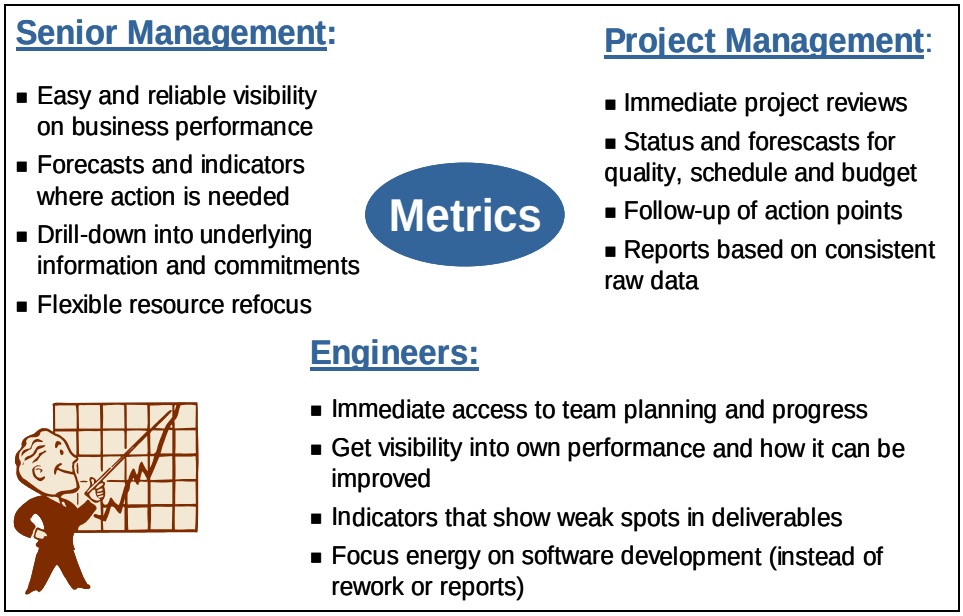


Figure 31: Metrics depends on stakeholder needs

A set of principles for the different areas of software quality are defined by Kandt in the following manner [Kandt 2006]:

- ***Practice for Management Personnel to Follow***
  - Inculcate an organizational vision
  - Commit to a software process improvement program
  - Create a software engineering steering group
  - Create a software engineering process group
  - Align the human resources organization
  - Assess the maturity of the organizational development processes
  - Identify changes and their scope
  - Demonstrate a quantitative financial benefit for each change
  - Obtain the commitment of practitioners
  - Measure personnel productivity and product quality

- ***Practice for Staffing an Organization***
  - Define minimal standards of knowledge for software personnel
  - Use a defined process to evaluate a candidate's knowledge
  - Hire summer interns and contractors a short-term basis
  - Hire personnel who have actually delivered software systems
  - Define graduated career opportunities to support growth in workface competencies
  - Define an orientation program for new personnel
  - Specify the number of days each year that people are expected to further develop their skills
  - Provide recent hires with on-the-job training
  - Train developers in the application domain
  - Relate skill development activities to the needs of individuals and projects
  - Reward outstanding accomplishments

- Define individuals performance goals and objects with each employee
- Provide small meeting rooms that can hold ten people
- Provide private, noise-free office space for software professionals
- Control problem employees
- Remove poor performers
- Challenge personnel
- Motivate employees
- Foster team cohesion
- Do not allow software engineers to work overtime beyond 40 consecutive days

- ***Practice for Planning a Project***
  - Conduct feasibility studies
  - Develop a project schedule following a defined procedure
  - Perform a risk assessment of a project following a defined procedure
  - Estimate the effort and cost of a software project following a defined procedure
  - Use metrics to manage a software project
  - Track milestones for large projects
  - Establish a project office for large projects
  - Use a hierarchical organizational structure for software projects
  - Collocate teams and the resources allocated to them
  - Assign personnel to projects ho are expects in key technology areas
  - Never add developers to a late project
  - Place an operational infrastructure into the work environment before the real work starts

- ***Practices for Managing Versions of Software Artefacts***
  - All sources artefacts should be under configuration control
  - All artefacts used to produce an artefact of a delivery should be under configuration control
  - Work within managed, private workspace
  - Save artefacts at the completion of intermediate steps of a larger change
  - Regularly synchronize development with the work of others
  - Define policies for branches, codelines, and workspaces
  - Document identified software defects
  - Create a defined process for requesting and approving changes
  - Apply defect repairs to existing releases and ongoing development efforts
  - Use shared, static build processes and tools
  - Build software on a regular, preferable daily, basis
  - Maintain a unique read-only copy of each release
  - A version manifest should describe each software release
  - Software artefacts that comprise a release should adhere to defined acceptance criteria
  - Configuration management tools should provide release updates
  - Use a software tool perform configuration management functions
  - Repositories should exist on reliable physical storage elements
  - Configuration management repositories should undergo periodic backups
  - Test and confirm the backup process

- ***Practice for Eliciting Requirements***
  - Identify and involve stakeholders
  - Identify the reason for developing a system
  - Define a clear, crisp project vision
  - Identify applicable operational policies
  - Identify user roles and characteristics
  - Describe systems similar to the “to be” system
  - Identify all external interfaces and enabling systems
  - Define a concept of operations
  - Emphasize the definition of vital non-functional requirements
  - Include specific quality targets in the requirements
  - Classify requirements using multidimensional approach
  - Verify the source of a requirement
  - Develop conceptual models
  - Record assumptions
  - Prioritize software requirements
  - Capture requirements rationales and discussions of them
  - Analyze the risk of each requirement
  - Allocate requirements in a top-down manner
  - Define a glossary
  - Uniquely identify each requirement and ensure its uniqueness
  - Differentiate between requirement, goal, and declaration statements
  - Use good writing style
  - Write consistent statements
  - Define a verification method for each requirement
  - Identify the relationships among requirements
  - Do not exclude higher-level requirements
  - Write requirements that are independent of each other
  - Fully specify all requirements
  - Assess the quality of each requirement
  - Validate the completeness of the defined requirements
  - Inspect requirements using a defined process
  - Use bilateral agreements
  - Monitor the status of software requirements following a defined procedure
  - Measure the number and severity of defects in the defined requirements
  - Control how requirements are introduced, changed, and removed

- ***Practices for Designing Architectures***
  - Reduce large systems into module realized by 5,000 to 10,000 lines of source code
  - Use different views to convey different ideas
  - Separate control logic from functions that provide services
  - Define and use common protocols for common operations
  - Provide models of critical system-level concepts
  - Use functions to encapsulate individual behaviours
  - Minimize the use of goto statements
  - Use program structuring techniques that optimize locality of reference
  - Avoid creating and using redundant data

- Design and implement features that only satisfy the needed requirements
- Periodically analyze and simplify software systems
- Create prototype of critical components to reduce risk
- Use a notation having precise semantics to describe software artefacts
- Define and use criteria and weightings for evaluating software design decisions
- Record the rationale for each design decision
- Compute key design metrics

- ***Practice for General-Purpose Programming***
  - Use the highest-level programming language possible
  - Use integrated development environments
  - Adopt a coding standard that prevents common types of defects
  - Prototype user interfaces and high-risk components
  - Define critical regions

- ***Practices for Inspecting Artefacts***
  - Provide explicit training in software inspection techniques
  - Require that the appropriate people inspect artefacts
  - Use checklist-based inspection techniques
  - Use two people to inspect artefacts
  - Conduct meeting-less inspections
  - Generate functional test cases from defined scenarios
  - Use a code coverage tool
  - Perform basis path testing
  - Examine the boundary conditions affecting the control flow of a program
  - Verify data and file usage patterns of a program
  - Verify that invalid and unexpected inputs are handled, as well as valid and expected ones
  - Verify all critical timing modes and time-out conditions
  - Verify that systems work in a variety of configurations
  - Verify the accuracy of the documentation

- ***Practice for Writing Useful User Documentation***
  - Orient the documentation around descriptions of real tasks
  - Organize the presentation of information
  - Provide an overview of the information of each major section of a document
  - Clearly present information
  - Use visual techniques to effectively convey information
  - Provide accurate information
  - Provide complete information

Verzuh suggests that an essential part of project management consists in the *project rules* such as [Verzuh 2005]

1. Agreement on the goals of the project among all parties involved
2. Control over the scope of the project
3. Management support

A ***responsibility matrix*** should be helpful in order to avoid communication breakdowns between departments and organizations. The steps for setting a responsibility matrix are [Verzuh 2005]

1. List the major activities of the project

2. List the stakeholder groups

3. Code the responsibility matrix

4. Incorporate the responsibility matrix into the project rules

Project start should be based on the following steps [Verzuh 2005]

1. The *project proposal* assembles the information necessary for a sponsor of project selection board.

2. A project sponsor can use the *charter* template to formally authorize the project and project manager.

3. The *statement of work* represents the formal agreement between project stakeholders about the goals and constraints of the project.

4. The *responsibility matrix* clarifies the role and authority of each project stakeholder.

5. Effective communication is no accident. Use the *communication planning matrix* to identify who needs what information and how you'll sure to get it to them. Remember that having more mediums of communication increases the likelihood your message will get through.

6. As you initiate the project, use the *definition checklist* to guide the team.

In order to develop the detailed project plan it must consider the following steps [Verzuh 2005]: *create the project definition, develop a risk management strategy, build a work breakdown structure, identify task relationships, estimate work packages, calculate initial schedule, assign and level resources.*

The process of resource levelling also defined by Verzuh should keep the following: *forecast the resource requirements throughout the project for the initial schedule, identify the resource peaks, at each peak, delay non-critical tasks within their float, eliminate the remaining peaks by re-evaluating the work package estimates.*

The typical project constraints are the *time, money* and *resources* [Verzuh 2005]. Furthermore, for balancing the project level these steps should be taken: *re-estimate the project, change task assignments to take advantage of schedule float, add people to the project increase productivity by using experts from within the firm, increase productivity by using experts from outside the firm, outsourcing the entire project or a significant portion of it, crashing the schedule, working overtime.*

Besides, some rules for effective communication in project teams are defined by Verzuh in the following manner [Verzuh 2005]:

1. *Responsibility.* Each team member needs to know exactly what part of the project he or she is responsible for.
2. *Coordination*. As team members carry out their work, he relies on each other. Coordination information enables them to work together efficiently.
3. *Status*. Meeting the goal requires tracking progress along the way to identify problems and take corrective action. The team members must be kept up to seed in the status of the project.
4. *Authorization*. Team members need to know about all the decisions made by customers, sponsors, and management that relate to the project and its business environment. Team members need to know these decisions to keep all project decisions synchronized.

The measurement of progress is one of the essential aspects for controlling the software project. Some rules are [Verzuh 2005]:

- *Measuring schedule performance*: using the 0-50-100 rule, take completion criteria seriously, schedule performance measures accomplishment, measuring progress when there are many similar tasks

- *Measuring cost performance*: every work package has cost estimates for labour, equipment, and materials; as each one is executed, be sure to capture the actual costs

- *Earned value reporting*: calculating the cost variance using earned value, us the cost variance to identify problems early.

Finally, Verzuh defines the following project management model (ERM) [Verzuh 2005].

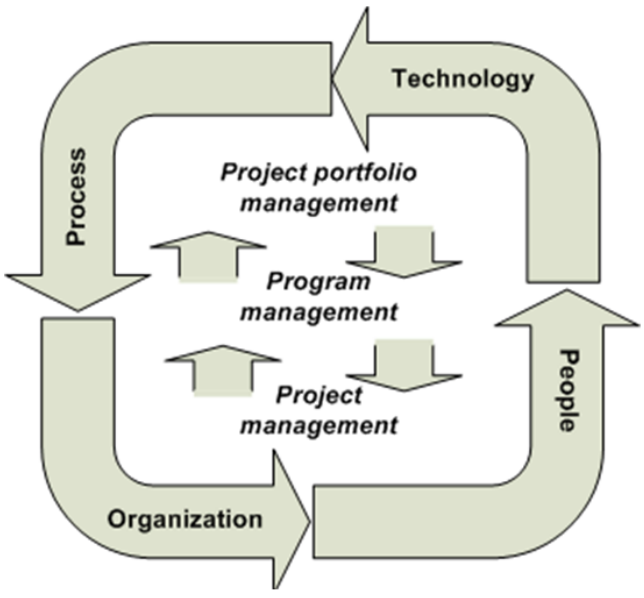


Figure 32: The enterprise project management model

The ***Zachman's Framework*** includes a two-dimensional classification of the various components of an information system in the following manner [Keyes 2003]

- *First framework dimension*: scope description, business model, information-system model, technology model, detailed description

- *Second framework dimension*: data description, process description, and network description

The following figure shows the essential components of the ***IT Balanced Scorecard*** defined by Gadatsch and Mayer [Gadatsch 2005].

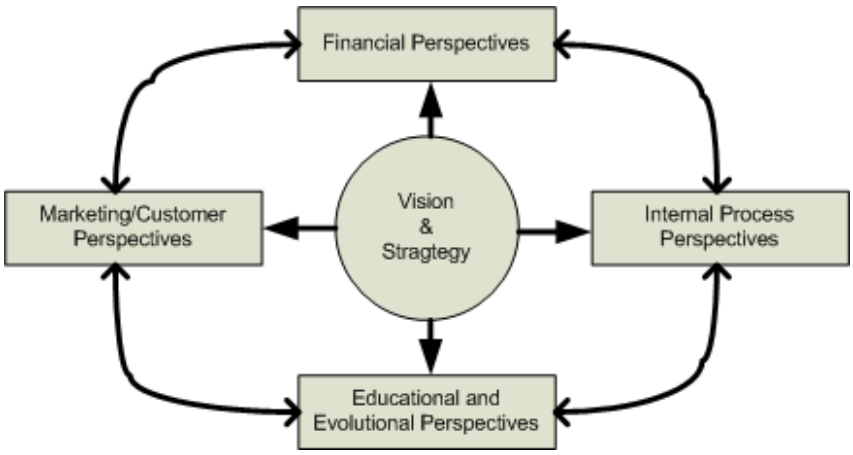


Figure 33: Schema of a IT Balanced Scorecard

The ***Corbin's Methodology for Establishing a Software Development Environment*** (SDE) includes the following procedures and issues (see [Keyes 2003])

- *The elements of SDE:* project management, business plan, architecture, methodologies, techniques, tools, ***metrics***, policies and procedures, technology platform, support, standards, education and training

- *The benefits of SDE:* improved problem definition, selection of the "right" problem according to the customer, joint customer and IS responsibilities and accountability, acknowledgement of customer ownership of system, reduced costs of system development and maintenance, reusability of software, models, and data definitions, acceptance of the disciplined approach to software engineering using a consistent methodology, productivity improvements through team efforts and tools such as CASE

- *Sample goals of SDE:* reduce system development costs, reduce maintenance costs, reduce MIS turnover rate

The ***Shetty's Seven Principles of Quality Leaders*** are the following (see [Keyes 2003])

1. Establish and communicate a clear vision of corporate philosophy, principles, and objectives relevant of product and service quality

2. Quality is a strategic issue

3. Employees are the key to consistent quality

4. Quality standards and measurement must be customer-driven

5. Many programs and techniques can be used to improve quality

6. All company activities have potential for improving product quality

7. Quality is a never-end process

The ***Kemayel's Controllable Factors in Programmer Productivity*** consists of the following principles and issues (see [Keyes 2003])

1. *Programmer productivity paradoxes*: There is enormous variance in the productivity of programmers, productivity invariance with respect to experience, productivity invariance with respect to tools, suitability of motivation factors

2. The *33 productivity factors* that are proposed can be divided into three categories: factors related to personnel, factors related to the software process, factors related to the user community

3. *Personnel factors*: two sets of controllable factors are likely to affect the productivity of data processing personnel: *motivation factors* and *experience factors*

4. *Personnel motivation* consists of many factors, 16 derive from research appear below: *recognition, achievement, the work, responsibility, advancement, salary, possibility of growth, interpersonal relations with subordinates, status, interpersonal relations: superiors, interpersonal relations: peers, technical supervision, company policy and administration, working conditions, factors interpersonal life, job security*

5. *Personal experience* is equally important.

6. Two classes of controllable factors pertaining to the software process have been identified by the authors: *project management* and *programming environments*

7. *Project management* consists of four controllable factors: *using a goal structure, adherence to a software life cycle, adherence to an activity distribution, usage of cost estimation procedures*

8. Programming environment is composed of four controllable factors: *programming tools, modern programming practice, programming standards, power of equipment used*

9. The *participation of users* has been found to have an important impact on programmer productivity.

The ***Redmill's Quality Considerations in the Management*** of software-based development projects was defined in five steps as following (see [Keyes 003])

1. Most common reasons given by project managers for failure to meet budget, time scale, and specification are as follows: *incomplete and ambiguous requirements, incomplete and imprecise specifications, difficulties in modelling systems, uncertainties in cost and resource estimation, general lack of visibility, difficulties with progress monitoring, complicated error and change control,* ***lack of agreed-upon metrics****, difficulties in controlling maintenance, lack of terminology, uncertainties in software or hardware apportionment, raid changes in technology, determining suitability of languages,* ***measuring and predicting reliability,*** *problems with interfacing, problems with integration*

2. Audits of systems development efforts reveal shortcomings in projects: *lack of standards, failure to comply with existing standards, non-adherence to model in use, no sin-off at end of stages, lack of project plans, no project control statistics recorded or stored, no quality assurance procedures, no change-control procedures, no configuration control procedures, no records of test data and results*

3. The three causes for the lack of control of projects: *attitude to quality, attitude to management, attitude to project*

4. In finding solutions, the principal reasons for project management shortcomings should be reviewed.

5. Solutions: *Training, management, standards, guidelines, procedures, and checklists, quality assurance (QA), QA team, audits, planning, reporting, feedback, continuo review, project manager, non-technical support team.*

The ***Hewlett Packard's TQC Guidelines for Software Engineering Productivity*** involves the following procedures and policies (see [Keyes 2003])

- The HP's productivity equation

  $Productivity = function\_of\_doing\_the\_right\_things \times function\_of\_doing\_things\_right$

- Cultural organizational issues are addressed to be able to motivate support positive changes. Productivity managers are used in each division: *understand productivity and quality issues, evaluate, select, and install CASE tools, communicate best software engineering practices, training, establish* ***productivity and quality metrics****, a group productivity council created to share the best R&D practices across divisions,* ***metrics definition, metrics tracing,*** *productivity councils, software quality and productivity assessment, communication best practices*
- A ***software metrics council*** was created consisting of QA managers and engineers whose objective was to identify key software metrics and promote their use.

- ***Project/product quality metrics****: break-even time measures return on investment, time-to-market measures responsiveness and competitiveness, kiviat diagram measures variables that affect software quality and productivity.*
- ***Progress quality metrics:*** *turnover rate measures morale, training measures investment in career development.*
- Basic ***software quality metrics:*** *Code size (KNCSS which is thousands of lines noncomment source statements), number of pre-release defects requiring fix, pre-release defect density, calendar months for pre-release QA, total pre-release QA test hours, number of post-release defect reported after one year, post-release defect density, calendar months from investigation checkpoint to release.*
- The system software certifications program was established to ensure measurable, consistent, high-quality software. The four metrics chosen were: *breadth* (measures the testing coverage of user-accessible and internal functionality of the product), *depth* (measures the proportion of instructions or blocks of instructions executed during the testing process), *reliability* (measures the stability and robustness of a product and its ability to recover gracefully from error conditions, *defect density* (measures the quantity and severity of reported defects found and a product's readiness for use).

4.4 Software Process Rules of Thumb

Considering the process related aspects in requirements engineering, Ebert has founded the following general experience about the project phases [Ebert 2005].

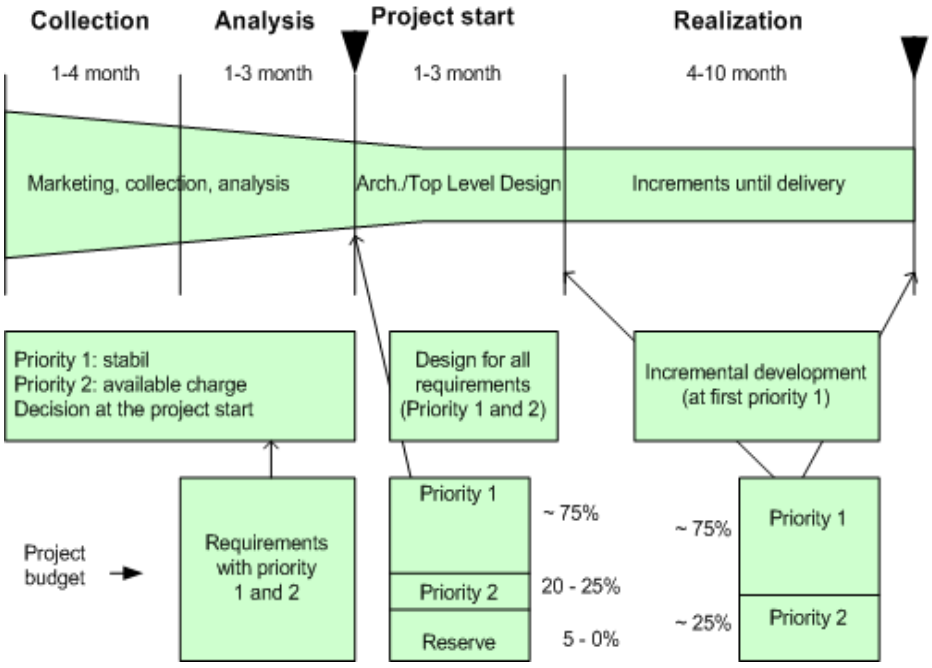


Figure 34: Project definition, priorities and incremental development

An estimation of the expenditures based on activity for a conventional project is given by [Royce 1998] shown in the following table.

ACTIVITY	COST
Management	5%
Requirements	5%
Design	10%
Code and unit testing	30%
Integration and test	40%
Deployment	5%
Environment	5%
Total	100%

Table 4: Expenditures by activity for a conventional software project

Two examples of the rules of thumb are given by Verzuh in [Verzuh 2005] considering the cost of mistakes in a project:

> *"If a defect caused by incorrect requirements is fixed in the construction of maintenance phase, it can cost 50 to 200 times as much to fix as it would have in the requirements phase."*

> *"Each hour spent on quality assurance activities such as design reviews saves 3 to 10 hours on downstream costs."*

Experiences related to the function points (FP) are summarized by Sneed [Sneed 2005] and consider the "produced" FP per hour during the software development in different industrial domains shown in the following diagram.

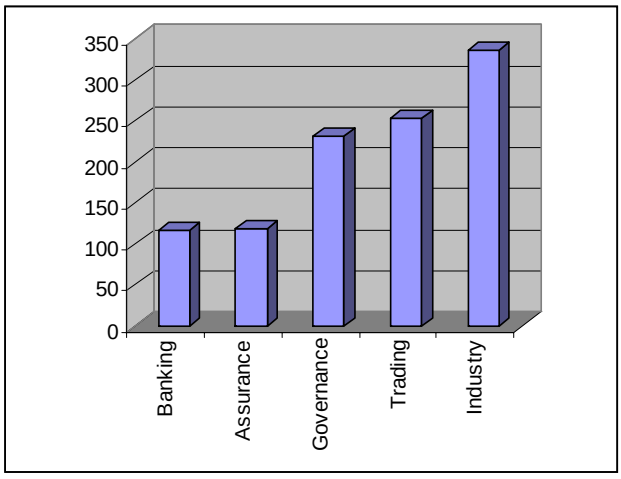


Figure 35: Function Points per hour in different IT domains

4.5 Software Process Experiments

Experiments are usually performed in an environment resembling a laboratory to ensure a high amount of control while carrying out the experiment. The assignments of the different factors for the experiment are allotted totally at random. More about this random assignment can be found in the following sections. The main task of an experiment is to manipulate variables and to measure the effects they cause. This measurement data is the

basis for the statistical analysis that is performed afterwards. In case that it is not possible to assign the factors through random assignment, so-called quasi-experiments can be used instead of the experiments described above.

Experiments are used for instance to confirm existing theories, to validate measures or to evaluate the accuracy of models [Wohlin 2000]. Other than surveys and case studies the experiments only provide data for a quantitative study. The difference between case studies and experiments is that case studies have a more observational character. They track specific attributes or establish relationships between attributes but do not manipulate them. In other words they observe the on-going project. The characteristic of an experiment in this case is that control is the main aspect and that the essential factors are not only identified but also manipulated.
It is also possible to see a difference between case studies and surveys. A case study is performed during the execution of a project. The survey looks at the project in retrospect. Although it is possible to perform a survey before starting a project as a kind of prediction of the outcome, the experience used to do this is based on former knowledge and hence based on those experiences gained in the past.

Carrying out experiments in the field of Software Engineering is different from other fields of application [Juristo 2003]. In software engineering several aspects are rather difficult to establish. These are: Find variable definitions that are accepted by everyone, Prove that the measures are nominal or ordinal scale, Validation of indirect measures: models and direct measures have to be validated.

To be able to carry out an experiment several steps have to be performed [Basili 1986]: The definition of the experiment, The planning, Carrying out the experiment, Analysis and Interpretation of the outcomes, Presentation of the results.

Now we take a more detailed look on the different steps mentioned above. The Experiment definition is the basis for the whole experiment. It is crucial that this definition is performed with some caution. When the definition is not well founded and interpreted the whole effort spent could have been done in vain and one worse thing to happen is that the result of the experiment is not displaying what was intended. The definition sets up the objective of the experiment. Following a framework can do this. The GQM templates could supply such a framework for example [Solingen 1999].

After finishing the definition, the planning step has to be performed. While the previous step was to answer the question why the experiment is performed, this step answers the question how the experiment will be carried out. 6 different stages will be needed to complete the planning phase [Wohlin 2000].

Context selection: The environment in which the experiment will be carried out is selected.

Hypothesis formulation and variable selection: Hypothesis testing is the main aspect for statistical analysis when carrying out experiments. The goal is to reject the hypothesis with the help of the collected data gained through the experiment. In the case that the hypothesis is rejected it is possible to draw conclusion out of it. More details about hypothesis testing can be read in the following sections. The selection of variables is a

difficult task. Two kinds of variables have to be identified: dependent and independent ones. This also includes the choice of scale type and range of the different variables. The section above also contains more information about dependent and independent variables.

Subject selection: It is performed through sampling methods. Different kinds of sampling can be found at the end of this chapter. This step is the fundament for the later generalisation. Therefore the selection chosen here has to be representative for the whole population. The act of sampling the population can be performed in two ways either probabilistic or non-probabilistic. The difference between those two methods is that in the latter the probability of choosing a sample of the selection is not known. Simple random sampling and systematic sampling, just to name two, are probability-sampling techniques. Those and other methods can be found at the end of this chapter. The size of the sample also has influence on the generalisation. A rule of thumb is that the larger the sample is the lower the error in generalising the results will be. There are some general principles described in [Juristo 2003]:

- If there is large variability in the population, a large sample size is needed.
- The analysis of the data may influence the choice of the sample size. It is therefore needed to consider how the data shall be analysed already at the design stage of the experiment.

Experiment design: The design tells how the tests are being organized and performed. An experiment is so to speak a series of tests. A close relationship between the design and the statistical analysis exists and they have effect on each other. The choices taken before (measurement scale, etc.) and a closer look at the null-hypothesis help to find the appropriate statistical method to be able to reject the hypothesis. The following sections provide a deeper view into the subject described shortly above.

Instrumentation: In this step the instruments needed for the experiment are being developed. Therefore three different aspects have to be addressed: experiment objects (i.e. specification and code documents), guidelines (i.e. process description and checklists) and measurement. Using instrumentation does not affect the outcome of the experiment. It is only used to provide means for performing and to monitor experiments [Wohlin 2000].

Validity evaluation: After the experiments are carried out the question arises how valid the results are. Therefore, it is necessary to think of possibilities to check the validity.

The following components are an important vocabulary needed for the software engineering experimentation process: *Dependent & Independent variables*: Variables that are being manipulated or controlled are called independent variables. When variables are used to study the effects of the manipulation etc. they are called dependent; *Factors*: independent variables that are used to study the effect when manipulating them. All the other independent variables remain unchanged; *Treatment*: a specific value of a factor is called treatment; *Object & Subject*: an example for an object is a review of a document. A subject is the person carrying out the review. Both can be independent variables; *Test (sometimes referred to as Trial)*: an experiment is built up using several tests. Each single test is structured in treatment, objects and subjects. However, these tests should not be mixed up with statistical tests, *Experimental error*: gives an indication of how much confidence can be put in the experiment. It is affected by how many tests have been carried out; *Validity:* there

are four kinds of validity: internal validity (validity within the environment and reliability of the results), external validity (how general are the findings), construct validity (how does the treatment reflects the cause construct) and conclusion validity (relationship between treatment and outcome), *Randomisation:* the analysis of the data has to be done from independent random variables. It can also be used to select subjects out of the population and to average out effects, *Blocking:* is used to eliminate effects that are not desired, *Balancing:* when each treatment has the same number of subjects it is called balanced.

Software engineering experimentation could be supported by the following sampling methods [Wohlin 2000]: *Simple random sampling*: the subjects that are selected are randomly chosen out of a list of the population. *Systematic sampling*: only the first subject is selected randomly out of the list of the population. After that every n-the subject is chosen. *Stratified random sampling*: first the population is divided into different strata, also referred to as groups, with a known distribution between the different strata. Second the random sampling is applied to every stratum. *Convenience sampling*: the nearest and most convenient subjects are selected. *Quota sampling*: various elements of the population are desired. Therefore convenience sampling is applied to get every single subject.

CONTROLLED EXPERIMENTS: The advantage of this approach is that it promotes comparison and statistical analysis. Controlled here means that the experiment follows the steps as mentioned above (Basili 1986], [Zelkowitz 1997]):

Experiment definition: it should provide answers to the following questions: *"what is studied?"* (object of study)*,"what is the intention?"* (purpose)*, "which effect is studied?"* (quality focus)*, "whose view is represented?"* (perspective) *and "where is the study conducted?"* (context).

Experiment planning: null hypothesis and alternative hypothesis is formulated. The details (personnel, environment, measuring scale, etc.) are determined and the dependent and independent variables are chosen. First thoughts about the validity of the results.

Experiment realization: the experiment is carried out according to the baselines established in the design and planning step. The data is collected and validated.

Experiment analysis: the data collection gathered during the realization is the basis for this step. First descriptive statistics are applied to gain an understanding of the submitted data. The data is informally interpreted. Now the decision has to be made how the data can be reduced. After the reduction the hypothesis test is performed. More about hypothesis testing can be found in the following sections.

Portrayal of the results and conclusion about the hypothesis: the analysis provides the information that is needed to decide whether the hypothesis was rejected or accepted. These conclusions are collected and documented. This thesis comprises the lessons learned.

The quality of the design decides whether the study is a success or a failure. So it is very important to meticulously design the experiment [Juristo 2003]. Several principles of how to design an experiment are known. Those are randomisation, blocking and balancing. In general a combination of the three methods is applied. The experimental design can be

divided into several standard design types. The difference between them is that they have distinct factors and treatment. The first group relies on one factor, the second on two and the third group on more than two factors.

4.6 Software Process Case Studies

A *case study* is used to monitor the project. Throughout the study data is collected. This data is then investigated with statistical methods. The aim is to track variables or to establish relationships between different variables that have a leading role or effect on the outcome of the study. With the help of this kind of strategy it is possible to build a prediction model. The statistical analysis methods used for this kind of study consists of linear regression and principle component analysis. A disadvantage of this study is the generalisation. Depending on the kind of result it can be very difficult to find a corresponding generalisation. This also influences the interpretation and makes it more difficult. Like the survey the case study can provide data for both qualitative and quantitative research.

The following table shows an overview about used management practices in European companies from Dutta et al. cited from [Emam 2005].

Organizational Structure and Management Practices	**Adoption Percentage**
Nominating software project managers for each project	92
Having the software project manager report to a business project manager responsible for the project's overall benefits to the business	81
Software quality assurance function with independent reporting line from software development project management	48
Establishing a change control function for each project	55
Required training program for new project managers to familiarize them	40
Maintaining awareness of CASE or other new software engineering technologies	41
Ensuring user/customer/marketing input at all stages of the project	64
Ensure availability of critical non-software resources according to plan	45

Table 5: Percentage of respondents in a European survey of management practices

An overview about the delivered defects per Function Points is shown in the following table by [Emam 2005].

	Small projects		*Medium projects*		*Large projects*	
Business Domain	*Average*	*Best*	*Average*	*Best*	*Average*	*Best*
MIS	0.15	0.025	0.588	0.066	1.062	0.27
System software	0.25	0.013	0.44	0.08	0.726	0.15
Commercial	0.25	0.013	0.495	0.08	0.792	0.208
Military	0.263	0.013	0.518	0.04	0.816	0.175

Table 6: Percentage of respondents in a European survey of management practices

The following table shows the distribution of software process activities for different kinds of projects by [Emam 2005].

Process activity	System project (%)	Commercial project (%)	Military project (%)
Design	21	16	19
Requirements Definition	11	6	13
Project Management	17	16	17
Documentation	10	16	13
Change Management	14	8	15
Coding	27	39	23

Table 7: Percentages of process activities in different kinds of projects

The following case study from Rubin is cited from [Emam 2005] and considers QA and metrics programs in companies worldwide.

Business Domain	Existence of a QA Function (%)	Existence of a Metrics Program (%)
Aerospace	52	39
Computer manufacturing	70	42
Distribution	16	-
Finance	57	25
Government	30	5
Health	51	8
Manufacturing	52	25
Oil and gas	40	20
Software	60	36
Telecommunication	54	44
Transportation	33	33
Utility	40	10

Table 8: Percentage of Organizations having QA and metrics efforts in place Based on a worldwide survey

4.7 Software Process Metrics and Measures

A special form of ***formulas for measuring software reliability*** based on the failure rates and probabilistic characteristics of software systems are [Singpurwalla 1999]:

- *Jelinski-Moranda model*: Jelinski and Moranda assume that the software contains an unknown number of, say *N,* of bugs and that each time the software fails, a bug is detected and corrected and the failure rate T_i is proportional to $N - i + 1$ the number of remaining the code.

- *Baysian reliability growth model*: This model devoid a consideration that the relationship between the relationship between the number of bugs and the frequency of failure is tenuous.

- *Musa-Okumoto models*: These models are based on the postulation a relationship between the intensity function and the mean value function of a Poisson process that has gained popularity with users.

- *General order statistics models*: This kind of models is based on statistical order functions. The motivation for ordering comes from many applications like hydrology, strength of materials and reliability.

- *Concatenated failure rate model*: These models introduce the infinite memories for storage the failure rates where the notion infinite memory is akin to the notion of invertibility in time series analysis.

A simple evaluation of the priorities of the requirements based on a relationship matrix is defined by Kandt in the following manner [Kandt 2006]:

$$p_i = \sqrt[n]{\prod_{j=1}^{n} a_{i,j}} \tag{4.3}$$

The priorities of each attribute $a_{i,j}$ were executed as an approximation by computing p_i. Another formula by Kandt helps to evaluate the SQA situation as

Requirements coverage = (*Number of Requirements traced to functional test cases*)/
(Number of requirements) (4.4)

System architecture statement coverage = (*Executed SLOC of system architecture*)/
(Total SLOC of system architecture)

System architecture edge coverage =
(Executed decision outcomes of system architecture)/
(Total decision outcomes of system architecture)

System Statement coverage = (Executed SLOC of system)/(Total SLOC of system)

System edge coverage = (Executed decision outcomes of system)/
(Total decision outcomes of system)

Otherwise, the defect estimation techniques are summarized by Kandt in the following manner [Kandt 2006]

$$D_1 = (l \times d) - D_d \tag{4.5}$$

where D_1 stands for the number of remaining defects in a module, l is the number code lines, d is the typical number of defects per source line of code, and D_d is the number of detected defects.

$$D_2 = ((N_1 + N_2)\log(n_1 + n_2))/3000 - D_d$$

as an estimation based on the Halstead's software science. Finally as a capture-recapture technique for defect estimation the formula

$$D_3 = (m_1 \times m_2)/(m_{12} - (m_1 + m_2 - m_{12}) \quad (4.6)$$

where m_1 and m_2 are the number of defects found in these research groups and m_{12} denotes the common defects found in both groups.

The ***customer cost*** of a software product was executed by Emam [Emam 2005] in the following manner.

$$Customer\ Cost = Defect\_density \times KLOC \times Cost\_per\_defect \times Defects\_find\_by\_customer \quad (4.7)$$

The ***return on investment (ROI)*** was executed by Emam as [Emam 2005] as

$$ROI_1 = (Cost\ saved - Investment)/ Investment \quad (4.8)$$

$$ROI_2 = (Cost\ saved - Investment)/ Original\ cost$$

$$New\ cost = Original\ cost \times (1 - ROI_2)$$

$$Schedule\ reduction = (Original\ schedule - New\ schedule)/ Original\ schedule \text{ [personal month]}$$

The general relationship between different indicators of quality, quantity, effort and productivity are defined by Sneed in the following manner [Sneed 2005]:

1. $quantity = (productivity \times effort) / quality$ (4.9)
2. $quality = (productivity \times effort) / quantity$
3. $productivity = (quantity \times quality) / effort$
4. $effort = (quantity \times quality) / productivity$

Especially, different kinds of software process effort estimation are using the ***point approach*** [Lother 2001]. Some of these point metrics are:

(IFPUG) Function Points: The function point method is based on counting system components relating to their functionality such as *input, output, inquiries, files,* and *interfaces* ([Albrecht 1983], [Dreger 1989]). These characteristics were weighted by a classification of *simple, average* and *complex* (s, a, c) and leads to the (unadjusted) function points (UFP) as

$$UFP = a \times inputs + b \times outputs + c \times requires + d \times files + e \times interfaces \quad (4.10)$$

with the (s, a, c) for a =(3,4,6), b=(4,5,7), c=(3,4,6), d=(7,10,15), and e=(5,7,10). The *adjusted function points* (FP) are executed by application of a weighted number (0 … 5) for every 14 factors (cost drivers) as *data communication, distributed functions, performance requirement, hardware configuration, high transaction rate, online data entry, end-user efficiency, online update, complex processing, reusable requirements, ease of installation, operational ease, multiple sites,* and *ease of modification.* The special kind of execution is

$$FP = 0.65 + 0.01 \times cost\ drivers$$

The effort estimation is based on experience data and could be executed by [Bundschuh 2000]

$$Person\ month \approx 0.015216\ FP^{1.29}$$

The IFPUG Function Point method is well-established and was supported by the *International Function Point User Group* (IFPUG).

Mark II Function Points: This method is modification of the function point method described above by changing the viewpoint to the data-based system approach [Symons 1993]. The counting characteristics are *input, entities referenced,* and *output.* The weight factors are quite different to the FP method (0.58 for inputs, 1.66 for entities referenced and 0.26 for outputs).

$$FP = 0.58\ W_i + 1.66\ W_e + 0.26\ W_o \tag{4.11}$$

The 14 FP adjustment factors were extended by six other factors considering actual system aspects and leads to the possibility of effort estimation.

Data Points: The data point method was created by Sneed and is based on the analysis of information systems [Sneed 1990]. The general execution of the data point is

$$Data\ point = information\ points + communication\ points \tag{4.12}$$

The information points are counted from the data model and the communication points evaluate the user interface. The estimation process was supported by different weight factors for the different system

Object Points: One of the objects point method was defined by Sneed and consider the different characteristics of OO system design [Sneed 1996]. The counted elements for object points (OP) are

- in the *class diagram*: class=4, non inherited attribute: 1, non inherited method: 3, class association: 2

- in the *sequence diagram*: message: 2, parameter: 1,sender: 2, potential receiver: 2

- in the *use case diagram*: online use case: 2×#outputs, batch use case: 4×#outputs, system use case: 8×#outputs

The consideration of the complexity leads a classification of low (75 percent of the OP), average complexity (100 percent of the OP), and high complexity (12 percent of the OP).

Feature Points: The feature point method (FPM) was defined by Jones considers the other/new kinds of systems like real time, embedded or communication software [Jones 1991]. The execution of the unadjusted feature points is

$$FPM = \#algorithms \times 3 + \#inputs \times 4 + \#outputs \times 5 + \#inquiries \times 4 + \#data\_files \times 7 + \#interfaces \times 7 \quad (4.13)$$

In order to estimate the effort adjustment principle was used like in the IFPUG FP methodology described above.

3-D Function Points: This point metric considers the following three evaluation areas (dimensions) and was defined by Whitmire [Whitmire 1992]:

- the data model according to IFPUG FP),

- the functional model considering the number of functions and their complexity

- the process model counting the control statements as system states and state transitions

Use Case Points: The use case point metric is addressed to UML-based software modelling and implementing (see [Sneed 2005]). The use case points (UCP) are computed as

$$UCP = TCP \times ECF \times UUCP \times PF \quad (4.14)$$

where TCP stands for the technical complexity factors which evaluate by weights the technological type of the system such as distributed system, reusability, concurrent etc., ECF the environmental complexity factors which characterize the system background like stability of the requirements, experience in OO and UML etc., UUCP the unadjusted use case points which counts the different use case diagram components, PF the productivity factors which weights the UCP considering the person hours per use case.

COSMIC FFP: The COSMIC Full Function Point (FFP) method was developed in the Common Software Measurement International Consortium (COSMIC) and is established as ISO/IEC 19761 (see [Ebert 2004]). A full function point only considers a *data movement* which means that there are not a (weighted) difference between inputs, outputs etc. The Cfsu (COSMIC functional size unit) is the FFP measurement unit. The basic for COSMIC FFP counting is

$$CFP = counting(((entry,exits),(reads,writes))_{archictureLevel\ i}) \text{ [Cfsu]} \qquad (4.15)$$

The COSMIC FFP measurement method is designed to be independent of the implementation decisions embedded in the operational artefacts of the software to be measured. To achieve this characteristic, measurement is applied to the (functional user requirement) FUR of the software to be measured expressed in the form of the COSMIC FFP *generic software model*. This form of the FUR is obtained by a mapping process from the FUR as supplied in or implied in the actual artefacts of the software. The architectural reasoning of boundaries is given through the *software layers* such as tiers, service structures or component deployments. The functional size of software is directly proportional to the number of its *data transactions.* All data movement sub processes move data contained in exactly one data group. Entries move data from the users across the boundary to the inside of the functional process; exits move data from the inside of the functional process across the boundary to the users; reads and writes move data from and to persistent storage.

An overview about the history of function points is shown in the following figure created in [Fetcke 1999], [Lother 2001] and [Dumke 2005a].

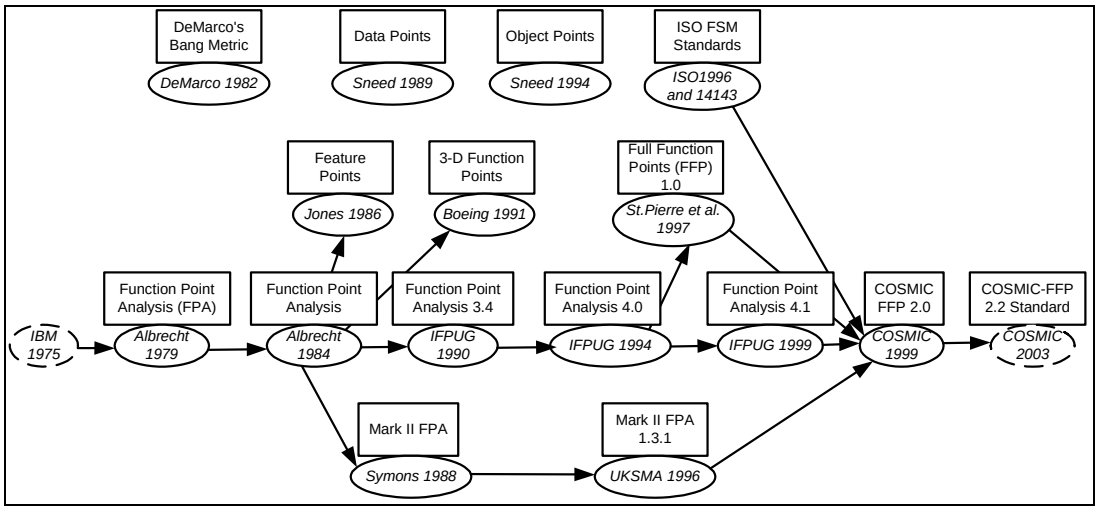


Figure 36: The history of function point methods development

Further methods of estimation are based on the ***size*** of the developed software system. Examples of these estimation methods are (see also [Bielak 2000], [Boehm 2000a], and [Hale 2000]):

COCOMO: The Constructive Cost Model (COCOMO) was defined by Boehm [Boehm 1984] and is based on the formula

$$Personal\ effort = scale\_factors \times KDSI^{type\_of\_project}\ [PM] \quad (4.16)$$

where KDSI means *Kilo Delivered Source Instruction* that must be estimated at the beginning. The scale factors define the *cost drivers* Boehm classify three types of projects: organic, semidetached, and embedded.

COCOMO II: The COCOMO II approach extends the set of cost drivers and considers the different/new aspects of software systems like code adaptation, reuse and maintenance. Furthermore, it is possible to execute/estimate the development time *TDEV* as

$$TDEV = scale\_factors \times PM^{calibration} \quad (4.17)$$

Helpful variants of COCOMO II are *COPSEMO* (Constructive Phased Schedule and Effort Model), *CORADMO* (Constructive Rapid Application Development cost Model), *COCOTS* (Constructive COTS cost model), *COQUALMO* (Constructive Quality cost Model) and *COPROMO* (Constructive Productivity cost Model). A special kind of COCOMO is called as *early design model equation* and was executed by (see also [Keyes 2003])

$$Effort = KLOC \times adjustment\_factor$$

SLIM: Considering the *Software Life Cycle Management* (SLIM) Putnam adapted the Rayleigh curve for the software development area in the following manner [Putnam 1992]

$$Current\_effort = (Total\_effort/duration) \times t \times e^{(-t \times t/2 \times duration)} \quad (4.18)$$

where *duration* stands for the square of total duration of the development and *t* means the time point of evaluation. The current effort was measured in *personal years.* Another kind of estimation based on the Rayleigh formula is known as ***software equation*** (see also [Keyes 2003]) as

$$System\_size = technology\_constant \times Total\_effort^{1/3} \times duration^{2/3}$$

where the *technology_constant* depends on the development methodology.

WOA: The Weighted Average of Individual Offsets (WOA) model supports the defect estimation based on inspection data [Biffl 2000]. The WOA model uses weights for individual estimation contributions and calculates the number of unique defects found by e team as

$$\#defects = D + \Sigma((defect\_before\text{-}inspection - exported\_defects) \times weights) / \Sigma\, weights \quad (4.19)$$

A special method of project visualization is defined by Hansen and uses different colours in order to mark different levels of development like *implementation proposal, function description, design description, code* and *test* [Hansen 2006].

The following set of metrics is defined by Kulpa and Johnson in order to keep the quantified requirements for the different CMMI levels [Kulpa 2003].

CMMI LEVEL 2 Metrics: (4.20)

Requirements Management

1. Requirements volatility- (percentage of requirements changes)
2. Number of requirements by type or status (defined, reviewed. approved. and implemented)
3. Cumulative number of changes to the allocated requirements, including total number of changes proposed, open, approved, and incorporated into the system baseline
4. Number of change requests per month, compared to the original number of requirements for the project
5. Amount of time spent, effort spent, and cost of implementing change requests
6. Number and size of change requests after the Requirements phase is completed
7. Cost of implementing a change request
8. Number of change requests versus the total number of change requests during the life of the project
9. Number of change requests accepted but not implemented
10. Number of requirements (changes and additions to the baseline)

Project Planning

11. Completion of milestones for the project planning activities compared to the plan (estimates versus actuals)
12. Work completed, effort and funds expended in the project planning activities compared to the plan
13. Number of revisions to the project plan
14. Cost, schedule, and effort variance per plan revision
15. Replanning effort due to change requests
16. Effort expended over time to manage the hmject compared to the plan
17. Frequency, causes, and magnitude of the replanning effort

Project Monitoring and Control

18. Effort and other resources expended in performing monitoring and oversight activities
19. Change activity for the project plan, which includes changes to size estimates of the work products, cost/resource estimates, and schedule
20. Number of open and closed corrective actions or action items
21. Project milestone dates (planned versus actual)
22. Number of project milestone dates made on time
23. Number and types of reviews performed
24. Schedule, budget, and size variance between planned and actual reviews
25. Comparison of actuals versus estimates for all planning and tracking items

Measurement and Analysis

26. Number of projects using progress and performance measures
27. Number of measurement objectives addressed

Supplier Agreement Management

28. Cost of the COTS (commercial off-the-shelf) products
29. Cost and effort to incorporate the COTS products into the project
30. Number of changes made to the supplier requirements
31. Cost and schedule variance per supplier agreement
32. Costs of the activities for managing the contract compared to the plan
33. Actual delivery dates for contracted products compared to the plan
34. Actual dates of prime contractor deliveries to the subcontractor compared to the plan
35. Number of on-time deliveries from the vendor, compared with the contract
36. Number and severity of errors found after delivery
37. Number of exceptions to the contract to ensure schedule adherence
38. Number of quality audits compared to the plan
39. Number of Senior Management reviews to ensure adherence to hudget and schedule plan
40. Number of contract violations by supplier or vendor

Process and Product Quality Assurance (QA)

41. Completions of milestones for the QA activities compared to the plan
42. Work completed, effort expended in the QA activities compared to the plan
43. Number of product audits and activity reviews compared to the plan
44. Number of process audits and activities versus those planned

45. Number of defects per release and/or build
46. Amount of time/effort spent in rework
47. Amount of QA time/effort spent in each phase of the life cycle
48. Number of reviews and audits versus number of defects found
49. Total number of defects found in internal reviews and testing versus those found by the customer or end user after delivery
50. Number of defects found in each phase of the life cycle
51. Number of defects injected during each phase of the life cycle
52. Number of noncompliances written versus the number resolved
53. Number of noncompliances elevated to senior management
54. Complexity of module or component (McCabe, MeClure, and Halstead metrics)

Configuration Management (CM)

55. Number of change requests or change board requests processed per unit of time
56. Completions of milestones for the CM activities compared to the plan
57. Work completed, effort expended, and funds expended in the CM activities
58. Number of changes to configuration items
59. Number of configuration audits conducted
60. Number of fixes returned as "Not Yet Fixed"
61. Number of fixes returned as "Could Not Reproduce Error"
62. Number of violations of CM procedures (non-compliance found in audits)
63. Number of outstanding problem reports versus rate of repair
64. Number of times changes are overwritten by someone else (or number of times people have the wrong initial version or baseline)
65. Number of engineering change proposals proposed, approved, rejected, and implemented
66. Number of changes by category to code source, and to supporting documentation
67. Number of changes by category, type, and severity
68. Source lines of code stored in libraries placed under configuration control

CMMI LEVEL 3 Metrics:

Requirements Development

69. Cost, schedule, and effort expended for rework
70. Defect density of requirements specifications
71. Number of requirements approved for build (versus the total number of requirements)
72. Actual number of requirements documented (versus the total number of estimated requirements)
73. Staff hours (total and by Requirements Development activity)
74. Requirements status (percentage of defined specifications out of the total approved and proposed; number of requirements defined)
75. Estimates of total requirements, total requirements definition effort, requirements analysis effort, and schedule
76. Number and type of requirements changes

Technical Solution

77. Cost, schedule, and effort expended for rework
78. Number of requirements addressed in the product or productcomponent design
79. Size and complexity of the product, product components, interfaces, and documentation
80. Defect density of technical solutions work products (number of defects per page)
81. Number of requirements by status or type throughout the life of the project (for example, number defined, approved, documented, implemented, tested, and signed-off by phase)
82. Problem reports by severity and length of time they are open
83. Number of requirements changed during implementation and test
84. Effort to analyze proposed changes for each proposed change and cumulative totals
85. Number of changes incorporated into the baseline by category (e.g., interface, security, system configuration, performance, and useability)
86. Size and cost to implement and test incorporated changes, including initial estimate and actual size and cost

87. Estimates and actuals of system size, reuse, effort, and schedule 88. The total estimated and actual staff hours needed to develop the system by job category and activity
89. Estimated dates and actuals for the start and end of each phase of the life cycle
90. Number of diagrams completed versus the estimated total diagrams
91. Number of design modules/units proposed
92. Number of design modules/units delivered
93. Estimates and actuals of total lines of code - new, modified, and reused
94. Estimates and actuals of total design and code modules and units
95. Estimates and actuals for total CPU hours used to date
96. The number of units coded and tested versus the number planned
97. Errors by category, phase discovered, phase injected, type, and severity
98. Estimates of total units, total effort, and schedule
99. System tests planned, executed, passed, or failed
100. Test discrepancies reported, resolved, or not resolved
101. Source code growth by percentage of planned versus actual

Product Integration

102. Product-component integration profile (i.e., product-component assemblies planned and performed, and number of exceptions found)
103. Integration evaluation problem report trends (e.g., number written and number closed)
104. Integration evaluation problem report aging (i.e., how long each problem report has been open)

Verification

105. Verification profile (e.g., the number of verifications planned and performed, and the defects found; perhaps categorized by verification method or type)
106. Number of defects detected by defect category
107. Verification problem report trends (e.g., number written and number closed)
108. Verification problem report status (i.e., how long each problem report has been open)
109. Number of peer reviews performed compared to the plan
110. Overall effort expended on peer reviews compared to the plan
111. Number of work products reviewed compared to the plan

Validation

112. Number of validation activities completed (planned versus actual)
113. Validation problem reports trends (e.g., number written and number closed)
114. Validation problem report aging (i.e., how long each problem report has been open)

Organizational Process Focus

115. Number of process improvement proposals submitted, accepted, or implemented
116. CMMI maturity or capability level
117. Work completed, effort and funds expended in the organization's activities for process assessment, development, and improvement compared to the plans for these activities
118. Results of each process assessment, compared to the results and recommendations of previous assessments

Organizational Process Definition

119. Percentage of projects using the process architectures and process elements of the organization's set of standard processes
120. Defect density of each process element of the organization's set of standard processes
121. Number of on-schedule milestones for process development and maintenance
122. Costs for the process definition activities

Organizational Training

123. Number of training courses delivered (e.g., planned versus actual)
124. Post-training evaluation ratings
125. Training program quality surveys
126. Actual attendance at each training course compared to the projected attendance

127. Progress in improving training courses compared to the organization's and projects' training plans
128. Number of training waivers approved over time

Integrated Project Management for IPPD

129. Number of changes to the project's defined process
130. Effort to tailor the organization's set of standard processes
131. Interface coordination issue trends (e.g., number identified and closed)

Risk Management

132. Number of risks identified, managed, tracked, and controlled
133. Risk exposure and changes to the risk exposure for each assessed risk, and as a summary percentage of management reserve
134. Change activity for the risk mitigation plans (e.g., processes, schedules, funding)
135. Number of occurrences of unanticipated risks
136. Risk categorization volatility
137. Estimated versus actual risk mitigation effort
138. Estimated versus actual risk impact
139. The amount of effort and time spent on risk management activities versus the number of risks
140. The cost of risk management versus the cost of actual risks
141. For each identified risk, the realized adverse impact compared to the estimated impact

Integrated Teaming

142. Performance according to plans, commitments, and procedures for the integrated team, and deviations from expectations
143. Number of times team objectives were not achieved
144. Actual effort and other resources expended by one group to support another group or groups, and vice versa
145. Actual completion of specific tasks and milestones by one group to support the activities of other groups, and vice versa

Integrated Supplier Management

146. Effort expended to manage the evaluation of sources and selection of suppliers
147. Number of changes to the requirements in the supplier agreement
148. Number of documented commitments between the project and the supplier
149. Interface coordination issue trends (e.g., number identified and number closed)
150. Number of defects detected in supplied products (during integration and after delivery)

Decision Analysis and Resolution

151. Cost-to-benefit ratio of using formal evaluation processes

Organizational Environment for Integration

152. Parameters for key operating characteristics of the work environment

CMMI LEVEL 4 Metrics:

Organizational Process Performance

153. Trends in the organization's process performance with respect to changes in work products and task attributes (e.g., size growth, effort, schedule, and quality)

Quantitative Project Management

154. Time between failures
155. Critical resource utilization
156. Number and severity of defects in the released product
157. Number and severity of customer complaints concerning the provided service
158. Number of defects removed by product verification activities (perhaps by type of verification, such as peer reviews and testing)
159. Defect escape rates
160. Number and density of defects by severity found during the first year following product delivery or start of service
161. Cycle time

162. Amount of rework time
163. Requirements volatility (i.e., number of requirements changes per phase)
164. Ratios of estimated to measured values of the planning parameters (e.g., size, cost, and schedule)
165. Coverage and efficiency of peer reviews (i.e., number/amount of products reviewed compared to total number, and number of defects found per hour)
166. Test coverage and efficiency (i.e., number/amount of products tested compared to total number, and number of defects found per hour)
167. Effectiveness of training (i.e., percent of planned training completed and test scores)
168. Reliability (i.e., mean time-to-failure usually measured during integration and systems test)
169. Percentage of the total defects inserted or found in the different phases of the project life cycle
170. Percentage of the total effort expended in the different phases of the project life cycle
171. Profile of subprocesses under statistical management (i.e., number planned to be under statistical management, number currently being statistically managed, and number of statistically stable)
172. Number of special causes of variation identified
173. The cost over time for the quantitative process management activities compared to the plan
174. The accomplishment of schedule milestones for quantitative process management activities compared to the approved plan (i.e., establishing the process measurements to be used on the project, determining how the process data will be collected, and collecting the process data)
175. The cost of poor quality (e.g., amount of rework, re-reviews and re-testing)
176. The costs for achieving quality goals (e.g., amount of initial reviews, audits, and testing)

CMMI LEVEL 5 Metrics:

Organizational Innovation and Deployment

177. Change in quality after improvements (e.g., number of reduced defects)
178. Change in process performance after improvements (e.g., change in baselines)
179. The overall technology change activity, including number, type, and size of changes
180. The effect of implementing the technology change compared to the goals (e.g., actual cost saving to projected)
181. The number of process improvement proposals submitted and implemented for each process area
182. The number of process improvement proposals submitted by each project, group, and department
183. The number and types of awards and recognitions received by each of the projects, groups, and departments
184. The response time for handling process improvement proposals
185. Number of process improvement proposals accepted per reporting period
186. The overall change activity including number, type, and size of changes
187. The effect of implementing each process improvement compared to its defined goals
188. Overall performance of the organization's and projects' processes, including effectiveness, quality, and productivity compared to their defined goals
189. Overall productivity and quality trends for each project
190. Process measurements that relate to the indicators of the customers' satisfaction (e.g., surveys results, number of customer complaints, and number of customer compliments)

Causal Analysis and Resolution

191. Defect data (problem reports, defects reported by the customer, defects reported by the user, defects found in peer reviews, defects found in testing, process capability problems, time and cost for identifying the defect and fixing it, estimated cost of not fixing the problem)
192. Number of root causes removed
193. Change in quality or process performance per instance of the causal analysis and resolution process (e.g., number of defects and changes in baseline)
194. The costs of defect prevention activities (e.g., holding causal analysis meetings and implementing action items), cumulatively
195. The time and cost for identifying the defects and correcting them compared to the estimated cost of not correcting the defects
196. Profiles measuring the number of action items proposed, open, and completed
197. The number of defects injected in each stage, cumulatively, and over-releases of similar products
198. The number of defects

4.8 Process Metrics Repositories

The following section includes the main activities for defining and implementing ***measurement repositories*** using in an organizational context. The repository contains both product and process measures that are related to an organization's set of standard processes ([SEI 2002]). It also contains or refers to the information needed to understand and interpret the measures and assess them for reasonableness and applicability. For example, the definitions of the measures are used to compare similar measures from different processes.

Typical Work Products:

1. Definition of the common set of product and process measures for the organization's set of standard processes
2. Design of the organization's measurement repository
3. Organization's measurement repository (i.e., the repository structure and support environment)
4. Organization's measurement data

Sub practices:

1. Determine the organization's needs for storing, retrieving, and analyzing measurements.
2. Define a common set of process and product measures for the organization's set of standard processes. The measures in the common set are selected based on the organization's set of standard processes. The common set of measures may vary for different standard processes. Operational definitions for the measures specify the procedures for collecting valid data and the point in the process where the data will be collected. Examples of classes of commonly used measures include the following:
    - Estimates of work product size (e.g., pages)
    - Estimates of effort and cost (e.g., person hours)
    - Actual measures of size, effort, and cost
    - Quality measures (e.g., number of defects found, severity of defects)
    - Peer review coverage
    - Test coverage
    - Reliability measures (e.g., mean time to failure).

    Refer to the Measurement and Analysis process area for more information about defining measures.
3. Design and implement the measurement repository.
4. Specify the procedures for storing, updating, and retrieving measures.
5. Conduct peer reviews on the definitions of the common set of measures and the procedures for storing and retrieving measures. Refer to the Verification process area for more information about conducting peer reviews.

6. Enter the specified measures into the repository. Refer to the Measurement and Analysis process area for more information about collecting and analyzing data.
7. Make the contents of the measurement repository available for use by the organization and projects as appropriate.
8. Revise the measurement repository, common set of measures, and procedures as the organization's needs change. Examples of when the common set of measures may need to be revised include the following:
    - New processes are added
    - Processes are revised and new product or process measures are needed
    - Finer granularity of data is required
    - Greater visibility into the process is required
    - Measures are retired.

Especially the CMMI level four involves a metrics-based management of all parts and elements of software product, processes and resources.

During software process measurement the results are stored in different kinds of metrics databases and metrics repositories [Braungarten 2005]. Special kinds of metrics exploration lead to experience bases known as experience factories. The following figure shows some layers about metrics data bases (MDB).

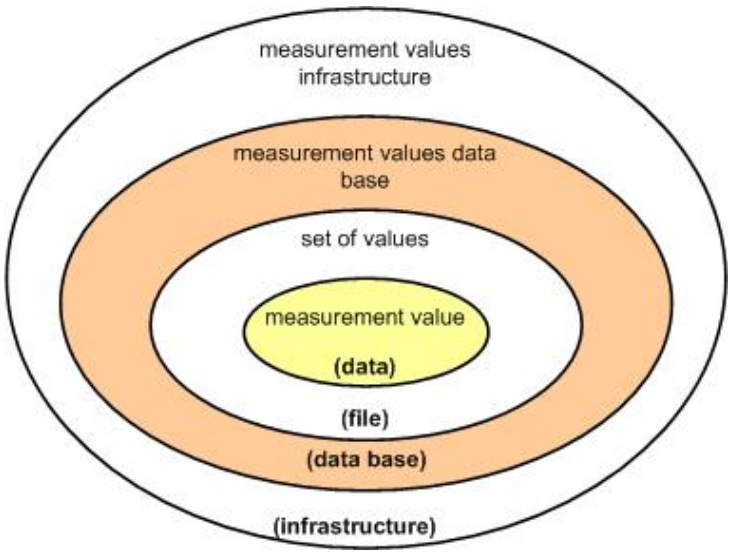


Figure 37: Layers of metrics data bases

MDB's are built from any kind of measurement and evaluation. A special kind of *process-related MDB*, the ***International Software Benchmarking Standards Group (ISBSG),*** maintains a repository of data from numerous organizations' completed software projects ([Hill 1999], [Lokan 2001]). The ISBSG database includes the following parameters of a project [Braungarten 2005].

Project Data Parameters

Project ID
(A primary key, for identifying projects.)

Count Approach
(A description of the technique used to count the function points; e.g. IFPUG, MKII, NESMA, COSMIC-FFP etc.)

Function Points
(The adjusted function point count number. Adjusted by the Value Adjustment Factor.)

Function Size Metric Used
(The functional size metric used to record the size of the project, e.g.. IFPUG3, IFPUG4, in-house etc.)

Value Adjustment Factor
(The adjustment to the function points, applied by the project submitter, that takes into account various technical and quality characteristics e.g.: data communications, end user efficiency etc. This data is not reported for some projects, (i.e. it equals 1).)

Counting Technique
(The technology used to support the counting process. Certain technologies used in function point counting can impact on the count's potential accuracy.)

Development Platform
(Defines the primary development platform, (as determined by the operating system used). Each project is classified as either, a PC, Mid Range or Mainframe.)

Summary Work Effort
(Provides the total effort in hours recorded against the project by the development organization. The three methods provided for are A, B and C.)

Resource Level
(Data is collected about the people whose time is included in the work effort data reported. Four levels (1 to 4) are identified in the data collection instrument.)

Data Quality Rating
(This field contains an ISBSG rating code of A, B, C or D applied to the project data by the ISBSG quality reviewers.)

Max Team Size
(The maximum number of people that worked at any time on the project, (peak team size).)

Development Type
(This field describes whether the development was a new development, enhancement or re-development.)

Reference Table Approach
(This describes the approach used to handle counting of tables of code or reference data, (a comment field).)

Architecture
(Defines the architecture type of the project. e.g.: Client/Server, LAN, WAN etc.)

Language Type
Defines the language type used for the project: e.g. 3GL, 4GL, Application Generator etc.

Primary Programming Language
The primary language used for the development: JAVA, C++, PL/1, Natural, Cobol etc.

DBMS Used
(Whether the project used a DBMS.)

Upper CASE Used
(Whether project used upper CASE tool.)

Lower CASE Used (with code generator)
(Whether project used lower CASE tool with code generator.)

Integrated CASE Used
(Whether project used integrated CASE tool.)

Used Methodology
(States whether a methodology was used.)

Project Elapsed Time
(Total elapsed time for project in months.)

Development Techniques
(Techniques used during development. (e.g.: JAD, Data Modeling, OO Analysis etc.).)

How Methodology Acquired
(Describes whether the methodology was purchased or developed in-house.)

Project Inactive Time
(This is the number of months in which no activity occurred, (e.g. awaiting client sign off, awaiting acceptance test data). This time, subtracted from Project Elapsed Time, derives the elapsed time spent working on the project.)

Implementation Date
(Actual date of implementation. (Note: the date is shown in the data in date format 1/mm/yy).)

Defects Delivered
(Defects reported in the first month of system use. Three columns in the data covering the number of Extreme, Major and Minor defects reported.)

User Base – Business Units
(Number of business units that the system services, (or project business stakeholders).)

User Base – Locations
(Number of physical locations being serviced/supported by the installed system.)

User Base – Concurrent Users
(Number of users using the system concurrently.)

Organization Type
(This identifies the type of organization that submitted the project. (e.g.: Banking, Manufacturing, and Retail).)

Business Area Type
(This identifies the type of business area being addressed by the project where this is different to the organization type. (e.g.: Manufacturing, Personnel, and Finance).)

Application Type
(This identifies the type of application being addressed by the project. (e.g.: information system, transaction/production system, process control.))

Package Customization
(This indicates whether the project was a package customization. (Yes or No).)

Degree of Customization
(If the project was based on an existing package, this field provides comments on how much customization was involved.)

Project Scope
(This data indicates what tasks were included in the project work effort data recorded. These are: Planning, Specify, Design, Build, Test, and Implement.)

Work Effort Breakdown
(*When provided in the submission, these fields contain the breakdown of the work effort reported by five categories: Plan, Specify, Build, Test and Implement.****)***

Ratio of Project Work Effort to Non-Project Activity
(*The ratio of Project Work Effort to Non-Project Activities.****)***

Percentage of Uncollected Work Effort
(The percentage of Work Effort not reflected in the reported data. i.e. an estimate of the work effort time not collected by the method used.)

Function Point Categories
(When provided in the submission, the following five fields which breakdown the Function Count are provided: external Inputs, external Outputs, external Enquiries, internal logical files, and external interface files.)

Enhancement Data
(When provided in the submission, for enhancement projects the three fields Additions, Changes, and Deletions, which breakdown the Function Point Count are provided.)

Total Defects Delivered
(Defects reported in the first month of system use. This column shows the total of Extreme, Major and Minor defects reported. Where no breakdown is available, the single value is shown here.)

Source Lines of Code (SLOC)
(A count of the SLOC produced by the project.)

Unadjusted Function Points
(The unadjusted function point count (before any adjustment by a Value Adjustment Factor if used).)

Normalized Work Effort
(For projects covering less than a full development life-cycle, this value is an estimate of the full development life-cycle effort. For projects covering the full development life-cycle, and projects where development life-cycle coverage is not known, this value is the same as Summary Work Effort.)

Work Effort Unphased
(Where no phase breakdown is provided in the submission, this field contains the same value as the Summary Work Effort. Where phase breakdown is provided in the submission, and the sum of that breakdown does not equal the Summary Work Effort, the difference is shown here.)

Unadjusted Function Point Rating
(This field contains an ISBSG rating code of A, B, C or D applied to the unadjusted function point count data by the ISBSG quality reviewers.)

Productivity Rates Parameters	
Project ID *(The primary key, for identifying projects.)*	**Normalized Productivity Delivery Rate** *(Project productivity delivery rate in hours per function point calculated from Normalized Work Effort divided by Unadjusted Function Point count. Use of normalized effort and unadjusted count should render more comparable rates.)*
Project Productivity Rate *(Project productivity delivery rate in hours per function point calculated from Summary Work Effort divided by Unadjusted Function Point count.)*	**Normalized Productivity Delivery Rate (adjusted)** *(Project productivity delivery rate in hours per function point calculated from Normalized Work Effort divided by Adjusted Function Point count.)*
Reported Productivity Delivery Rate (adjusted) *(Project productivity delivery rate in hours per function point calculated from Summary Work Effort divided by Adjusted Function Point count.)*	

Table 9: Attributes of the ISBSG Benchmarking Data CD Release 8

The following diagram shows the distribution of projects (stored in the ISBSG repository 2003) considering provided defect data [Emam 2005].

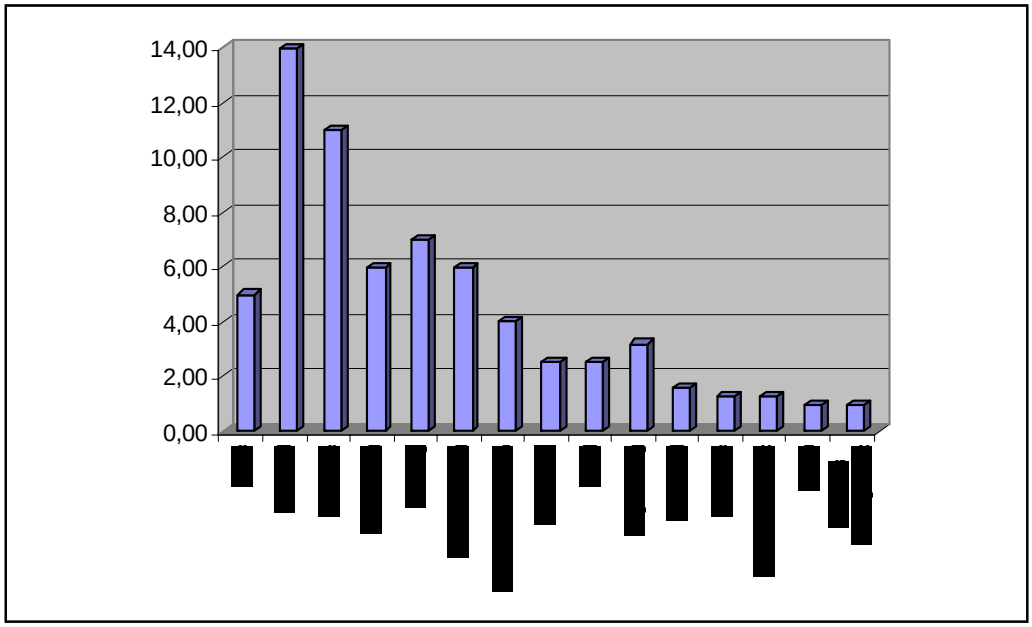


Figure 38: Distribution by business domain of ISBSG projects that provided defect data in percentage

Currently, it is possible to use the ISBSG data repository in the Web showing the following component of the *Functional Size e-Measurement Portal* (FSeMP) [Lother 2004].

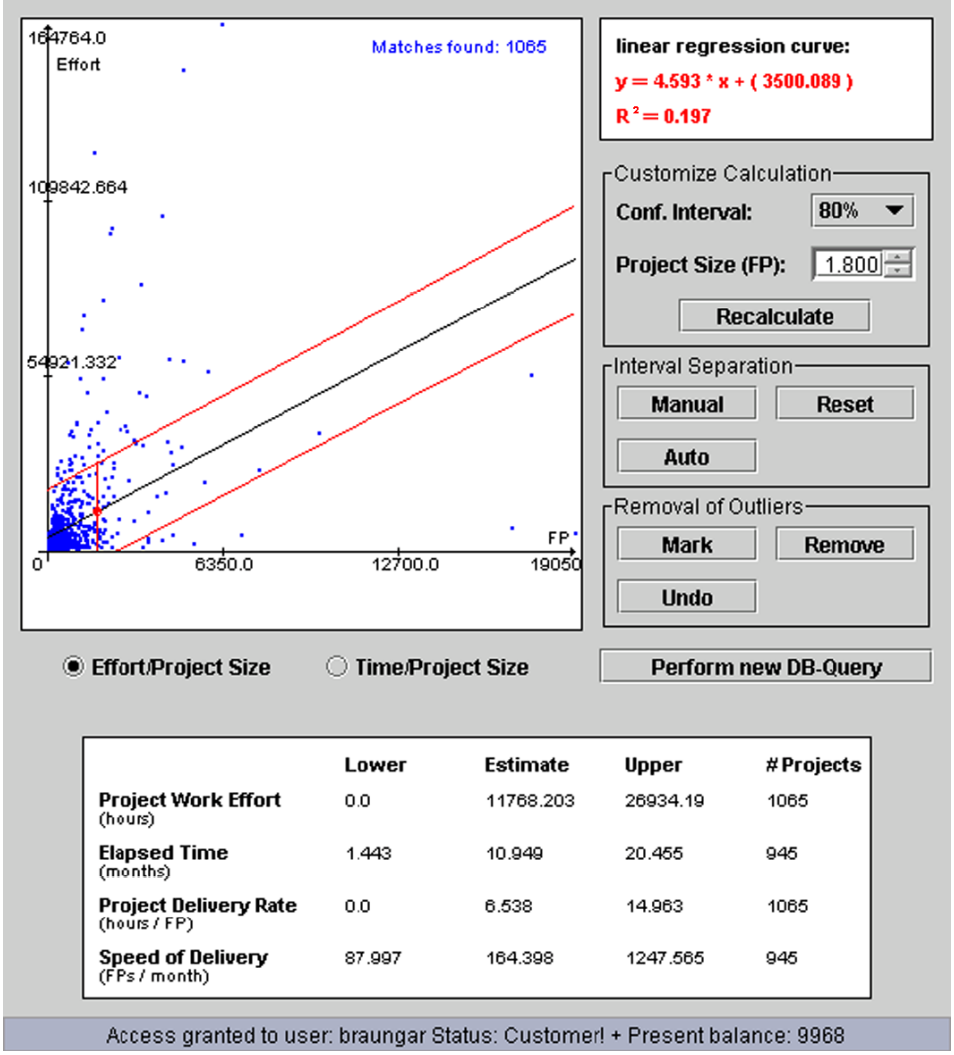


Figure 39: The ISBSG repository using in the Web

Aiming at the development of higher quality software systems at lower costs complying with the *Quality Improvement Paradigm* (QIP), this challenge leads to the development of so called Experience Factories incorporating repositories, which Basili defines as following (see [Braungarten 2005]):

> *"The Experience Factory is the organization that supports reuse of experience and collective learning by developing, updating and delivering upon request to the project organizations clusters of competencies [..] as experience packages."*

> *"The Experience Factory is a logical and/or physical organization that supports project developments by analyzing and synthesizing all kinds of experience, acting as a repository for such experience, and supplying that experience to various projects on demand."*

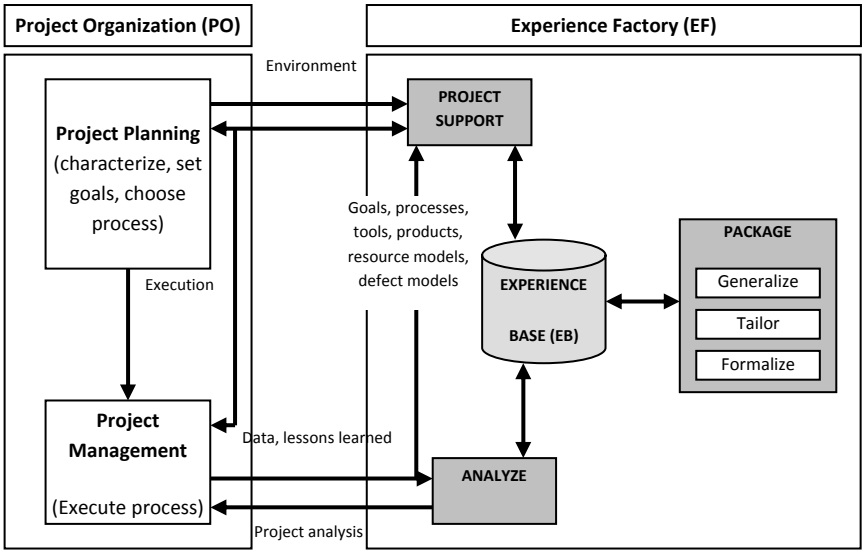


Figure 40: The concept of Basili's Experience Factory

Finally, we will characterize very shortly three approaches of measurement repositories described in [Braungarten 2006] and [Wille 2006].

- ***Measurement Data Warehouse***

    - Data Integration Approach: Data Consolidation
    - Data Integration Technology: ETL
    - Storage: Analytical Transactional-processing databases, DW

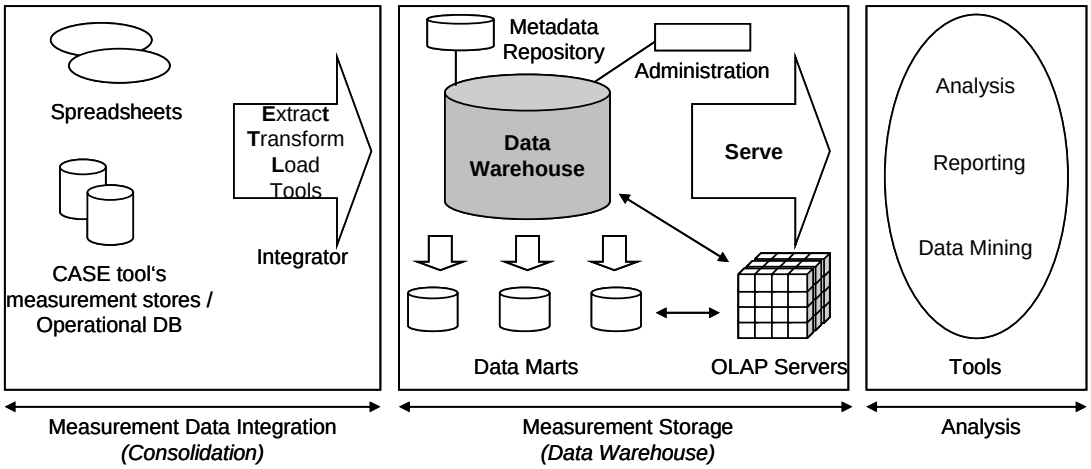


Figure 41: The measurement data warehouse approach

- ***Mediated Measurement Repository:***

    - Data Integration Approach: Data Federation
    - Data Integration Technology: EII
    - Storage: Mediated schema provides access to measurement data sources (type does not matter)

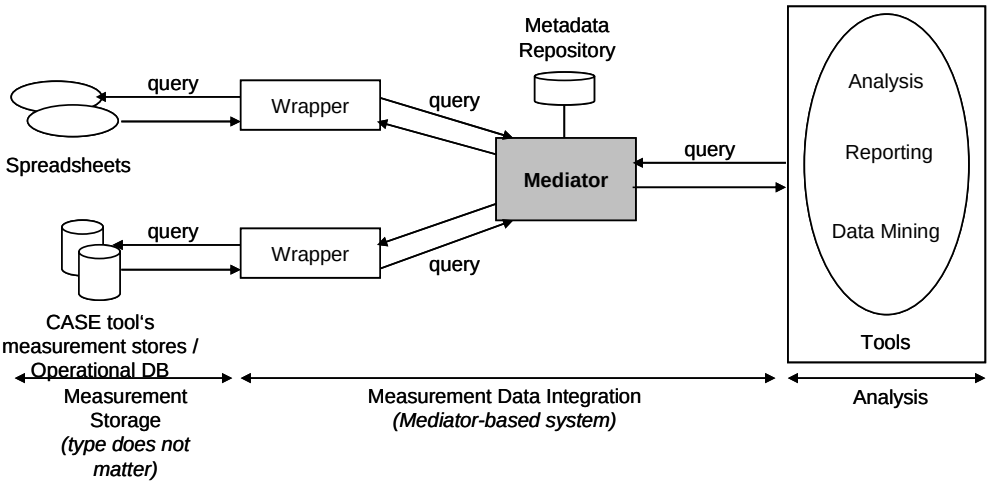


Figure 42: The mediated measurement repository

- ***Service Bus-oriented Measurement Repository:***

  - Data Integration Approach: Data Propagation
  - Data Integration Technology: EAI
  - Storage: propagation from measurement application via service bus to storage or analysis service

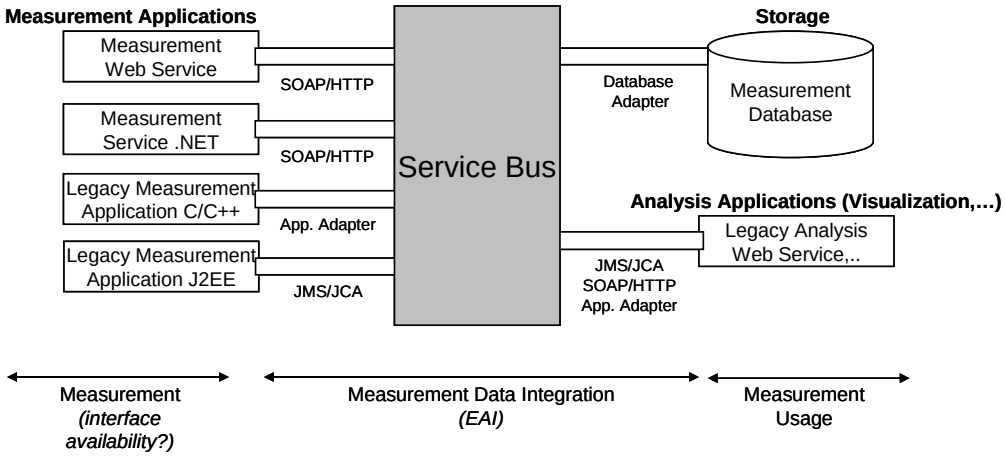


Figure 43: The service bus-oriented measurement repository

4.9 Process Measurement Levels

We will start with a definition of software process modelling using a description from Wang and King [Wang 2000] as given below:

> *"**Software process model (SPM)** is a process of a model system that describes process organisation, categorization, hierarchy, interrelationships, and tailorability."*

A complete presentation of the process based aspects which we must consider in the management of software processes is shown in following chartbased on the process modelling figures in chapter two.

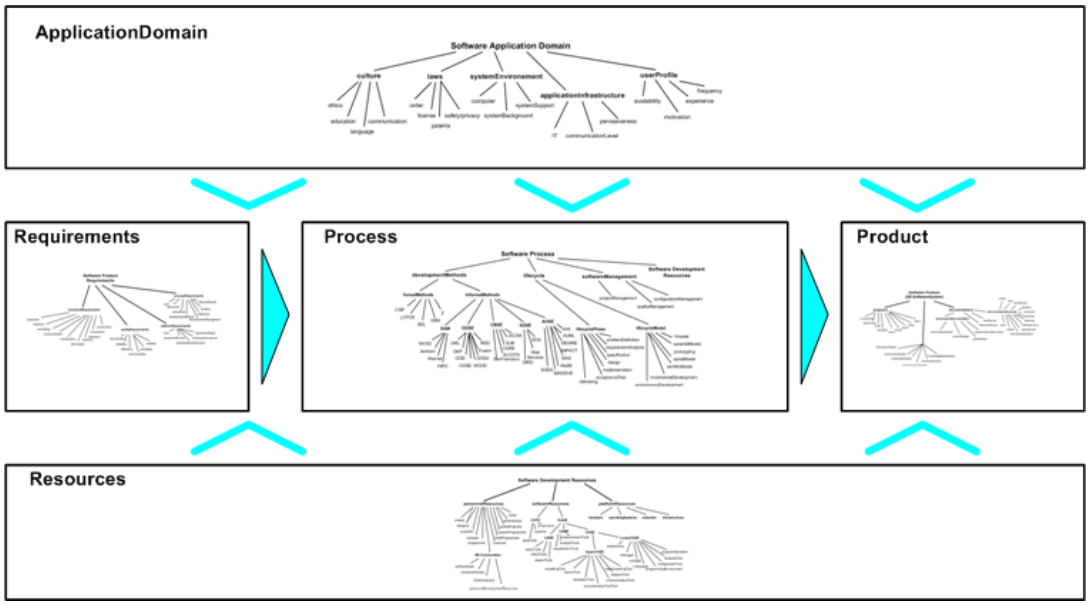


Figure 44: A holistic presentation of the software process involvements

Based on this description we will investigate different relationships of process aspects in order to qualify the process management.

Our approach will consider the software process involvements in an explicit (shown) manner by using different detailed semantic networks which should guarantee the following:

- *In every level of measurement-based process management we can see the missing parts or the parts of weak kinds of evaluation*
- *The semantic relationships between the process related aspects should be explained definitely and could be analyzed through their appropriateness in a changing IT area.*

We will point to the fact again that the following discussion is based on our technical report in [Dumke 2008] that includes process metrics descriptions and levels.

4.9.1 Software Process Establishment by Indicators and Criteria

At first, the following definition given in chapter two is helpful as a kind of improvement of our SPM.

> *"The **software process establishment (SPE)** is a systematic procedure to select and implement a process system by model tailoring, extension, and/or adaptation techniques."*

The general software process model (SPM) is qualifying to a software process establishment (SPE) as a concrete IT adaptation. We use this consideration in order to improve our SPM with more details. The next figure includes a refinement of the general structure described in the figure above and the derived process indicators and criteria from [Emam 2005], [Gadatsch2005], [Haywood 198], [Kandt 2006], [Lecky-Thompson 2005], [Lepaaar 2001], [Putnam 2003], and [Zettel 2001].

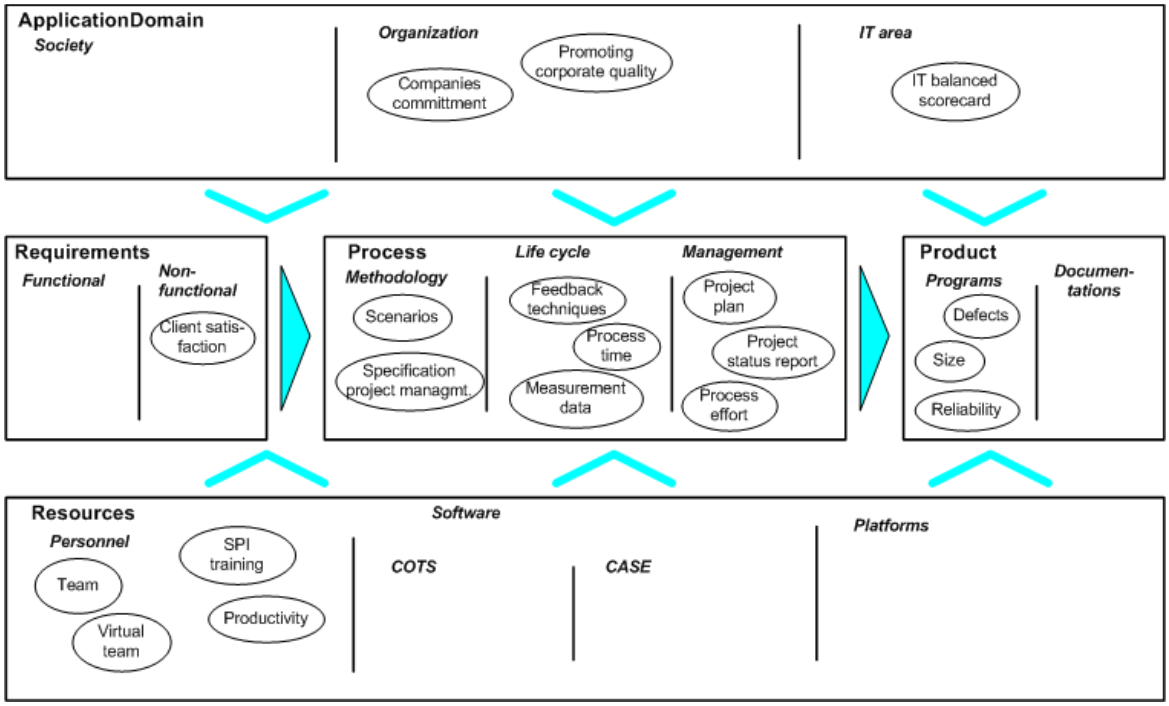


Figure 45: The SPE based process network

Note that our SPE refinement of the SPM based on the scientific investigation leads us to the nodes of our semantic network approach only. But we could establish the following situation:

- *The aspects of process evaluation do not involve the area of software (as COTS or CASE), platforms and society aspects like cultural background and marketplace.*

- *Simple process management criteria are not addressed to the product documentations as is the typical situation in the agile development approaches.*

4.9.2 Software Process Improvement Modelling by Laws, Process Principles and Rules

Considering the improvement context of software processes, the following definition also given in chapter two is helpful again:

> *"A **process improvement model (PIM)** is an operational model that provides guidance for improving a process system's capability by changing, updating, or enhancing existing processes based on the findings provided in a process assessment."*

We will choose some laws exemplary from [Endres 2003] and process principles and rules from [Kandt 2006], [Keyes 2003], [Messerschmitt 2003], [Verzuh 005], [Walter 2006], and [Wong 2001] and derive the following simple kind of a process related semantic network.

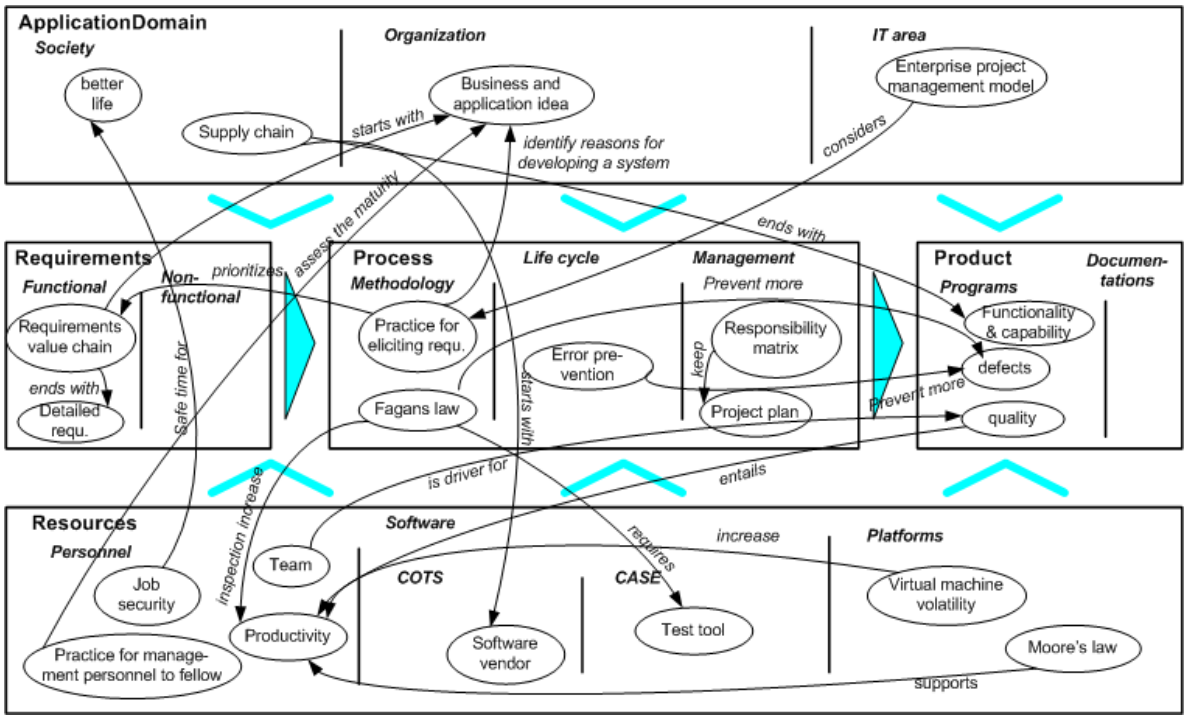


Figure 46: The PIM based process related semantic network

Analyzing this simple kind of process related semantic network, we can establish the following:

- *Based on the considered approaches in the literature, we can see a better coverage of the involved software process related areas*
- *The derived semantic network is helpful for identifying quality improvements principles and laws for process controlling*
- *Considering the roots of this semantic network gives a good orientation for the essential process aspects that would lead to best process improvement and management.*

4.9.3 Empirical Software Process Modelling by Rules of Thumb, Process Experiments and Case Studies

In order to qualify our process semantic network by quantitative experience, we consider an empirical (based) process model. Therefore, the following definition given in chapter two is helpful again:

> *"An **empirical process model (EPM)** is a model that defines an organized and benchmarked software process system and best practices captured and elicited from the software industry."*

As rules of thumb, experiments and case studies will chose some typical kinds from [Basili 2001], [Dumke 2005], [Emam 2005], and [Sneed 2005] which include a lot of such experience. The following figure shows the involvement of such empirical studies in our kind of semantic network.

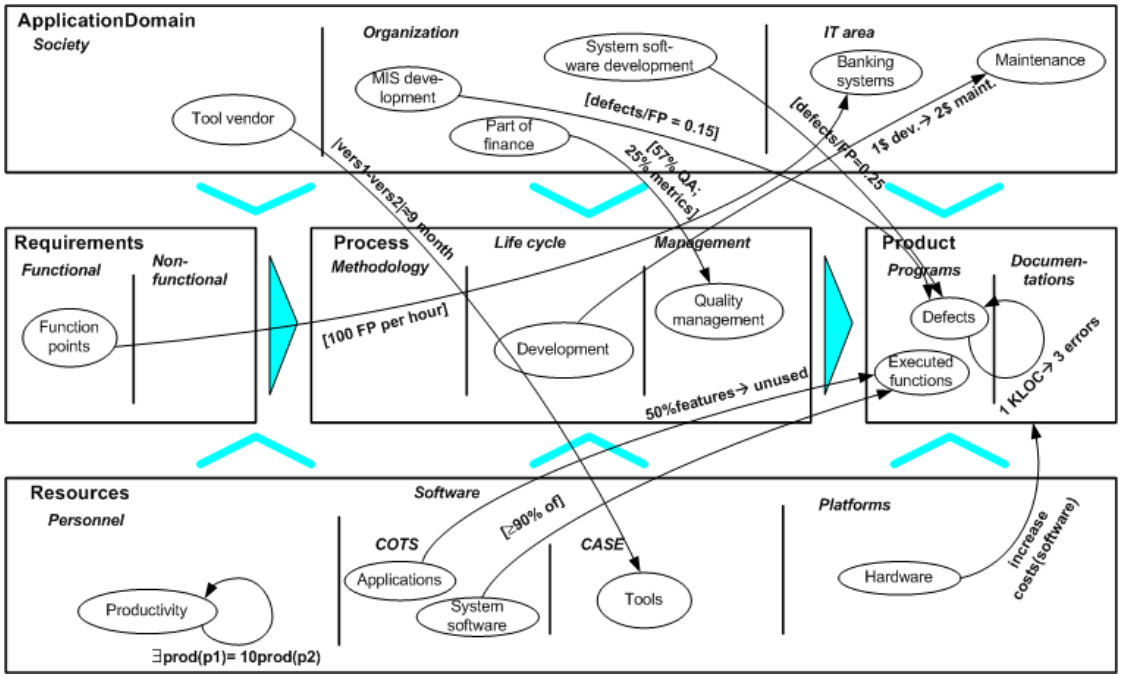


Figure 47: An EMP based process related semantic network

Of course, our network presentation is very simple but suggests some of the following short characteristics:

- *The relationships between the network components are quantified and can support the process management in this manner*

- *Using the kinds of semantic network applied before, we can identify some tasks for improving the process relationships for higher software process control.*

Note that we have chosen some examples of rules of thumb, experiments and case studies in order to avoid confusion in the figure above. You can find further examples in our process measurement technical report in [Dumke 2008].

4.9.4 Software Process Measurement Model by Process Metrics

Using metrics leads us to the highest level of process related semantic network because of quantified relationships between different process involvements. We will start with a simple definition again addressing to the process measurement as following:

> *"**Software process measurement model (SPM)** is a model that defines an organized and benchmarked software process system and applies process metrics and measures with quantitative measurement characteristics."*

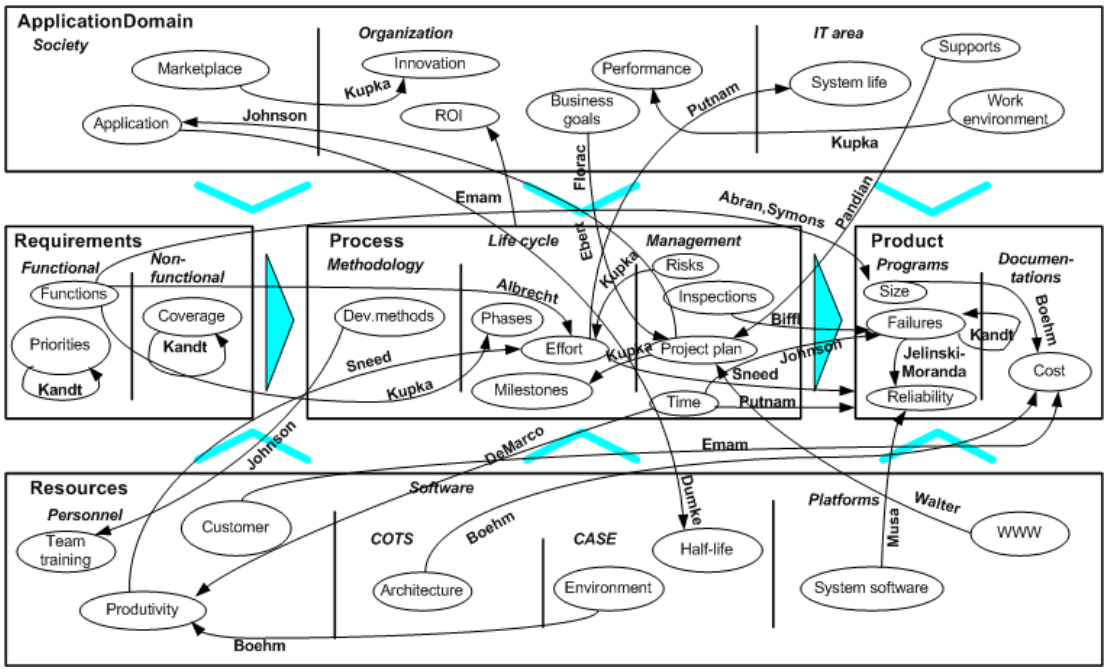


Figure 48: A SMP based process related semantic network

In order to derive a semantic network based on these characteristics we will consider the experiences in [Dumke 2004], [Dumke 2006], [Ebert 2004], [Kandt 2006], [Keyes 2003], [Kulpa 2003], [Pandian 2004], [Putnam 2003], [Sneed 2005], and [Walter 2006]. We will choose only some of these process metrics in the figure above in order to better see the principal structuring of process related involvements.

This kind of semantic process related network helps us in the following manner:

- *The semantic network shows the process components which are involved in the quantitative process management*

- *It is possible to find out the components which are necessary to measure in order to apply the chosen process metrics*

- *Finally it is possible to estimate the level of quantitative management as a fundamental basis of process management level.*

4.9.5 Software Process Management Models by Process Improvement Approaches

We will start with the following definition from Wang and King [Wang 2000] addressing the management area:

> *"The **software management processes (SMP)** are processes that belong to a supporting process subsystem, which control the development processes by means of resources, staff, schedule, and quality."*

Involving improvement/maturity level (evaluation) approaches like *CMMI, SPICE, ITIL* or *Six Sigma* (described in principle in [Dumke 2006]), we can analyze the process metrics structuring in the following manner:

- $SMP^{(SPE)} \rightarrow \text{CMMI}_{levelTwo}^{Best\ Practices}$

- $SMP^{(PIM)} \rightarrow \text{CMMI}_{levelThree}^{ITIL}$

- $SMP^{(EPM)} \rightarrow \text{CMMI}_{levelThree}^{SixSigma}$

- $SMP^{(SPM)} \rightarrow \text{CMMI}_{levelFour}^{SPM}$

This means that the improvement or evaluation approaches cover special areas of our general process involvement structure.

Hence, our derived figures of semantic process networks show different kinds of management levels shown in the following figure.

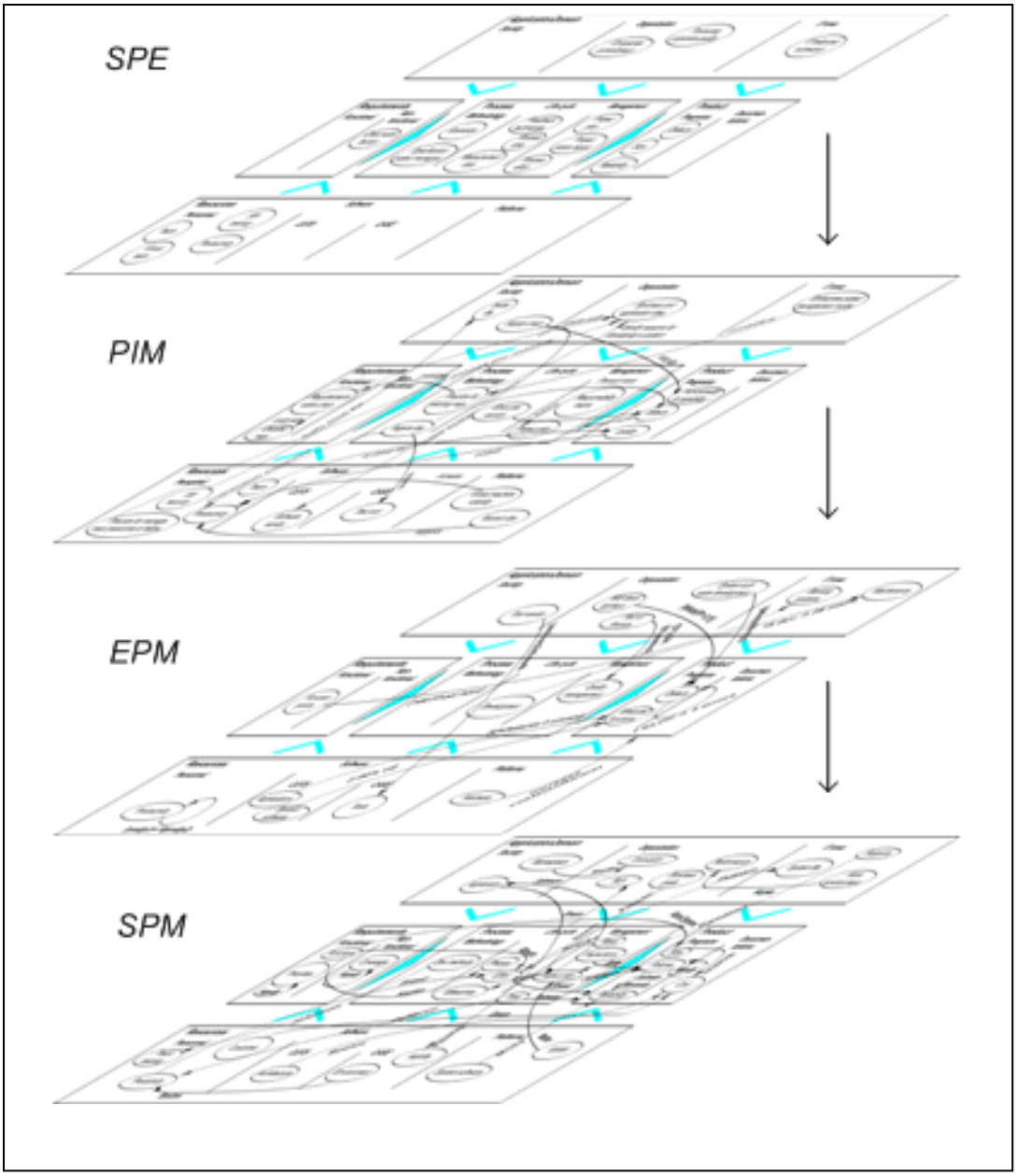


Figure 49: The different process semantic network levels

This visualization suggests that process management should consider all levels of process evaluation identified as *software process establishment (SPE)*, principles, laws and rules by *process improvement models (PIM)*, experimentation or rules of thumb by *empirical process models (EPM)* and process metrics by a *software process measurement model (SPM)*) in order to cover the process areas as much as possible.

On the other hand, our investigations and considerations have also clarified the current situation of process measurement using an appropriate method of structuring of (all) the software process involvements.

4.10 Summary

The fourth chapter has described the different existing experiences in software process measurement and evaluation. We have considered the different levels of evaluation as criteria and indicators (as nomination), as experiments and case studies (in an ordinal manner) and metrics or measures for the quantification of software development processes. For our thesis approach we can summarize that

- CASE tool evaluation could be based on software process indicators and criteria such as ***project management*** (invoice generation, reporting, payment tracking, order processing, account maintenance, customer management, stock management, and tax return) and ***promoting corporate quality*** (projecting quality (communicating quality, documentation, rewarding quality), managing quality (quality reviews, quality checklists, total quality management, quality circles), document quality (process description documents, benchmark reporting, badges))
- The efficiency of CASE tools could be improved by adaptation of ***essential process principles*** such as the use of highest-level programming language possible, the use of integrated development environments, adopting a coding standard that prevents common types of defects, and prototyping user interfaces and high-risk components
- CASE tools are ***essential quality drivers*** and should be used in all phases and activities of development processes
- CASE tools should be integrated in the software development management processes based on ***balanced scorecards, dashboards***, SDE's productivity models and ***process metrics***
- In CASE tools should be considered the current ***product metrics for quality assurance*** for the given development paradigm as system aspects
- CASE tools should implement and update a ***measurement repository*** in order to keep the quality improvement in future projects and developments
- CASE tools should consider external experiences such as ***experience factories*** or the ***ISBSG initiative***

A framework using the different kinds of software process analysis, evaluation and measurement for a CASE tool based software development is described in the next chapter.

5 Framework of Quality Assurance Using CASE-Tools

5.1 Framework Principles: CASE Tool Based Software Processes

5.1.1 General Principles

The general intention of the framework is to evaluate and improve the CASE tool based *measurement processes* itself. In chapter two we have summarized that,

- Software processes can be defined in different levels using experiences, considering development paradigms and determine process activities, categories and (sub) systems
- The main area of software processes is the development on software systems/products using software resources such as personnel, COTS, hardware and (basically) CASE tools
- Different approaches of software process descriptions such as CMMI or ITIL help to improve the process but define the quality assurance implicitly and in a verbal manner mostly

Using the process evaluation definitions of Wang and King [Wang 2000] we prefer a declarative evaluation and improvement model described in section 5.2 in details. The general principles can be characterized by the following definitions addressing the CASE tool based orientation of [Wang 2000]:

> *A **generic model of the CASE tool based software development organization** is a high-level process model of an organization which is designed to regulate the functionality and interactions between the roles of developers, managers, and customers by a CASE tool based software engineering process system.*
>
> *A **CASE tool based process reference model** is an established, validated, and proven software engineering process model that consists of a comprehensive set of software processes and reflects the CASE tool based benchmarked best practices in the software industry.*
>
> *A **CASE tool based process capability model (CPCM)** is a measurement scale of software process capability for quantitatively evaluating the existence, adequacy, effectiveness, and compatibility of a CASE-based process.*
>
> *A **CASE tool based process improvement model (CPIM)** is an declarative model that provides guidance for improving a CASE-based process system's capability by changing, updating, or enhancing existing processes based on the findings provided in a process assessment.*

These definitions are based on the detailed process descriptions in chapter two and the adapted definitions of *CASE-based software process assessment (CSPA)* and the *CASE-based software process improvement (CSPI)* explained in the chapter three.

5.1.2 CASE-Based Orientation

The special kind of evaluation and improvement of CASE-based processes is an essential circumscription of our framework. In chapter three we have summarized:

- CASE tools can be found in all software engineering dimensions such as technologies (OOSE, CBSE etc.) kinds of systems (embedded, information, knowledge-based etc.) and development process aspects (lifecycle, management etc.)
- An effective use of CASE tool requires their consistent integration over the different process activities and phases
- CASE tools can be used as monolithic tools, tool sequences, tool compositions and as (Web) services respectively
- CASE tool could be integrated in the software development process as CASE-based software process assessment and CASE-based software process improvement
- Evaluation and improvement standards and approaches define the functional background of CASE tools and their meaningful and effective manner as CASE tool based support and determination

That means that we intend to consider the CASE tool applications in the software processes as we have described in details in chapter three as

> ***CASE tool based support*** *for the examples of quality and process performance attributes for which needs and priorities might be identified, of process performance attributes for which objectives might be written, of sources for objectives, of sources for criteria used in selecting sub processes, of sources of the risks, of criteria for determining whether data are comparable, of techniques for analyzing the reasons for special causes of variation, of actions that can be taken when a selected sub process' performance does not satisfy its objectives, of activities for stakeholder involvement, of activities reviewed*
>
> ***CASE tool based determination*** *of the examples of quality attributes for which objectives might be written, of product and process attributes, of sub process measures, examples of where the natural bounds are calculated, of other resources provided, of training topics, of work products placed under configuration management, of measures used in monitoring and controlling*

Otherwise, we want to keep the different dimensions of CASE tool variations and applications as we have characterized in chapter three for the

different kinds of software engineering methodologies as

$$\{CASE_{developmentMethods}, CASE_{lifecycle}, CASE_{softwareManagement}\}$$

different kinds of software engineering technologies as

$$\{CASE_{SAM}, CASE_{OOSE}, CASE_{CBSE}, CASE_{AOSE}\}$$

different kinds of software engineering systems as

$$\{CASE_{informationSystem}, CASE_{constructionSystem}, CASE_{embeddedSystem}, CASE_{communicationSystem}, CASE_{distributedSystem}, CASE_{knowledgeBasedSystem}\}$$

5.1.3 CASE Tool Based Process Evaluation

The existing process evaluations can be and should be used in order to analyze the situation of CASE-based software development and suggest any improvements. In chapter four we have summarized

- CASE tool evaluation could be based on software process indicators and criteria such as ***project management*** (invoice generation, reporting, payment tracking, order processing, account maintenance, customer management, stock management, and tax return) and ***promoting corporate quality*** (projecting quality (communicating quality, documentation, rewarding quality), managing quality (quality reviews, quality checklists, total quality management, quality circles), document quality (process description documents, benchmark reporting, badges))
- The efficiency of CASE tool could be improved by adaptation of ***essential process principles*** such as the use the highest-level programming language possible, the use integrated development environments, adopting a coding standard that prevents common types of defects, and prototyping user interfaces and high-risk components
- CASE tools are ***essential quality drivers*** and should be used in all phases and activities of development processes
- CASE tool should be integrated in the software development management processes based on ***balanced scorecards, dashboards***, SDE's productivity models and ***process metrics***
- In CASE tools should be considered the current ***product metrics for quality assurance*** for the given development paradigm an system aspects
- CASE tools should implement and update a ***measurement repository*** in order to keep the quality improvement in future projects and developments
- CASE tools should consider external experiences such as ***experience factories*** or the ***ISBSG initiative***

This complex background intends the possibility of adaptation of the following process experiences in principle

> ***Software process indicators and criteria***: CASE-based processes should involve the management and lifecycle indicators for the (corporate) quality assurance
>
> ***Software process laws***, ***principles and rules***: the CASE tool evaluation should be based on the software engineering methodology principles and frameworks
>
> ***Software process rules of thumb***: CASE-based processes must be founded on industrial experiences of real system applications
>
> ***Software process experiments and case studies***: it must be consider the appropriate system and method related CASE tool variations in practice
>
> ***Software process metrics and measures***: measurement should be involved in CASE tool directly or integrated by measurement supports
>
> ***Software measurement repositories***: CASE-based processes must be implement an experience data base that can be connected with (international) experience data

5.2 Framework Kernel: Quality Measurement and Improvement

5.2.1 Software Measurement Components

Considering the measurement systems aspects we define a *software measurement system* in a declarative manner as following ([Ebert 2007], [Skyttner 2005]):

$$MS = (M_{MS}, R_{MS}) = (\{G, A, M, Q, V, U, E, T, P\}, R_{MS}) \quad (5.1)$$

where G is the set of the measurement goals, A the set of measured artefacts or measurement objects, M the set of measurement methods, objects or entities, Q the set of measurement quantities, V the set of measurement values (especially we could have the situation $Q = V$), U the set of measurement units, E the set of measurement-based experience, T the set of measurement CASE tools (respectively CAME tools), and P the set of the measurement personnel. R_{MS} defines all meaningful relations between the elements of M_{MS}. Especially, the *measurement process MP* as one of the instantiations of a software measurement system could be explained by the following sequence of relations ([Dumke 2008], [Yazbek 2010b])

$$MP: \quad (G \times A \times M)_{T,P} \rightarrow (Q \times E)_{T,P} \rightarrow (V \times U)_{T,P} \rightarrow E' \times A' \quad (5.2)$$

This measurement process description explains the process results as quantities including some thresholds, values involving their units and/or extended experiences combined with improved or controlled measurement artefacts. E' is the set of *extended* experience and A' the set of *improved* artefacts or measurement objects. Special variants of this general measurement description are

- the simple metrication $MP: (G \times A \times M)_{T,P} \rightarrow (Q \times E)$
- the software measurement $MP:(G \times A \times M)_{T,P} \rightarrow (V \times U)$
- the different kinds of measurement and improvement as

  $MP: \quad (G \times A \times M)_{T,P} \rightarrow (Q \times E)_{T,P} \rightarrow E' \times A'$

  $MP: \quad (G \times A \times M)_{T,P} \rightarrow (V \times U)_{T,P} \rightarrow E' \times A'$

Based on our software measurement experiences we can derive the following refinement[4] on the process description above. Furthermore, we give a first graduation of the described software measurement characteristics. The idea of classification of measurement aspects and processes is not new. Examples are

1. Zelkowitz defines a ranking of validation of research papers as a 14-scale taxonomy in decreasing manner as: *project monitoring, case study, field study, literature search, legacy data, lessons learned, static analysis, replicated experiment, synthetic, dynamic analysis, simulation, theoretical, assertion, no experimentation* [Zelkowitz 2007].
2. A consideration of the experiment levels by Kitchenham leads to (also decreasing): *industrial case studies, quasi experiment*, and *formal experiment* [Kitchenham 2007].

[4] This refinement does not fulfil the principle of completeness.

We will use these experiences and some of the results from our industrial projects at Alcatel, Siemens, Bosch and German Telekom (see in [Abran 2006], [Ebert 2007] and [Farooq 2008]) in order to achieve a *holistic approach*. Furthermore, we use the general kind of

$$\begin{matrix} process \\ operation \end{matrix} \; Component \; \begin{matrix} paradigm \\ situation \end{matrix} \tag{5.3}$$

as measurement process component description. The ordinal classifications of the measurement process components in an increasing manner are described in the next section.

MEASUREMENT INGREDIENTS

The tuple of ($G \times A \times M$) as measurement goals, artefacts and methods describes the input and basis for any software measurement. The detailed characteristics of these three sets are:

Measurement Goals *G*:
Evidence: In order to describe the evidence of the measurement goal we define based on the general viewpoint of evidence levels (see the measurement graduation in the ISO/IEC product quality standard in [ISO 9126]) such as *internal goals/quality, external goals/quality* and *goals/quality in use:*

evidence $\in$ {*internal_goals, external_goals, goals_in_use*}

Viewpoint: On the other hand the goals depend on the special viewpoint such as *development phase, implemented product* and *the use of the product in the marketplace* (adapting the different levels of software products by Bundschuh and Dekkers in [Bundschuh 2008])*:*

viewpoint $\in$ {*development, product, in_marketplace*}

Intention: We will consider in our approach the goal intentions as *understanding, evaluation, improving* and *managing.* This enumeration corresponds to an increasing level of measurement goals (see the classification by Basili in [Basili 1986]):

intention $\in$ {*understanding, evaluation, improving, managing*}

Purpose: Finally, the goals depend on the special purposes such as *characterization, evaluation* and *motivation* (using the distinctions of motivation by Basili in [Basili 1986])*:*

purpose $\in$ {*characterization, evaluation, motivation*}

These descriptions lead to the general characterization of the ***measurement goal*** as

$$\begin{matrix} evidence \\ viewpoint \end{matrix} \; G \; \begin{matrix} intention \\ purpose \end{matrix} \tag{5.4}$$

Measurement Artefacts *A:*
Domain: The considered measurement artefacts should be the general classification (based on the Fenton/Pfleeger classification in [Fenton 1997]) of software as *products (systems), processes (*e. g. *project)* and *resources* (including their different parts or aspects (e. g. *product model, process phases* or *personal resources*)):

$$domain \in \{(product\_aspects \vee process\_aspects \vee resources\_aspects), (product \vee process \vee resources), (product \wedge process \wedge resources)\}$$

Dependence: The measurement artefact could be aggregated with other in the cases of *integrated, associated,* and *monolithic* (using the Laird consideration of measurement difficulties in [Laird 2006])*:*

$$dependence \in \{ integrated, associated, monolithic \}$$

State: The state of the measurement artefact as measurement ingredient means that the artefact is only identified for the measurement process*:*

$$state \in \{referred\}$$

Origin: Note that we could consider a pendant or analogical artefact of measurement that led us to the kinds of measurement as *analogical conclusion*. Analogy can be defined as *tuning* (where we use a pendant in the same class of software systems) and as *adaptation* (where we use another pendant of artefact) (see the Pandian graduation in [Pandian 2004]):

$$origin \in \{ other\_pendant, pendant\_in\_same\_domain, original \}$$

Therefore, the ***measurement artefacts*** could be described as

$$\begin{matrix} domain \\ dependence \end{matrix} A \begin{matrix} state \\ origin \end{matrix} \tag{5.5}$$

Measurement Methods *M:*

Usage: The usage of the measurement method depends on the IT process environment and considers aspects such as *outsourced* or based on methodology of *global production* or *in-house* (adapting the classification by Dumke et al. in [Dumke 2010])*:*

$$usage \in \{ outsourced, global\_production, inhouse \}$$

Method: The chosen measurement methods should be classified here as *experiment/case study, assessment, improvement* and *controlling.* That means that measurement should contain the partial phases as *referencing, modelling, measurement, analysis, evaluation* and *application* and could cover different parts of these phases. Note that the dominant use of experiences could lead to the kinds of measurement as *estimation* or *simulation* (considering the Munson classification in [Munson 2003])*:*

$$method \in \{experiment/case\ study, assessment, improvement, controlling\}$$

Application: On the other hand the measurement application could be embedded in closed IT processes and can be differed in *closed component, remote application* and *Web service* (see the measurement infrastructure principles of Dumke et al. in [Dumke 2010])*:*

$$application \in \{closed\_component, remote\_application, Web\ service\}$$

Sort: Furthermore, depending on the measured artefact(s) that is involved in the measurement, we will distinguish between *analogical conclusion, estimation, simulation* and *measurement* (using the measurement overview by McConnel in [McConnel 2006]):

$$sort \in \{analogical\_conclusion, estimation, simulation, measurement\}$$

These characteristics lead to the following description of the ***measurement methods*** as

$$\begin{matrix} usage \\ method \end{matrix} M \begin{matrix} application \\ sort \end{matrix} \tag{5.6}$$

MEASUREMENT OUTPUT

The immediate output of software measurement consists of numbers that would be interpreted by using any experience described by the pair as ($Q \times E$). The typical properties of these sets are:

Measurement Quantities *Q*:

Evaluation: The kind of evaluation of the measurement output as quantities includes aspects such as *threshold, (min, max) criteria, gradient* and *formula* (see the criteria classification by Pandian in [Pandian 2004]) and is based on the measurement experience *E* that is explained below:

$evaluation \in \{threshold,\ min\_max\_criteria,\ gradient,\ formula\}$

Exploration: On the measurement experience *E* again the output could be analyzed/ explained by principles such as *intuition, rules of thumb, trend analysis* and *calculus* (considering the measurement exploration by Abran in [Abran 2006] and Endres in [Endres 2003])*:*

$exploration \in \{ intuition,\ rules\_of\_thumb,\ trend\_analysis,\ calculus \}$

Value: This set of metrics values/numbers characterises a *qualitative measurement* and are given in a *nominal scale* or *ordinal scale* (see the metrics scale classification by Whitmire in [Whitmire 1997])*:*

$value \in \{identifier/nomination,\ ordinal\_scale\}$

Structure: Measured values could be structured in different kinds of presentations and transformations such as s*ingle value, normalization* and *aggregation* (adapting the measurement evaluation by Juristo in [Juristo 2003] and Pfleeger in [Pfleeger 1998])*:*

$structure \in \{single\_value,\ (normalization \vee\ transformation),\ aggregation\}$

These aspects are summarized in the following description of the ***measurement output as quantities***

$$\begin{matrix} evaluation \\ exploration \end{matrix} Q \begin{matrix} value \\ structure \end{matrix} \tag{5.7}$$

MEASUREMENT RESULTS

As a higher level of measurement output we want to achieve real measures including their units. Characteristics of the sets in the tuple ($V \times U$) as values and their units are:

Measurement Values *V:*

Measure: This set of metrics values characterises a *quantitative measurement* and is given an *interval scale* or *ratio scale* (considering the metrics scale analysis by Zuse in [Zuse 1998])*:*

$$measure \in \{interval\_scale, ratio\_scale\}$$

Aggregation: The values could be built as different structures and aggregations such as *measurement repositories, simple visualizations* (e. g. diagrams scatter plots)*, dashboards* and *cockpits* (see the measurement process description in the ISO 15939 standard in [ISO 15939])*:*

$$aggregation \in \{values, (data\_basis \vee repository), (dashboard \vee cockpit)\}$$

Unit: The measurement unit *U* could be *CFP* (COSMIC FFP functional size), *program length* of Halstead, *kilo delivered lines of code* (KDSI)*, cyclomatic complexity* of McCabe etc. (using the measurement unit mentioned by Ebert in [Ebert 2007]):

$$unit \in \{sociological\_unit, economical\_unit, physical\_unit, hardware\_unit, software\_unit\}$$

Interpretation: Furthermore, the measurement values could be interpreted based on experiences such as *analogical project, IT project data base* and *(international) ISBSG project data base* (adapting the benchmark concept of the International Software Benchmark Standard Group (ISBSG) in [ISBSG 2003])*:*

$$interpretation \in \{analogical\_project, project\_data\_base, ISBSG\_data\_base\}$$

These characteristics lead to the following description of the ***measurement results as values*** as

$$\overset{measure}{aggregation}\, V \,\overset{unit}{interpretation} \tag{5.8}$$

MEASUREMENT RESOURCES

Every phase of the software measurement process is supported by tools used by personnel. The detailed characteristics of these sets are:

Measurement Tools *T:*

Level: The level of the measurement tool and the tool support should be classified as *manual* (without any tools), *semi-automatic* and *automatic* (using the support classification by Pfleeger in [Pfleeger 1998])*:*

$$level \in \{manual, semi\text{-}automatic, automatic\}$$

Support: On the other hand the tool could be applied in the IT area (as *internal measurement*) or by vendors (as *external measurement*) (see the IT situation described by Bundschuh in [Bundschuh 2008]):

support ∈ { *external_measurement, internal_measurement* }

Context: Furthermore, the measurement tool could be applied as *simple task application, embedded in a measurement task sequence* or as an *integrated part of the measurement process* (adapting the Munson graduation in [Munson 2003]):

context∈ {*simple_task, task_sequence, intergrated_task*}

Degree: This characteristic determines the availability of the tool application as a *simple tool, decision-supported tool* and *experience-based measurement and evaluation tool* (see the measurement levels described by Dumke et al. in [Dumke 2008]):

degree ∈ {*simple_tool, decision_based, experience-based* }

These descriptions lead to the general characterization of the ***measurement tool*** as

$$\begin{matrix} level \\ support \end{matrix} T \begin{matrix} context \\ degree \end{matrix} \tag{5.9}$$

Measurement Personnel *P:*

Kind: The measurement personnel involve different kinds of measurement and intentions and could be distinguished as *measurement researchers, practitioners* and *managers* (see the different IT roles by Pfleeger in [Pfleeger 1998])*:*

kind ∈ {*manager, researcher, practitioner*}

Area: Furthermore, the measurement personnel could be divided in *origin measurement staff* (measurement analyst, certifier, librarian, metrics creator, user and validator) and in *IT staff* who use the software measurement indirectly (administrator, analyst, auditor, designer, developer, programmer, reviewer, tester, maintainer, customer and user) (considering the Pandian classification in [Pandian 2004]):

area∈ { *measurement_application_staff, measurement_expert_staff* }

Qualification: As an essential aspect the qualification of the measurement personnel can be distinguished as *beginners, certified user* and *experienced user* (using the experience classification by Ebert in [Ebert 2007]):

qualification∈ { *beginners, certified_user, experienced_user* }

Coaching: This aspect considers the motivation and intention of the measurement personnel and can be distinguished as *engaged user, extern motivated user* and *self motivated user* (adapting the different roles by Dumke et al. in [Dumke 2008a]):

coaching∈ { *engaged_user, extern_motivated_user, self_motivated_user* }

Therefore, the ***measurement personnel*** could be described as

$$\begin{matrix} kind \\ area \end{matrix} P \begin{matrix} qualification \\ coaching \end{matrix} \tag{5.10}$$

MEASUREMENT REPERCUSSIONS

Finally, the software measurement could/should lead to extensions of the experience and to improvements of the measures artefacts explained in the tuple ($E' \times A'$). Typical properties are:

Measurement Experience *E':*

Form: The appropriate experiences for *Q* and *V* are given as *analogies, axioms, correlations, intuitions, laws, trends, lemmas, formulas, principles, conjectures, hypothesises* and *rules of thumb* (see the different kind of experience by Davis in [Davis 1995])*:*

$$form \in \{(intuition \vee law \vee trend \vee principle), analogy, (criteria \vee rules\_of\_thumb), (axiom \vee lemma \vee formula)\}$$

Contents: The contents or kinds of experience could be *thresholds, lower and upper limits, gradients, calculus* and *proofs* (considering the causal-based levels of experience by Dumke et al. in [Dumke 2008])*:*

$$contents \in \{(limits \vee threshold), (gradient \vee calculus), proof\}$$

Source: Furthermore, the experience could be derived from different sources such as *case study, project-based* and *long years practice* (adapting the Kitchenham classification in [Kitchenham 2007]):

$$source \in \{ case\ study, project\text{-}based, long\ years\ practice \}$$

Extension: Especially the marked set of experiences explains the extended knowledge based on the measurement, evaluation and exploration and can produce *formula correction, principle refinement, criteria approximation* and *axiom extension* (see the Pandian graduation in [Pandian 2004])*:*

$$extension \in \{correction, (refinement \vee approximation \vee adaptation), extension\}$$

These aspects are summarized in the following description of the ***measurement experience***

$$\begin{matrix} form \\ contents \end{matrix} E' \begin{matrix} source \\ extension \end{matrix} \tag{5.11}$$

A': The application of software measurement leads to changed measurement artefacts. Therefore, this description only extends the state characteristic of the measurement artefact as

State: Depending on the measurement process goals and methods, the artefact could be *understood, evaluated, improved, managed* or *controlled* (consider the graduation by Bundschuh in [Bundschuh 2008] and Ebert in [Ebert 2007])*:*

$$state \in \{\text{referred}, understood, improved, managed, controlled\}$$

The measurement process *MP* itself should be characterized by the level of covered/measured artefacts (as *approach*) and by the kind of IT relationship (as *IT process*). Hence, we could define the essential measurement process characteristics in the following formal manner (adapted from [Yazbek 2010b]):

$$\begin{aligned}
&{}^{ITprocess}_{solution}MP^{\,approach}_{\,realisation}: \left({}^{evidence}_{viewpoint}G^{\,intention}_{\,purpose}\right. \\
&\times {}^{domain}_{dependence}A^{\,state}_{\,origin} \times {}^{usage}_{method}M^{\,application}_{\,sort}\left.\right)^{\,level}_{\,support}T^{\,context}_{\,degree},\ {}^{kind}_{area}P^{\,qualification}_{\,coaching} \\
&\rightarrow \left({}^{evaluation}_{exploration}Q^{\,value}_{\,structure}\right)^{\,level}_{\,support}T^{\,context}_{\,degree},\ {}^{kind}_{area}P^{\,qualification}_{\,coaching} \\
&\rightarrow \left({}^{measure}_{aggregation}V^{\,unit}_{\,interpretation}\right)^{\,level}_{\,support}T^{\,context}_{\,degree},\ {}^{kind}_{area}P^{\,qualification}_{\,coaching} \\
&\rightarrow \left({}^{form}_{contents}E^{\,source}_{\,extension} \times {}^{domain}_{dependence}A^{\,state}_{\,origin}\right)
\end{aligned} \tag{5.12}$$

where *E* and *U* are involved in the sets of *Q* and *V*. The classification of the measurement process *MP* itself is based on the measured characteristics. Hence, the *MP* is defined by their involvements and meaningfulness in the IT processes themselves. In a first approximation we consider the IT processes as *quality evaluation, quality improvement* and *quality assurance:*

ITprocess ∈ { *quality evaluation, quality improvement, quality assurance* }

The solution aspect considers the measurement process depending on the kind of application such as *ad hoc usage, scheduled usage* and *ubiquitous usage* as [Dumke 2008]

solution ∈ { ad_hoc, scheduled, ubiquitous }

The measurement of aspects (aspect of a product, process or resources) leads us to the *aspect-oriented measurement*. The measurement of all aspects of a product or all aspects of the process or all aspects of the resources would be called as *capability-oriented measurement*. If we involve all software artefacts (product and process and resources) we will call this as a *whole measurement*. These characteristics build the "approach" attribute of measurement process [Dumke 2008]:

approach ∈ { *aspect-oriented_measurement, capability-oriented_measurement, whole_measurement* }

Otherwise, the "realisation" characteristic defines the measurement process based on the existing *research approaches, wide-used methodologies* and *established standards* as [Ebert 2007]

realisation ∈ {research approach, wide-used methodology, established standard}

5.2.2 Software Measurement Process Evaluation

In the following we will present some examples of this kind of measurement aspects scaling. The different aspects of the measurement process component are defined as a first assumption in an *ordinal manner/scale* (considering also [Dumke 2008], [Farooq 2008] and [Kulpa 2003]). Note that the exponents address the main characteristics and the indexes

show the sub characteristics. This assumption explains some first relationships. We use the symbol "$\preccurlyeq$" in order to explain the difference of main levels (*process* and *paradigm* of the measurement component) and the symbol "$\preccurlyeq$" in order to explain the difference of sub levels (*operation* and *situation* of the measurement component) as characterization of the so-called ***evidence level*** (see [Kitchenham 2007]). The sign "$\precsim$" would be used for any combined ordering between measurement characteristics.

MEASUREMENT LEVELS

Related to the measurement artefacts we can establish

$$\begin{matrix}\textit{product\_aspect}\\ \textit{dependence}\end{matrix} A \begin{matrix}\textit{state}\\ \textit{origin}\end{matrix} \preccurlyeq \begin{matrix}\textit{product}\\ \textit{dependence}\end{matrix} A \begin{matrix}\textit{state}\\ \textit{origin}\end{matrix} \preccurlyeq \begin{matrix}\textit{product} \wedge \textit{process} \wedge \textit{resources}\\ \textit{dependence}\end{matrix} A \begin{matrix}\textit{state}\\ \textit{origin}\end{matrix} \tag{5.13}$$

and

$$\begin{matrix}\textit{domain}\\ \textit{dependence}\end{matrix} A \begin{matrix}\textit{referred}\\ \textit{origin}\end{matrix} \preccurlyeq \begin{matrix}\textit{domain}\\ \textit{dependence}\end{matrix} A \begin{matrix}\textit{understood}\\ \textit{origin}\end{matrix} \preccurlyeq \begin{matrix}\textit{domain}\\ \textit{dependence}\end{matrix} A \begin{matrix}\textit{controlled}\\ \textit{origin}\end{matrix}$$

Otherwise, considering the sub criteria we can constitute

$$\begin{matrix}\textit{domain}\\ \textit{integrated}\end{matrix} A \begin{matrix}\textit{state}\\ \textit{origin}\end{matrix} \preccurlyeq \begin{matrix}\textit{domain}\\ \textit{associated}\end{matrix} A \begin{matrix}\textit{state}\\ \textit{origin}\end{matrix} \preccurlyeq \begin{matrix}\textit{domain}\\ \textit{monolithic}\end{matrix} A \begin{matrix}\textit{state}\\ \textit{origin}\end{matrix}$$

and also

$$\begin{matrix}\textit{domain}\\ \textit{dependence}\end{matrix} A \begin{matrix}\textit{state}\\ \textit{other\_pendant}\end{matrix} \preccurlyeq \begin{matrix}\textit{domain}\\ \textit{dependence}\end{matrix} A \begin{matrix}\textit{state}\\ \textit{same\_domain}\end{matrix} \preccurlyeq \begin{matrix}\textit{domain}\\ \textit{dependence}\end{matrix} A \begin{matrix}\textit{domain}\\ \textit{original}\end{matrix}$$

Comparing both aspects leads to the following consideration

$$\begin{matrix}\textit{product\_aspects}\\ \textit{associated}\end{matrix} A \begin{matrix}\textit{referred}\\ \textit{original}\end{matrix} \precsim \begin{matrix}\textit{product}\\ \textit{associated}\end{matrix} A \begin{matrix}\textit{referred}\\ \textit{original}\end{matrix} \precsim \begin{matrix}\textit{product}\\ \textit{monolithic}\end{matrix} A \begin{matrix}\textit{managed}\\ \textit{original}\end{matrix}$$

Addressing the measurement methods we can establish

$$\begin{matrix}\textit{outsourcd}\\ \textit{case\_study}\end{matrix} M \begin{matrix}\textit{component}\\ \textit{estimation}\end{matrix} \precsim \begin{matrix}\textit{global\_production}\\ \textit{case\_study}\end{matrix} M \begin{matrix}\textit{component}\\ \textit{simulation}\end{matrix} \precsim \begin{matrix}\textit{global\_prod.}\\ \textit{improvement}\end{matrix} M \begin{matrix}\textit{Web\_service}\\ \textit{measurement}\end{matrix}$$

Using our formal approach we can characterize the measurement process in plenty of graduations. This aspect will be used to evaluate the different measurement process levels in the next section.

MEASUREMENT PROCESS EVALUATION

In order to characterize the different software measurement approaches and methodologies we can establish an ordinal scaled multi-dimensional "space" of software measurement aspects that consists of the ***lowest measurement level*** as

$$\begin{matrix}\textit{quality\_evaluation}\\ \textit{ad\_hoc}\end{matrix} MP \begin{matrix}\textit{aspect\_oriented}\\ \textit{research\_approach}\end{matrix} : \tag{5.14}$$

$$\left(\begin{matrix}\textit{internal\_goals}\\ \textit{development}\end{matrix} G \begin{matrix}\textit{understanding}\\ \textit{characterization}\end{matrix} \times \begin{matrix}\textit{aspect}\\ \textit{integrated}\end{matrix} A \begin{matrix}\textit{referred}\\ \textit{other\_pendant}\end{matrix} \right.$$

$$\left. \times \begin{matrix}\textit{outsourced}\\ \textit{experiment}\end{matrix} M \begin{matrix}\textit{component}\\ \textit{analogical\_conclusion}\end{matrix} \right) \begin{matrix}\textit{manual}\\ \textit{external}\end{matrix} T \begin{matrix}\textit{simple\_task}\\ \textit{simple\_tool}\end{matrix}, \begin{matrix}\textit{manager}\\ \textit{meas.\_appl.\_staff}\end{matrix} P \begin{matrix}\textit{beginners}\\ \textit{engaged\_user}\end{matrix}$$

$$\rightarrow \left(\begin{matrix}\textit{threshold}\\ \textit{intuition}\end{matrix} Q \begin{matrix}\textit{nomination}\\ \textit{single\_value}\end{matrix} \right)$$

some ***immediate levels*** or measurement situations such as

$$\begin{matrix}\textit{quality\_improvement}\\ \textit{scheduled}\end{matrix} MP \begin{matrix}\textit{capability\_oriented}\\ \textit{wide\_used\_methodology}\end{matrix}:$$

$$\left(\begin{matrix}\textit{external\_goals}\\ \textit{product}\end{matrix} G \begin{matrix}\textit{improving}\\ \textit{evaluation}\end{matrix} \times \begin{matrix}\textit{process}\\ \textit{associated}\end{matrix} A \begin{matrix}\textit{referred}\\ \textit{other\_pendant}\end{matrix}\right.$$

$$\left.\times \begin{matrix}\textit{global\_production}\\ \textit{estimation}\end{matrix} M \begin{matrix}\textit{remote\_application}\\ \textit{measurement}\end{matrix}\right)^{\begin{matrix}\textit{semi\_automatic}\\ \textit{external}\end{matrix}} T^{\begin{matrix}\textit{task\_sequence}\\ \textit{decision\_based}\end{matrix}, \begin{matrix}\textit{manager}\\ \textit{meas.\_appl.\_staff}\end{matrix}} P^{\begin{matrix}\textit{certified\_user}\\ \textit{motivated\_user}\end{matrix}}$$

$$\rightarrow \left(\begin{matrix}\textit{threshold}\\ \textit{gradient}\end{matrix} Q \begin{matrix}\textit{nomination}\\ \textit{normalization}\end{matrix}\right) \rightarrow \left(\begin{matrix}\textit{formula}\\ \textit{gradient}\end{matrix} E \begin{matrix}\textit{project\_based}\\ \textit{adaptation}\end{matrix} \times \begin{matrix}\textit{process}\\ \textit{integrated}\end{matrix} A \begin{matrix}\textit{improved}\\ \textit{other\_pendant}\end{matrix}\right)$$

and the ***highest software measurement level***

$$\begin{matrix}\textit{quality\_assurance}\\ \textit{ubiquitous}\end{matrix} MP \begin{matrix}\textit{whole\_measurement}\\ \textit{established\_standard}\end{matrix}:$$

$$\left(\begin{matrix}\textit{goals\_in\_use}\\ \textit{in\_marketplace}\end{matrix} G \begin{matrix}\textit{managing}\\ \textit{motivation}\end{matrix} \times \begin{matrix}\textit{product} \wedge \textit{process} \wedge \textit{resources}\\ \textit{monolithic}\end{matrix} A \begin{matrix}\textit{referred}\\ \textit{original}\end{matrix}\right.$$

$$\left.\times \begin{matrix}\textit{inhouse}\\ \textit{controlling}\end{matrix} M \begin{matrix}\textit{Web\_service}\\ \textit{measurement}\end{matrix}\right)^{\begin{matrix}\textit{automatic}\\ \textit{internal}\end{matrix}} T^{\begin{matrix}\textit{integrated\_task}\\ \textit{experience\_based}\end{matrix}, \begin{matrix}\textit{practitioner}\\ \textit{meas.\_expert.\_staff}\end{matrix}} P^{\begin{matrix}\textit{experience\_user}\\ \textit{self\_motivated\_user}\end{matrix}}$$

$$\rightarrow \left(\begin{matrix}\textit{ratio\_scale}\\ \textit{dashboard}\end{matrix} V \begin{matrix}\textit{software\_unit}\\ \textit{ISBSG\_data\_base}\end{matrix}\right)$$

$$\rightarrow \left(\begin{matrix}\textit{formula}\\ \textit{proof}\end{matrix} E \begin{matrix}\textit{long\_years\_practice}\\ \textit{extension}\end{matrix} \times \begin{matrix}\textit{product} \wedge \textit{process} \wedge \textit{resources}\\ \textit{monolithic}\end{matrix} A \begin{matrix}\textit{controlled}\\ \textit{original}\end{matrix}\right)$$

Some first applications consider the different software paradigms and technologies shows first graduations in software measurement [Yazbek 2010b] such as *SOA-based metrication* versus *traditional product quality assurance* or *e-Measurement services* versus *agent-based controlling* (see also [Dumke 2008] for more details).

In this thesis we describe the measurement characteristics based on our extended approach for well-known standards and (process) improvement methodologies. At first we consider the ISO/IEC 9126:2000 product quality standard [ISO 9126]. The (kernel) measurement process is based on many metrics whose scale types are ratio and absolute at mostly. We will not discuss such evaluation here to avoid confusions. Indeed we describe the ISO 9126 based measurement process as following

$$MP(\text{ISO 9126}): \begin{matrix}\textit{quality\_evaluation}\\ \textit{scheduled}\end{matrix} MP \begin{matrix}\textit{product}\\ \textit{ISO9126\_metrics}\end{matrix}$$

The Six Sigma approach is the next considered process improvement methodology [Tayntor 2003]. The basic idea is reduction of errors to a small σ based on the DMAIC (*define, measure, analyze, improve, control*) measurement process. Therefore, the measurement process of Six Sigma can be described as

$$MP(\text{Six Sigma}): \begin{matrix}\textit{quality\_improvement}\\ \textit{scheduled}\end{matrix} MP \begin{matrix}\textit{product\_errors}\\ \textit{DMAIC}\end{matrix}$$

Furthermore, the Goal Question Metric (GQM) paradigm helps for the orientation of software measurement for any (special) goals. The GQM+Strategy approach involves the business (strategy) aspect [Basili 2007] and extends the GQM method by any business improvements. The measurement process could be characterized as

$$MP(\text{GQM+Strategy}): \begin{matrix}\textit{quality\_improvement}\\ \textit{ad\_hoc}\end{matrix} MP \begin{matrix}\textit{business\_aspect}\\ \textit{GQM}\end{matrix}$$

Finally, the Capability Maturity Model Integration (CMMI) is one of the complex methodologies for process improvement including quality assurance aspects based on the higher CMMI level four. In [Kulpa 2003] we can find the essential metrics in the different CMMI levels. Therefore, the CMMI-based measurement process could be described as

$$MP(\text{CMMI}): \quad \begin{matrix} quality\_assurance \\ scheduled \end{matrix} MP \begin{matrix} whole\_process\_components \\ CMMI\_level\_4 \end{matrix}$$

This general characterization gives first information about the general support of quality assurance and their effectiveness by applying these methods and standards. In following we will discuss the improvement aspect using different levels of software measurement.

5.2.3 Software Measurement Improvements

In this section we will differentiate the following *graduation of measurement improvements* as a first kind of improvement classification:

- ***Weak measurement improvement***: This kind of improvement consists of an improvement of a *measurement sub characteristic* to the next level (as one step using "$\preccurlyeq$").
- ***Moderate measurement improvement***: The improvement of the measurement process based on *more than one step of a/some sub characteristic(s)* building this kind of measurement process improvement (using "$\preccurlyeq_n$").
- ***Essential measurement improvement***: This kind of improvement consists of an improvement of a *measurement main characteristic* to the next level (as one step using "$\preccurlyeq$").
- ***Remarkable measurement improvement:*** The improvement of the measurement process based on *more than one step of a/some main characteristic(s)* building this kind of measurement process improvement (using "$\preccurlyeq^n$").

Note, that we define some preferences for the general software measurement process description as

$$\begin{matrix} ITprocess \\ solution \end{matrix} MP \begin{matrix} approach \\ realisation \end{matrix} . \tag{5.15}$$

The criteria of the left side prefers the criteria on the right side. That means that a higher value of the (main/sub) criteria on the left side defines an *improvement* against the lower values of the (main/sub) criteria on the right side. Therefore, based on the formal described measurement process methods of measurement improvement are identified easily. Considering our examples in the section before we can establish

$$MP(\text{ISO 9126}): \quad \begin{matrix} quality\_evaluation \\ scheduled \end{matrix} MP \begin{matrix} capability\_oriented \\ established\_standard \end{matrix}$$

$$MP(\text{Six Sigma}): \quad \begin{matrix} quality\_improvement \\ scheduled \end{matrix} MP \begin{matrix} aspect\_oriented \\ wide\_used\_methodolgy \end{matrix}$$

$$MP(\text{GQM@Strategy}): \quad \begin{matrix} quality\_improvement \\ ad\_hoc \end{matrix} MP \begin{matrix} aspect\_oriented \\ research\_approach \end{matrix}$$

MP(CMMI): $\overset{\text{quality\_assurance}}{\text{scheduled}}\ MP\ \overset{\text{whole\_mesurement}}{\text{wide\_used\_methodology}}$

That leads to the relationships between these measurement processes as

$$MP(\text{ISO 9126}) \lessdot MP(\text{GQM@Strategy}) \lessdot_{n} MP(\text{Six Sigma}) \lessdot^{n} MP(\text{CMMI}) \qquad (5.16)$$

Otherwise, using the methodologies above including their existing measurement approaches we can establish

- changing the paradigm from *MP*(ISO 9126) to *MP*(GQM@Strategy) leads to *essential measurement improvements*
- the further adaptation of the *MP*(Six Sigma) can led to a *moderate measurement improvement*
- finally, changing to the *MP*(CMMI) based measurement could led to a *remarkable measurement improvement*.

5.3 Framework Steps: Phases and Contents

5.3.1 Analyzing the CASE Tool Based Process Situation

The whole purpose of employment of CASE-Tools and software measurement tools should be considered. However it is still difficult to answer some questions like:

- Which measurement tools support the used programming language?
- Is the measurement tool combinable with the used CASE-tool?
- How many metrics and measurement tools are necessary?
- If more than one measurement tool are needed, can be combined the selected measurement tools?
- Another problem is the long-term dependency of tool manufacturers, especially when they disappear from the market.

In order to evaluate the CASE-based situation we establish the following fifteen criteria [Yazbek 2010]:

I. **Metrics coverage:** Within the literature there are a large amount of metrics, including sets of metrics that try to cover a large area of the (object-oriented) quality refinements (Chidamber & Kemerer metrics suite [Chidamber 1994], the MOOD metrics [Harrison 1998] or the collection in [Lorenz 1994]). Unfortunately, the software characteristics that can be easily measured, such as size and cyclomatic complexity, don't have a clear and consistent relationship with quality attributes such as understandability and maintainability. However, individual software metric cannot measure the overall quality characteristic of the product. Moreover, a combination of metrics can be used to make such an analysis more focused and thus more effective (cf. e.g. [Bauer 1999]). Therefore, a CASE tool should provide the maximum possible number of different metrics of the supported programming paradigm.

That means that we must evaluate the covered areas defined in the quality measurement goals in

$$\begin{matrix} evidence \\ viewpoint \end{matrix} G \begin{matrix} intention \\ purpose \end{matrix}$$

II. Metrics providing: It should be absolutely clear what the metrics are measuring. The metrics should provide objective information on the metrics dialog as following:

- ***Name and abbreviation:*** The name should be determined. The abbreviations should reflect the well-known and universally accepted abbreviations.
- ***Description:*** The description explains how the metric is calculated, and how the results can be used.

These aspects are determined by the chosen measurement method described as

$$\begin{matrix} usage \\ method \end{matrix} M \begin{matrix} application \\ sort \end{matrix}$$

III. Metrics suites: For non- or less experienced developers should a default subset of all available metrics be active.

Moreover, the experienced developer will not want to run every metric in the default active set every time, but rather some specific subset of available metrics, it should allow him to create saved sets of active metrics that can be loaded and processed as he choose.

Here we intend the responsability and qualification of the measurement personal described in

$$\begin{matrix} kind \\ area \end{matrix} P \begin{matrix} qualification \\ coaching \end{matrix}$$

IV. Metrics Customizing: For each metric should be there settings for options such as limits and granularity. The default values of limits should be general recommended. The developer can change the settings if necessary.

These characteristics are defined considering the metrics quantities and values respectively as

$$\begin{matrix} evaluation \\ exploration \end{matrix} Q \begin{matrix} value \\ structure \end{matrix} \text{ and } \begin{matrix} measure \\ aggregation \end{matrix} V \begin{matrix} unit \\ interpretation \end{matrix}$$

V. Metrics Extending: It should support the measurement of known metrics like for example the metrics suite (cf. [Chidamber 1994]) but it should also allow the definition of self defined metrics.

This essential characteristic could be considered in the measurement component as

$$\begin{matrix} level \\ support \end{matrix} T \begin{matrix} context \\ degree \end{matrix}$$

VI. **Metrics feedback:** The most interesting result of every measurement is the consequences that can be extracted from the interpretation and the improvements that can be reached through them. Metrics should be able to give tips on how to interpret the results and on what to do to improve them. Metrics results should highlight parts of code that need to be redesigned.

The application of the quality measurement should be usable to the measured artefact themselves as

$$\begin{matrix} domain \\ dependence \end{matrix} A \begin{matrix} state \\ origin \end{matrix}$$

VII. **Metrics filtering:** Depending of the project size the calculation of the metrics might yield tons of numbers. A technique for handling this amount of data is to use

- ***Value filters:*** Like suggested in ISO 9126 it is possible to define several classification limits (cf. [ISO 1926]). These limits can be used to filter from the large amount of measurement values only the interesting ones, e.g. the upper limit one. Another filter would be to concentrate on the x (or x %) largest or smallest measurement values, because they are extreme in some way and should be interpreted at first.
- ***Component filtering:*** Today's software development is based more and more on existing software components (e.g. libraries, frameworks, generators etc.) Most of these reused components are not in control of the engineer because they are fixed. The measurement values might be interesting for the self-written parts or only for some component (e.g. packages, classes, methods etc.)

Especially for measure we must consider the meaningfulness of unit transformations etc. as

$$\begin{matrix} form \\ contents \end{matrix} E \begin{matrix} source \\ extension \end{matrix}$$

VIII. **Sorting results:** When viewing the output of metrics, it should be able to compare and organize the items in the results table. A technique for handling this amount of data is to use

- ***Sorting by one column:*** Ascending and descending sorting all the items according to the values for a specific column
- ***Complex sorting:*** it should be able to prioritize several columns for sorting.

These characteristics are defined considering the metrics quantities and values also as

$$\begin{matrix} evaluation \\ exploration \end{matrix} Q \begin{matrix} value \\ structure \end{matrix} \text{ and } \begin{matrix} measure \\ aggregation \end{matrix} V \begin{matrix} unit \\ interpretation \end{matrix}$$

IX. **Metrics visualization:** As we know from our experiences that a well chosen visualization of metrics values has the largest potential to handle the large amount of numbers and to help us interpreting these numbers. It emphasizes the knowledge that "a picture tells more than thousand words". Visualization as process of

representing data can combine all previous techniques. Metrics results should be viewed as graphical output and metrics should use different kinds of visualization of the measurement data, e.g. Bar chart, Distribution Graph, Kiviat Graph, Histogram, Scatter plot and Pareto chart etc. To make these charts more efficient, it should be colored, e. g. green for acceptable values, red for values exceeding the upper limit, etc.

This further characteristic could be considered in the measurement tool support as

$$\begin{matrix} level \\ support \end{matrix} T \begin{matrix} context \\ degree \end{matrix}$$

X. Saving and loading metrics results: After run a metrics analysis, it should be able to save the results and later view the results table independently of the project. Developer can also use results files to share metrics results with other users.

Especially for measure we must consider the possibility of building repositories based on the experiences as

$$\begin{matrix} form \\ contents \end{matrix} E \begin{matrix} source \\ extension \end{matrix}$$

XI. Comparing metrics results: The metrics should help developers to control his work by comparing projects or by comparing changes in a project over time. Metrics should help developers to compare the “present state” with the “target state”, the “present sate” with the “old state” and the sates from two or more systems. Differences between results can be both highlighted.

In this kind of metrics application should be involved the tool facilities and the measurement users

$$\begin{matrix} level \\ support \end{matrix} T \begin{matrix} context \\ degree \end{matrix} \quad \text{and} \quad \begin{matrix} kind \\ area \end{matrix} P \begin{matrix} qualification \\ coaching \end{matrix}$$

XII. Printing results: Preview and print results either by printing the table or by printing graphs.

This characteristic could be considered in the measurement tool support also as

$$\begin{matrix} level \\ support \end{matrix} T \begin{matrix} context \\ degree \end{matrix}$$

XIII. Exporting results: For metrics results, it should be possible to generate a report in separate file in various formats (text, HTML, csv, etc.).

In the same manner this characteristic could be considered in the measurement tool support as

$$\begin{matrix} level \\ support \end{matrix} T \begin{matrix} context \\ degree \end{matrix}$$

XIV. **Copying metrics results to the clipboard:** Developers should able also to copy results from the Code Metrics Results to the clipboard and paste them to another application, such as a text editor or word processor.

Essential metrics application should be involved the tool facilities and the measurement users as developers as

$$\begin{matrix} level \\ support \end{matrix} T \begin{matrix} context \\ degree \end{matrix} \quad \text{and} \quad \begin{matrix} kind \\ area \end{matrix} P \begin{matrix} qualification \\ coaching \end{matrix}$$

XV. **Metrics verification:** The results of metrics analysis are tightly connected with source code. From any line of the results table, Developers should able to navigate to the appropriate location both in the diagram and in the source code.

$$\begin{matrix} form \\ contents \end{matrix} E \begin{matrix} source \\ extension \end{matrix} \quad \text{and} \quad \begin{matrix} kind \\ area \end{matrix} P \begin{matrix} qualification \\ coaching \end{matrix}$$

Afterwards, we can summarize these evaluation steps in a final presentation as (quality) measurement level such as

$$\begin{matrix} quality\_improvement \\ scheduled \end{matrix} MP \begin{matrix} capability\_oriented \\ wide\_used\_methodology \end{matrix} :$$

$$\left(\begin{matrix} external\_goals \\ product \end{matrix} G \begin{matrix} improving \\ evaluation \end{matrix} \times \begin{matrix} process \\ associated \end{matrix} A \begin{matrix} refetrred \\ other\_pendant \end{matrix} \right.$$

$$\left. \times \begin{matrix} global\_production \\ estimation \end{matrix} M \begin{matrix} remote\_application \\ measurement \end{matrix} \right) \begin{matrix} semi\_automatic \\ external \end{matrix} T \begin{matrix} task\_sequence \\ decision\_based \end{matrix} , \begin{matrix} manager \\ meas.\_appl.\_staff \end{matrix} P \begin{matrix} certified\_user \\ motivated\_user \end{matrix}$$

$$\rightarrow \left(\begin{matrix} threshold \\ gradient \end{matrix} Q \begin{matrix} nomination \\ normalization \end{matrix} \right) \rightarrow \left(\begin{matrix} formula \\ gradient \end{matrix} E \begin{matrix} project\_based \\ adaptation \end{matrix} \times \begin{matrix} process \\ integrated \end{matrix} A \begin{matrix} improved \\ other\_pendant \end{matrix} \right)$$

Therefore, we have built a general characterization of the quality measurement level that help us to improve the situation in a wanted or necessary manner.

5.3.2 Planning the CASE Tool Based Process Improvements

In order to plan the appropriate improvements we use the quality measurement improvement definition from the section 5.2. In the quality assurance of CASE-based software development it means

- ***Weak quality measurement improvement***: This kind of improvement consists of an improvement of a *measurement sub characteristic* to the next level (as one step using "$\preccurlyeq$").
- ***Moderate quality measurement improvement***: The improvement of the measurement process based on *more than one step of a/some sub characteristic(s)* building this kind of measurement process improvement (using "$\preccurlyeq_n$").

- ***Essential quality measurement improvement***: This kind of improvement consists of an improvement of a *measurement main characteristic* to the next level (as one step using "$\prec$").
- ***Remarkable quality measurement improvement:*** The improvement of the measurement process based on *more than one step of a/some main characteristic(s)* building this kind of measurement process improvement (using "$\prec^n$").

These general intentions of improvement can also be used in order to plan special activities for chosen systems, paradigms and/or methodologies in the CASE-based software development area.

5.3.3 Implementing the CASE Tool Based Process Improvements

The implementation depends on the current application field, IT situation, corporate intentions and development environments.

An example of quality measurement implementation in general is a solution using separate tools in which development and measuring tools are integrated (Figure 50).

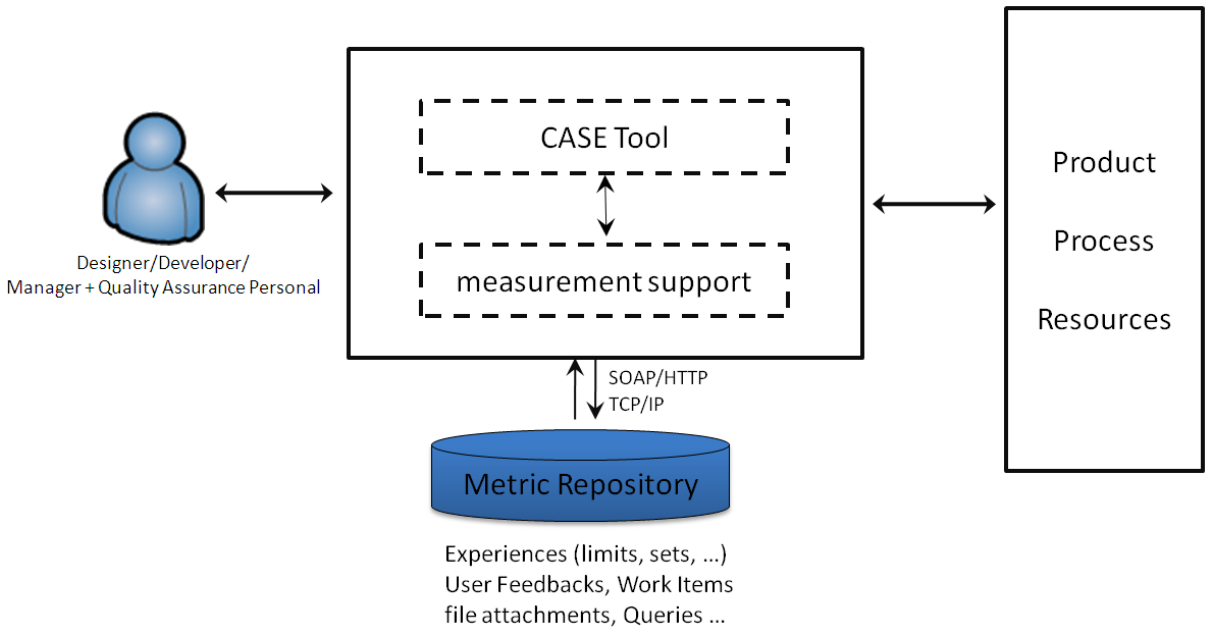


Figure 50: metrics tools integrated into CASE-Tool

The metricDB is to save team experiences (e. g. limits and metrics sets), and to export metrics results to share them with team members or compare them later with other results. Some CASE tool manufacturers thought of this solution and have built some metrics into their tool or provided it as plug-in. However there is no concept how to integrate metrics into the CASE-tools. So the provided solutions are still not really helpful.

5.4 Summary

In this chapter we have defined a framework that can be helpful to evaluate a special situation of the quality measurement level in a CASE-based software development IT area. The framework could be characterized as following.

- The ***framework principles*** are directed on the determination of the quality measurement level explicitly. The basic would be built by an *CASE tool based process improvement model (CPIM)* as a declarative model that provides guidance for improving a CASE-based process system's capability by changing, updating, or enhancing existing processes based on the findings provided in a process assessment. We consider that a CASE-based software development evaluation could be based on software process indicators and criteria such as *project management* and *promoting corporate quality* (managing quality and document quality). In CASE tools would be considered the current *product metrics for quality assurance* for the given development paradigm as system aspects, the implemented and updated *measurement repository* in order to keep the quality improvement in future projects and developments, and considered external experiences such as *external experience factories* or the *ISBSG initiative.*

- The ***framework kernel*** considers the measurement systems aspects as a *software measurement system* in a declarative manner. Hence, we can determine the quality measurement level explicitly in an *ordinal manner*. Therefore, considering quality assurance of CASE-based software development we can classify in a *weak quality measurement improvement*, *moderate quality measurement improvement*, *essential quality measurement improvement* and *remarkable quality measurement improvement.*

- The ***framework steps*** of determination of the CASE-based situation would be considered at the metrics application in CASE tools. Therefore, we established the fifteen criteria such as *metrics coverage, metrics providing, metrics suites, metrics customizing, metrics extending, metrics feedback, metrics filtering, sorting results, metrics visualization, saving and loading metrics results, comparing metrics results, printing results, exporting results, copying metrics results to the clipboard* and *metrics verification.* Afterwards we can improve this situation by a meaningful quality measurement implementation. This implementation depends on the current application field, IT situation, corporate intentions and development environments.

The created framework using different kinds of software process analysis, evaluation and measurement for a CASE tool based software development is described formally. An application of this framework is presented in the next chapter. Furthermore, aspects of framework validation are discussed.

6 Framework Application and Validation

6.1 Chosen CASE Tool Situation

In order to apply our framework, an industrial situation was chosen as a CASE-based software development. That means that we can establish the three CASE characterizations as

$$CASE_{developmentMethods}\text{ , }CASE_{OOSE}\text{ , and }CASE_{informationSystem}. \quad (6.1)$$

Plenty of CASE tools exist but for this work we chose among the CASE tools currently offered on the market which provide some metrics as well. The metrics are either integrated into the tool, or available as a plug-in in a separate component.

The considered CASE tools are the following:

1. The ***Borland Together product family*** is well-known in practice especially for object-oriented system development using current OO languages such C++ and Java [Together 2010]. Variants of this CASE tool are
    - Together 2008 (Release Date 04/07/2009)
    - Together 2006 Release 2 for Eclipse
    - Together 2006 for Visual Studio
    - ControlCenter 6.2

2. The ***Microsoft Visual Studio 2010 product family*** is well-used in the .NET developer community world-wide and support the visual programming mainly [VS 2010]. Variants of this CASE tool are
    - Visual Studio 2010 Premium
    - Visual Studio 2010 Ultimate
3. The ***Enterprise Architect Version 8.0*** was used in the enterprise application integration community (EAI) and prefer the building of infrastructures of business applications [EA 2010].
4. The ***Metrics Eclipse Plug-in Version 1.3.6*** from an IBM initiative for education and training was more and more used in practice [Eclipse 2010].
5. The ***metricsOne Rational Rose Plug-in*** is one of the well-known metrics extensions for the UML-based software development based on the Rational Unified Process (RUP) kind of development [Rational 2004].
6. The ***Embarcadero RAD Studio 2010*** (Delphi and C++Builder) was mainly used in the non IT area for scientific and government applications [RAD 2010].

The special industrial background of our framework application was characterized in the next section.

6.2 CASE-Based Test Scenario

To present information on how the quality assurance metrics available in the inspected CASE-Tools and its results, we have constructed here a project for an airline agency with different programming language (Java, C++, C# and Delphi) based on UML approach. The project is a simple example that nevertheless contains all important phases and structure elements of a real IT project. This prototype was important not only for manipulating and comparing the implemented metrics, but also for inspecting their suggested limits and feedbacks [Yazbek 2007].

As a minimum goal, the tools should first be able to use its metrics and show the results [Yazbek 2010a].

Furthermore, we have considered the software development processes using these CASE tools in three industrial environments in the south of Germany that we don't describe explicitly because of confidential reasons.

6.3 Appraisal of CASE Tool Evaluation Results

In following we describe the quality measurement situation of the different CASE tools considering the available metrics and their appropriateness for quality assurance.

6.3.1 Together Measurement Level

Together [Together 2010] provides object oriented metrics as Quality Assurance features. The metrics quantify the source code. It is up to the developer to examine the results and decide whether they are acceptable. Parts of code that need to be redesigned can be highlighted in the code editor, and metrics results can be used for creating reports and for comparing the overall impact of changes in a project. Together provides tips for using the full set of metrics and interpreting results.

Availability of metric features depends on the project language. Together supports a wide range of metrics for java projects. Other languages have smaller sets of metrics that have been adapted or created to fit the particular language. These metrics are listed as the following:

- 55 metrics available for Java
- 54 for C#
- 52 for Visual Basic .NET
- 52 for C++
- 17 for Visual Basic 6

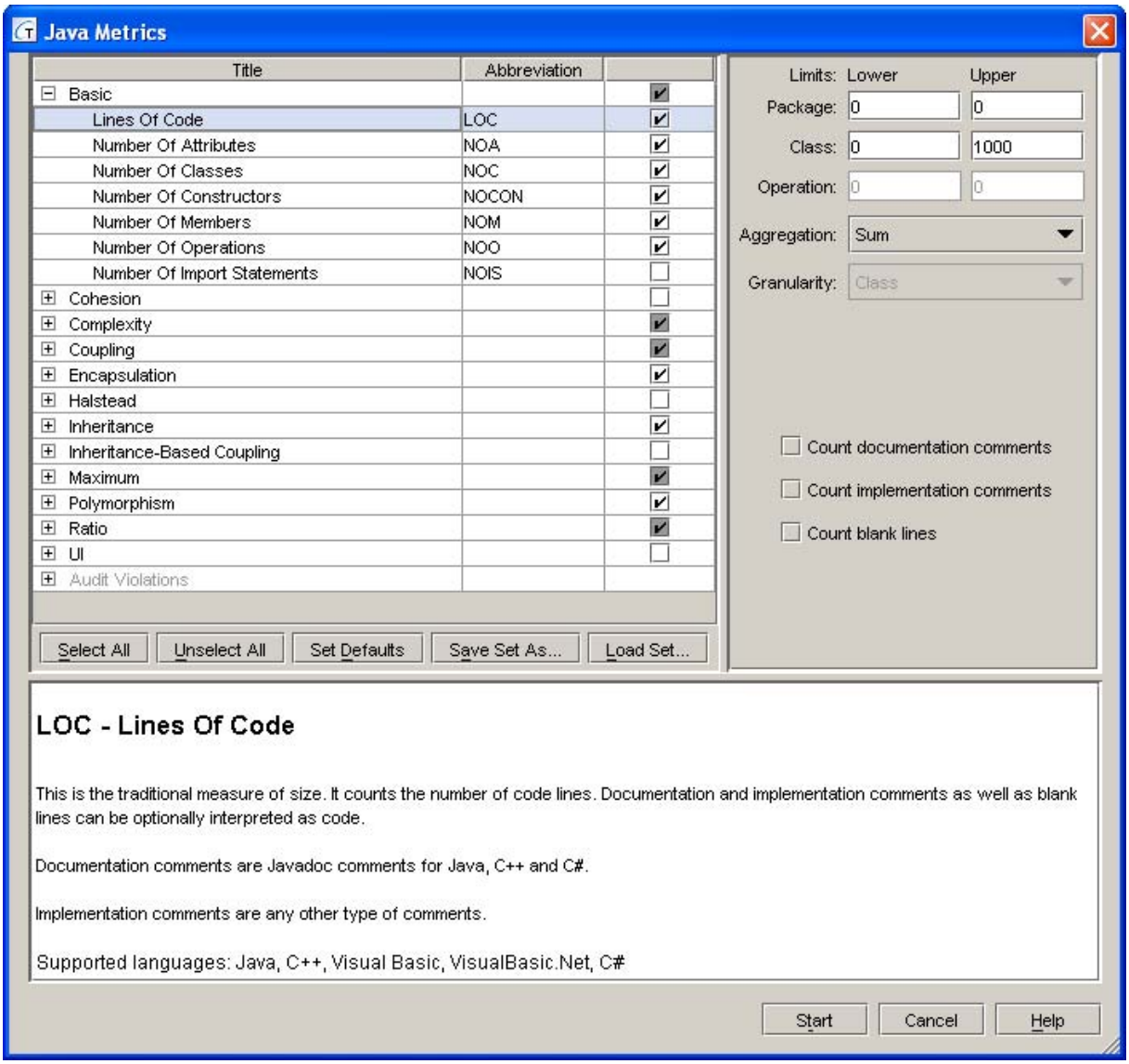


Figure 51: Together metrics dialog

However, if developers have more specific needs, they can create their own custom metric plug-ins to extend the QA module. Together provides settings for options such as limits and granularity for each metric. Developers can change the settings if necessary. Metrics results are displayed as a table: the classes, packages, or diagrams that were analyzed will be showed in the rows, and in the columns will be showed the corresponding values of selected metrics. The results of metrics analysis are tightly connected with source code. The developer can select any element in the table to navigate directly to it in the diagram or source code. The developer can also sort, update and export the result table. For comparing projects or for comparing changes in a project over time the developer can also compare the values in one table against the values in another table. Metrics results can be represented as graphic output (Bar Graph, Distribution Graph, or Kiviat Graph).

Together provides the most international and globally accepted metrics from Chidamber & Kemerer, the MOOD- metrics from Abreu, and the McCabe and Halstead metrics. Among these metrics are two metrics for the documentation (CR und TCR) as well as five metrics for management (CDBC, PPkgM, PPrivM, PProtM, PPubM) [Together 2010].

- **Comment Ratio (CR):** Counts the ratio of implementation and documentation comments to total lines of code (comments are included in the code count).
- **True Comment Ratio (TCR):** Counts the ratio of implementation and documentation comments to total lines of code (all comments are excluded from the code count).
- **Change Dependency Between Classes (CDBC):** measures the class level coupling.

But Together also includes metrics that are not documented in various standards as TRDu, TRDp, TRAu, TRAp. They are only provided in Together. These metrics are documented only in the Marinescu's diploma [Marinescu 1998].

The abbreviations for the metrics in Together do not correspond to known acronyms. The metrics WMC, DIT, NOC and LCOM by Chidamber & Kemerer are described for example in Together as WMPC1, DOIH, NOCC and LOCOM1.

6.3.2 Metrication in Visual Studio

Visual Studio [VS 2010] only provides five basic object oriented metrics to give developers better insight into the code they are developing - Lines of Code, Class Coupling, Depth of Inheritance, Cyclomatic Complexity und Maintainability Index. Visual Studio helps developers to generate code metrics data that measure the complexity and maintainability of their developed code. Code metrics data can be generated for a complete solution or a single project.

Code Metrics Results

Filter: None Min: Max:

Hierarchy	Maintainability Index	Cyclomatic Complexity	Depth of Inheritance	Class Coupling	Lines of Code
TCO(Debug)	72	1.275	12	269	34.452
TCO	72	1.183	12	245	34.168
TCO.Controls	57	82	17	61	1.275
FormDataEntry	57	82	17	61	1.275
AddCategory(Category, bool) : void	94	71		16	51
AddCategory(Category, bool, PropertyValueList) : void	20	40		50	63
AddCategoryList(CategoryList, bool) : void	94	12		16	51
AddCategoryList(CategoryList, bool, PropertyValueList) : void	9	82		16	52
AppendCategoryToPanel(object, AjaxEventArgs) : void	95	41		12	80
AppendTree(int, Tree) : void	94	17		12	50
CategoryExists(Category) : bool	71	13		16	55
CreateID(Category, CategoryMember) : string	81	16		10	92
DocumentId.get() : string	98	12		12	71
DocumentId.set(string) : void	85	21		10	22
FillCategorySelector(Tree) : void	95	11		12	51
Find(CategoryMemberList, PropertyValue) : CategoryMember	18	39		15	25
FormData.get() : PropertyValueList	39	19		17	45
FormDataEntry()	19	18		13	10
FormDataSessionName(DoxObject) : string	84	31		12	32
Page_Load(object, EventArgs) : void	38	10		12	90
PropertyValueList.get() : PropertyValueList	89	12		10	42
TabStyle.get() : bool	98	38		60	81
TabStyle.set(bool) : void	95	12		10	61
Title.set(string) : void	95	19		15	51
TCO.Handler.IHttp	57	10	13	28	34.639

Figure 52: Visual Studio Code Metrics Results Window

The following list shows the code metrics results that Visual Studio calculates. However, Visual Studio does not provide abbreviations for the metrics and info on how to interpret received numbers.

- ***Lines of Code:*** It is the known LOC and it counts the lines of source code of an executable software entity. It excludes Comments, white space, braces and the declarations of members, types and namespaces.
- ***Class Coupling:*** It measures the coupling that the class has on other classes through local variables, parameters, base classes, interface implementations, return types, method calls, generic or template instantiations, attribute decoration and fields that are defined on external types. It excludes primitive and built-in types such as string, int32 and object.
- ***Depth of Inheritance:*** It indicates the number of class definitions that are above the type in the inheritance tree starting from 0 and excludes interfaces.
- ***Cyclomatic Complexity:*** It is the known metric by McCabe. It is calculated by counting the number of different code paths in the flow of the program such as if blocks, switch cases, and do, while, for each and for loops.
- ***Maintainability Index:*** It is an index from 0 to 100 and is based on three other metrics - Halstead Volume, Cyclomatic Complexity and Lines of Code. A low value means worse maintainability as following:
    - Index between 20 and 100 indicates that the code has good maintainability.
    - Index between 10 and 19 indicates that the code is moderately maintainable.
    - Index between 0 and 9 indicates low Maintainability.

After Visual Studio analyzes the code, it displays the results as a simple table in the Code Metrics Results window. A toolbar is at the top of this window, and the columns are for displaying the results of metrics that are calculated. The toolbar at the top is used to filter the results displayed in the Code Metrics Results window. For example, developers might want to see only the results with a Cyclomatic Complexity below 7.

It is worth mentioning that metrics are only valid for source code that can compile. If the source code contains errors, or some libraries and paths are not included, audits and metrics might produce inaccurate results. Developers can also copy results from the Code Metrics Results window to the clipboard and paste them to another application, such as a text editor or word processor. Each copy puts multiple lines of information into the clipboard contains e.g. Project name, Configuration mode and metrics name and its values.

Developers can also export results from the Code Metrics Results window to a Microsoft Excel spreadsheet. Microsoft Excel will be launched if it is installed and the results list will be imported. Developers can then visualize the results in Microsoft Excel. However, this way of presenting results is not meaningful, because the results of metrics are not connected with source code and developers cannot navigate to elements in the source code. Developers also cannot extend this QA module if they have more specific needs and cannot customize it and define limitations for these metrics.

6.3.3 Measurement in Enterprise Architect

Enterprise Architect [EA 2010] is a UML-based CASE tool for developing and building software systems with UML. Enterprise Architect currently supports Round-trip engineering in the following programming languages: Java, C#, C, C++, ActionScript, Delphi, PHP, Python, Visual Basic and Visual Basic.NET.

Enterprise Architect supports techniques for testing, change control and maintenance. Enterprise Architect also provides support for managing projects. Project Managers can use Enterprise Architect to measure risk and effort, estimate the size of a project, and assign resources to elements. Project estimation is working out how much time and effort is required to build and deploy a solution. Enterprise Architect provides the Use Case metrics facility to measure the complexity of a system and getting an indication of the effort required to implement the model, and the project timescale. Project estimation is based on Karner's Use Case Points Method, which is based on this two metrics

- *EWE* (Estimated Work Effort) und
- *EC* (Estimated Cost).

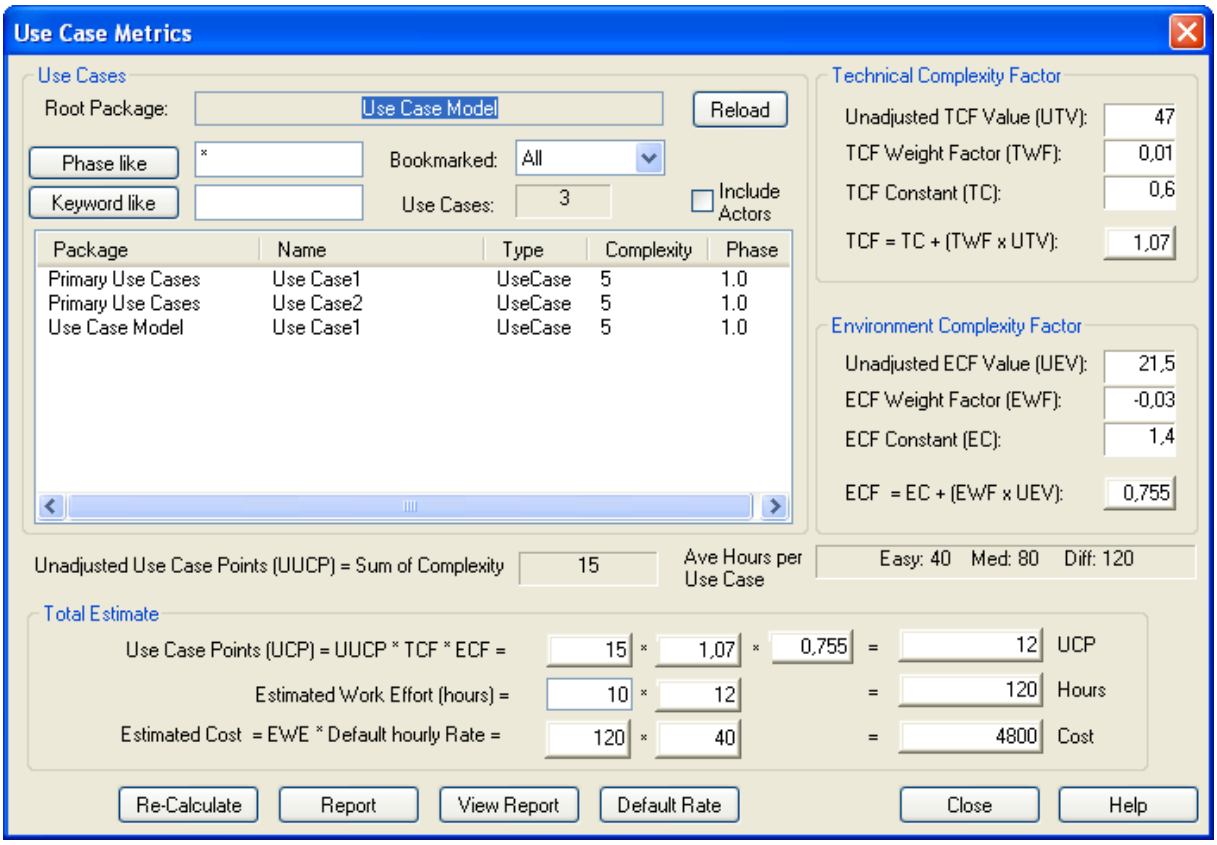


Figure 53: Enterprise Architect Use Case Metrics dialog

However, before estimating project size, the following values must be carefully calibrated in order to gain the best possible estimates:

- ***TCF – Technical Complexity Factors:*** These are editable values that indicate the degree of difficulty and complexity of the work in hand such as portable, easy to install and easy to use
- ***ECF – Environment Complexity Factors:*** These are editable values that indicate the degree of non-technical complexities such as team size, team experience, knowledge and motivation
- ***Default Hour Rate:*** This sets the project defaults for duration and hourly rate per Use Case point.
- ***Use case complexity:*** This value indicates the complexity for each Use Case as the following:
    - Easy (5 points): The use case is considered a simple piece of work. That means it uses a simple user interface and attends only to a single database entity; its success scenario consists of less than 3 steps; its implementation needs not more than 5 classes
    - Medium (10 points): The use case is more difficult, involves more interface design and attends to 2 or more database entities; its success scenario consists of between 4 to 7 steps; its implementation needs between 5 to 10 classes
    - Complex (15 points): The use case is very difficult, involves a complex user interface or processing and attends to 3 or more database entities; its success scenario consists of over seven steps; its implementation needs more than 10 classes.

Modifying these factors requires, in fact, a lot of experience, because the factors and the complexity are depending on the project and current situation. The most difficult factor in an accurate estimation is setting an hourly rate - which is best defined using experience with similar projects. Typical ranges can vary from 10 to 30 hours per Use Case point. The default value for each Use Case point is 10 hours. Once the user has entered all the calibration values, the project timescale will be estimated as following:

$$EWE = Duration * sum\ of\ Complexity * TCF * ECF \quad (6.2)$$

$$EC = EWE * Default\ Hour\ Rate$$

The results can be exported to a RTF file. Enterprise Architect provides no way of graphical representation of results, and the results are in no way comparable with other measurements, so that an improvement in the project is not visible.

6.3.4 Metrics Eclipse Plug-in

Metrics Eclipse Plug-in [Eclipse 2010] provides 23 object-oriented metrics for Java as Lines of Code, Number of Packages, Number of Methods, Chidamber & Kemerer metrics and McCabe Cyclomatic Complexity.

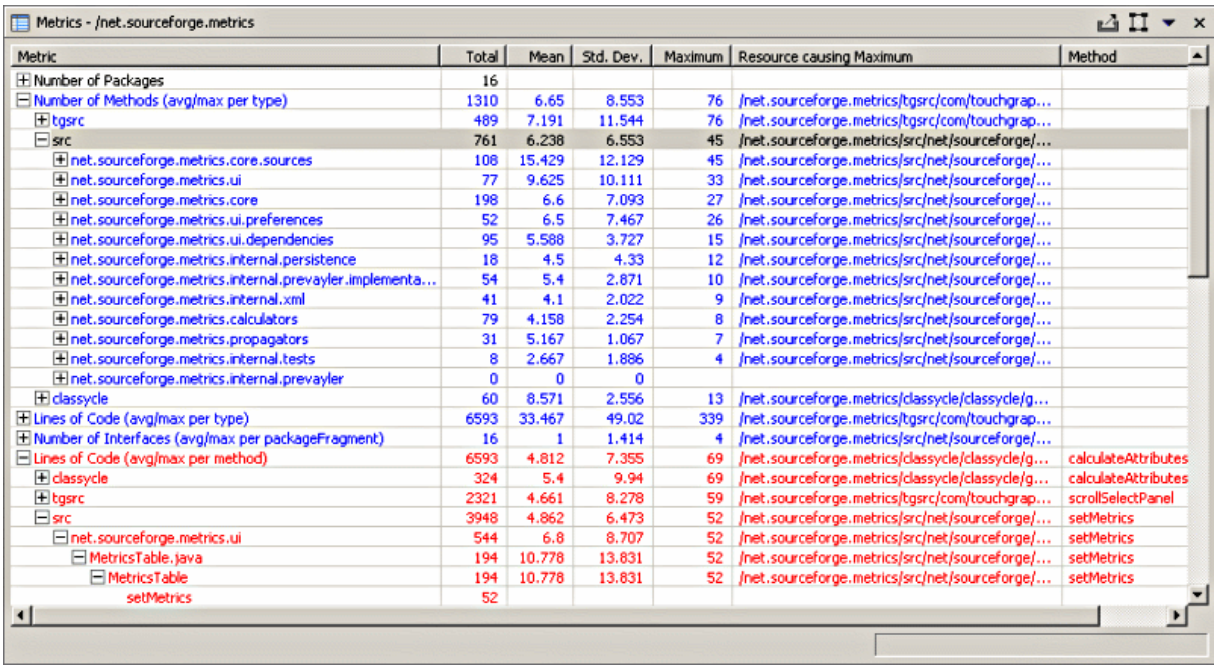
NSM – Number of Static methods	NOC – Number Of Classes
MLOC – New Method Lines Of Code	SIX – Specialization Index
WMC – Weighted Methods per Class	RMI – Instability
NORM – Number of Overridden Methods	NOF – Number Of Attributes
LCOM – Lack of Cohesion Of Methods	PAR – Number of Parameters
VG – McCabe Cyclomatic Complexity	NBD – Nested Block Depth
NSF – Number of Static Attributes	NOM – Number Of Methods
DIT – Depth of Inheritance Tree	RMA – Abstractness
NOP – Number Of Packages	NOI – Number Of Interfaces
TLOC – Total Line Of Code	CE – Efferent Coupling
CA – Afferent Coupling	NSC – Number of Children
RMD – Normalized Distance	

Table 10: The supported metrics in Metrics Eclipse Plug-in

As we see, the metrics are presented with names and abbreviations.
Lines of code has been changed and separated into:

- ***TLOC:*** Total lines of code will count the lines of source code of an executable software entity. It excludes comments and white spaces. Useful for those interested in computed KLOC.
- ***MLOC:*** Method lines of code will count the lines of source code inside method bodies. It excludes comments and white spaces.

The results are displayed as a simple table. However, we cannot in no way sort, save, compare or print these results. The minimum and maximum limits for each metric can be set; however, there are no default values for these limits. Out-of-range and in-range results will be colored in the table. Results can be exported to an XML file. Package Dependency Graph is the only way to plot the metric results.



Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum	Method
Number of Packages	16					
Number of Methods (avg/max per type)	1310	6.65	8.553	76	/net.sourceforge.metrics/tgsrc/com/touchgrap...	
tgsrc	489	7.191	11.544	76	/net.sourceforge.metrics/tgsrc/com/touchgrap...	
src	761	6.238	6.553	45	/net.sourceforge.metrics/src/net/sourceforge/...	
net.sourceforge.metrics.core.sources	108	15.429	12.129	45	/net.sourceforge.metrics/src/net/sourceforge/...	
net.sourceforge.metrics.ui	77	9.625	10.111	33	/net.sourceforge.metrics/src/net/sourceforge/...	
net.sourceforge.metrics.core	198	6.6	7.093	27	/net.sourceforge.metrics/src/net/sourceforge/...	
net.sourceforge.metrics.ui.preferences	52	6.5	7.467	26	/net.sourceforge.metrics/src/net/sourceforge/...	
net.sourceforge.metrics.ui.dependencies	95	5.588	3.727	15	/net.sourceforge.metrics/src/net/sourceforge/...	
net.sourceforge.metrics.internal.persistence	18	4.5	4.33	12	/net.sourceforge.metrics/src/net/sourceforge/...	
net.sourceforge.metrics.internal.prevayler.implementa...	54	5.4	2.871	10	/net.sourceforge.metrics/src/net/sourceforge/...	
net.sourceforge.metrics.internal.xml	41	4.1	2.022	9	/net.sourceforge.metrics/src/net/sourceforge/...	
net.sourceforge.metrics.calculators	79	4.158	2.254	8	/net.sourceforge.metrics/src/net/sourceforge/...	
net.sourceforge.metrics.propagators	31	5.167	1.067	7	/net.sourceforge.metrics/src/net/sourceforge/...	
net.sourceforge.metrics.internal.tests	8	2.667	1.886	4	/net.sourceforge.metrics/src/net/sourceforge/...	
net.sourceforge.metrics.internal.prevayler	0	0	0			
classycle	60	8.571	2.556	13	/net.sourceforge.metrics/classycle/classycle/g...	
Lines of Code (avg/max per type)	6593	33.467	49.02	339	/net.sourceforge.metrics/tgsrc/com/touchgrap...	
Number of Interfaces (avg/max per packageFragment)	16	1	1.414	4	/net.sourceforge.metrics/src/net/sourceforge/...	
Lines of Code (avg/max per method)	6593	4.812	7.355	69	/net.sourceforge.metrics/classycle/classycle/g...	calculateAttributes
classycle	324	5.4	9.94	69	/net.sourceforge.metrics/classycle/classycle/g...	calculateAttributes
tgsrc	2321	4.661	8.278	59	/net.sourceforge.metrics/tgsrc/com/touchgrap...	scrollSelectPanel
src	3948	4.862	6.473	52	/net.sourceforge.metrics/src/net/sourceforge/...	setMetrics
net.sourceforge.metrics.ui	544	6.8	8.707	52	/net.sourceforge.metrics/src/net/sourceforge/...	setMetrics
MetricsTable.java	194	10.778	13.831	52	/net.sourceforge.metrics/src/net/sourceforge/...	setMetrics
MetricsTable	194	10.778	13.831	52	/net.sourceforge.metrics/src/net/sourceforge/...	setMetrics
setMetrics	52					

Figure 54: Eclipse metrics view

The Metrics Eclipse Plug-in does not contain any description for the metrics and does not give any tips how to interpret the results. However, the most of these metrics are good documented in [Henderson-Sellers 1996]. The coupling metrics have been mentioned by Robert Martin in [Martin 2002]

- ***Afferent Coupling (CA):*** It measures the total number of classes outside a package that directly depend upon classes inside the package.
- ***Efferent Coupling (CE):*** It measures the total number of classes inside a package that directly depend upon classes outside the package.
- ***Instability (RMI):*** It is the ratio of efferent coupling to total coupling (afferent plus efferent). This will be a number between 0 and 1
- ***Abstractness (RMA):*** The ratio of the number of internal abstract classes (and interfaces) to the total number of internal classes (and interfaces) in a package
- ***Normalized Distance from Main Sequence (RMD):*** | Abstractness + Instability - 1 |; the range for this metric is 0 to 1. This number should be small, close to zero for good packaging design.

The abbreviations for the coupling metrics do not correspond to known acronyms by Martin [Martin 2002]. He defined it in his book as ***Ca, Ce, I, A, D***.

6.3.5 Metrics One Measurement Level

MetricsOne [Rational 2004] works as a Plug-in to Rational Rose and provides 46 object-oriented analysis and design metrics. The measurements computed by MetricsOne are grounded in the work done at MIT by Chidamber & Kemerer [Chidamber 1994], and are based primarily upon pragmatic insights and recommendations from Lorenz and Kidd

[Lorenz 1994]. The metrics that MetricsOne gathers are grouped into the following categories:

- Class metrics: calculate for example the number of classes, their attributes, and the relationships among it.
- Use Case metrics: calculate for example the number of Use Cases, actors and its relationship with the system
- Operations metrics: calculate for example the number of parameters (arguments) that are in the signature of the operation, the export control of the operation, that is, whether it is public, protected, private, or implementation.
- Packages metrics: calculate for example the number of classes, the dependencies among these classes, coupling between it and its Instability.

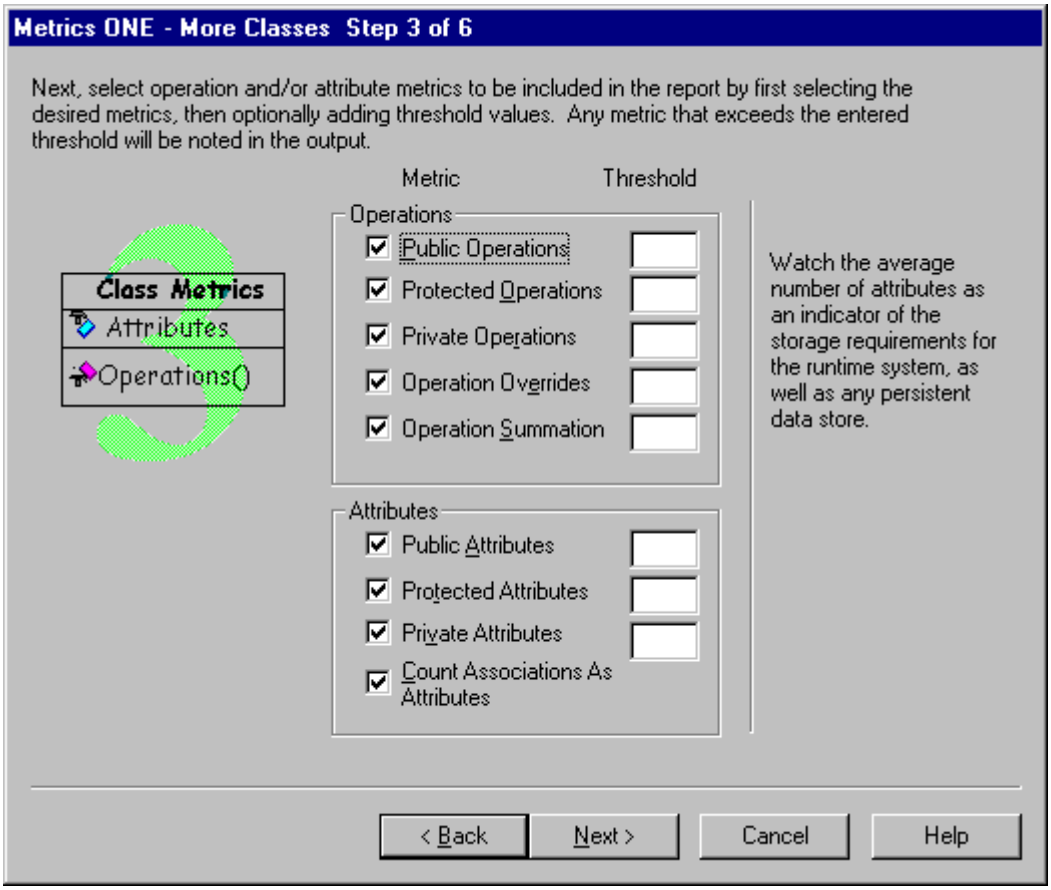


Figure 55: Metrics One Class metrics

It is possible to use existing limits or to define it. New user of MetricsOne can simply take the defaults. More sophisticated users of the tool will collect particular metrics, based upon the issues that are important to their particular project. All settings are saved for use on the next run.

Results will be sent to an Excel spreadsheet, one with a page for each of the metrics categories. Values of any metrics that exceeded limits will appear for example in red. Microsoft Excel must be installed in order for MetricsOne to succeed. Otherwise results can

optionally be sent to a text file (comma delimited) that can later be loaded into Excel or any other tabular-data-oriented application. Developers can then visualize the results in Microsoft Excel.

MetricsOne provides tips about how to use the metrics and how to interpret its results. If specified limits are exceeded, results will appear in a different color (red).

6.3.6 Metrication in Embarcadero RAD Studio 2010

Embarcadero RAD Studio 2010 [RAD 2010] is an integrated CASE-Tool for creating Windows applications. The RAD Studio IDE provides a comprehensive set of tools that streamline and simplify the development life cycle (analysis, design, and implementation). It supports also to make unit tests. RAD Studio supports UML modelling. RAD Studio supports Round-trip engineering for the programming languages C++ and Delphi.

RAD Studio provides 89 object-oriented code metrics for C++ and Delphi. For example Chidamber & Kemerer metrics, MOOD metrics, McCabe- und Halstead metrics. This feature is only for code metrics, it is not available for design projects. For the documentation, the metrics ***CR*** and ***TCR*** are available. RAD Studio provides tips for using metrics and interpreting results. However, it is up to developers to examine the results and decide whether they are acceptable. Metrics results can show up parts of code that should be redesigned.

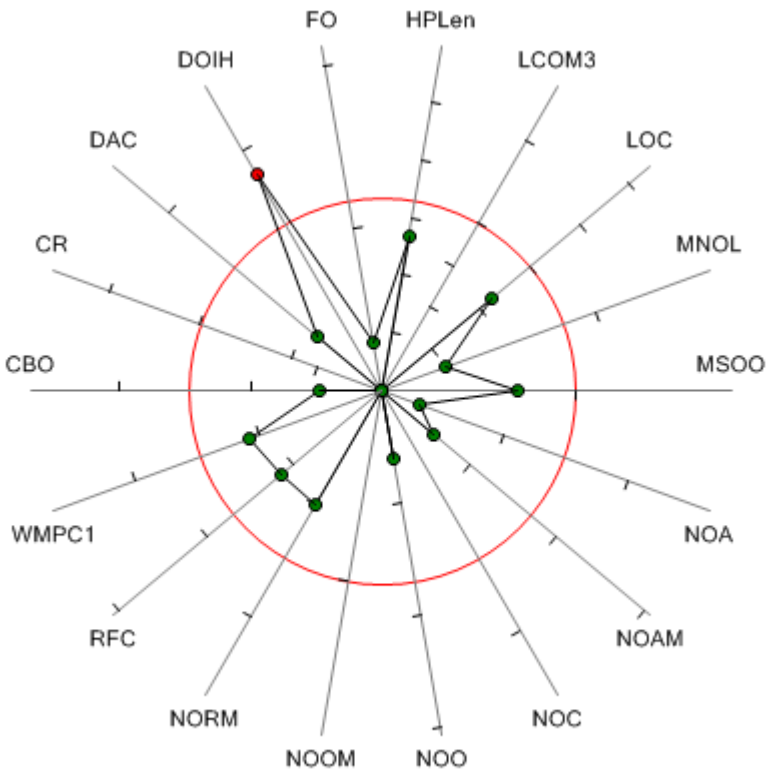


Figure 56: Kiviat charts in RAD Studio

RAD Studio can also show Metrics results graphically. Two graphic views allow summarizing metrics results: bar charts and Kiviat charts. Metrics charts will be created in temporary files, which will be deleted when the charts are closed. However, developers can save information of the chart in text files, export it to the preferred graphical format, and include graphics in the project.

A default subset of all available metrics is active. However, developer can define, save, and reuse sets of metrics that can be loaded and processed if they rather some specific subset.
For each metric there are settings for options such as limits and granularity. Some metrics have already editable default values for this feature. When viewing metric results developers can sort results by column, filter and update results, navigate to the source code, view the metric description. The results can be exported to a XML or HTML file to share them with team members or review them later.

6.4 Evaluation of CASE-Based Quality Assurance

It is no surprise that the tools meet different requirements at measuring. There is a common question that is asked and discussed remains: How efficient and mature is this measurement? This may be due to the missing of quality model by CASE tools. In addition, in some CASE tools it is difficult to learn how to use the metrics and it requires high hardware resources to calculate the values.

Our first evaluation summarizes the general framework steps in their fulfilling characteristics. We use the evaluation as “+” for “fulfilled”, “-“ as “not fulfilled” and “o” as “part fulfilled“. The following table shows this kind of metrics-based evaluation of CASE tool based measurement support.

Tool	Metrics coverage	Metrics providing	Metrics suites	Metrics customizing	Metrics extending	Metrics feedback	Metrics filtering	Sorting results
Together	o	o	+	+	+	+	+	+
Visual Studio	-	o	-	-	-	+	+	+
Enterprise Architect	-	o	-	+	-	-	-	-
Metrics Eclipse Plug-in	o	o	-	+	-	+	-	+
metricsOne	+	o	o	+	-	+	+	+
RAD Studio	+	o	+	+	+	+	+	+

Tool	Metrics visualization	Saving results	Comparing results	Printing results	Exporting results	Copying results	Metrics verification
Together	o	+	+	+	+	-	+
Visual Studio	-	o	-	-	+	+	-
Enterprise Architect	-	-	-	-	+	-	-
Metrics Eclipse Plug-in	o	-	-	-	+	-	+
metricsOne	o	+	+	+	+	+	-
RAD Studio	o	+	o	+	+	-	+

Table 11: Overview of metrics concept

As we see, the described CASE tools don't support all the defined criteria for metrics in different levels and ways.

From the quality measurement point of view we can establish the following general situation considering the CASE tools above.

$$MP(\text{conisdered\_CASE\_tools}) = {}^{\textit{quality\_improvement}}_{\textit{scheduled}}\,MP\,{}^{\textit{aspect\_oriented}}_{\textit{wide\_used\_methodology}}: \tag{6.3}$$

$$\left({}^{\textit{goals\_in\_use}}_{\textit{development}}\,G\,{}^{\textit{improving}}_{\textit{evaluation}} \times {}^{\textit{product\_aspects}}_{\textit{monolithic}}\,A\,{}^{\textit{referred}}_{\textit{original}}\right.$$

$$\left.\times {}^{\textit{inhouse}}_{\textit{improvement}}\,M\,{}^{\textit{closed\_component}}_{\textit{measurement}}\right)^{\textit{automatic}}_{\textit{internal}}\,T\,{}^{\textit{simple\_task}}_{\textit{simple\_tool}},\ {}^{\textit{practitioner}}_{\textit{meas.\_appl.\_staff}}\,P\,{}^{\textit{beginners}}_{\textit{motivated\_user}}$$

$$\rightarrow \left({}^{\textit{threshold}}_{\textit{rules\_of\_thumb}}\,Q\,{}^{\textit{ordinal\_scale}}_{\textit{aggregation}}\right) \rightarrow \left({}^{\textit{formula}}_{\textit{gradient}}\,E\,{}^{\textit{project\_based}}_{\textit{adaptation}} \times {}^{\textit{product\_aspect}}_{\textit{monolithic}}\,A\,{}^{\textit{improved}}_{\textit{original}}\right)$$

In order to simply identify where the possible improvements are, we will add in the formula above the MAX" and the "MIN" characterization. Hence, we obtain the following formula.

$$MP(\text{conisdered\_CASE-tools}) = {}^{\textit{quality\_improvement}}_{\textit{scheduled}}\,MP\,{}^{MIN}_{\textit{wide\_used\_methodology}}: \tag{6.4}$$

$$\left({}^{MAX}_{MIN}\,G\,{}^{\textit{improving}}_{\textit{evaluation}} \times {}^{\textit{product\_aspects}}_{\textit{monolithic}}\,A\,{}^{MAX}_{MAX}\right.$$

$$\left.\times {}^{MAX}_{\textit{improvement}}\,M\,{}^{MIN}_{MAX}\right)^{MAX}_{MAX}\,T\,{}^{MIN}_{MIN},\ {}^{MAX}_{\textit{meas.\_appl.\_staff}}\,P\,{}^{MIN}_{\textit{motivated\_user}}$$

$$\rightarrow \left({}^{\textit{threshold}}_{\textit{rules\_of\_thumb}}\,Q\,{}^{MAX}_{MAX}\right) \rightarrow \left({}^{MAX}_{\textit{gradient}}\,E\,{}^{\textit{project\_based}}_{\textit{adaptation}} \times {}^{\textit{product\_aspect}}_{MAX}\,A\,{}^{\textit{improved}}_{MAX}\right)$$

Some kinds of improvement are ***obvious***. We must only consider the "MIN" characterization. Therefore we can conclude that,

- The measurement of the *product aspects* (in the MP description) using some code metrics in the considered CASE tools could be improved by adding *metrics for the*

whole product measurement such as artefact metrics (for the product model) and documentation metrics (for the developer documentation and the manual)

- The quality measurement goal as "viewpoint" could be improved by using any other product metrics in the acceptance test field or experiences based in their application
- The two kinds of "MIN" in the (measurement) tool description are based on the missing measurement context (such as model *and* implementation metrics) and the missing experience base in these tools because of threshold use only
- The "MIN" in the personal characterization is an assumption and could be improved by the training of the developer as "certified metrics user" etc.

Furthermore, we will use our classification of quality measurement improvement. Some kinds of improvement of the considered CASE tools are as following

- At first we describe a ***weak quality measurement improvement*** that is the fact when we use some "trend analysis" in order to consider the quantities Q in the Visual Studio tool as

$$MP(\text{CASE}_{\text{Vsiual\_Studio}}) \preccurlyeq MP(\text{CASE}_{\text{Visual\_Studio}} \oplus T_{\text{trend\_analysis}}) \qquad (6.5)$$

- Otherwise a ***moderate quality measurement improvement*** that could be achieved by adding a stored threshold history over the development of different projects in the Together tool exemplary as

$$MP(\text{CASE}_{\text{Together}}) \preccurlyeq_{\text{n}} MP(\text{CASE}_{\text{Together}} \oplus T_{\text{experience\_based}}) \qquad (6.6)$$

- An ***essential quality measurement improvement*** would be achieved when we add some product metrics in order to achieve a whole product quality measurement in the Metrics Eclipse Pug-in as

$$MP(\text{CASE}_{\text{Metris\_Eclipse}}) \preccurlyeq MP(\text{CASE}_{\text{Metrics\_Eclispe}} \oplus {}^{product} A) \qquad (6.7)$$

- Finally ***remarkable quality measurement improvement*** could be achieved when we add any project metrics in the Metrics One tool that we achieved a "capability oriented" quality measurement level

$$MP(\text{CASE}_{\text{Metrics\_One}}) \preccurlyeq^{\text{n}} MP(\text{CASE}_{\text{Metrics\_One}} \oplus {}^{product \wedge process} A) \qquad (6.8)$$

Note, that the change by using measures (*V*) indeed of metrics (*Q*) would obviously be a remarkable quality measurement improvement in this quality measurement consideration.

6.5 Summary

In this chapter we have used our framework in order to evaluate a real CASE tool situation in an industrial IT environment. The results of our framework application can be described as follows

- Our investigation of CASE-based software development was circumscribed to the *development of object-oriented information systems*

- We have considered a *typical practical situation* of software development in small companies using well-known CASE tools that includes metrics application

- In a first evaluation we have identified any quality measurement characteristics of six CASE tools and shown that *none of these CASE tools support all essential metrics aspects*

- Using our framework we have seen that *many possibilities exist in order to improve the quality measurement situation* for every considered CASE tool for process and resources measurement mainly

The different kinds of our framework application have demonstrated the essential aspects of validation of this framework respectively.

7 Conclusions and Future Research

Our measurement experiences show that software product measurement can be helpful for software developers and software managers. Though metrics are not a “magic bullet”, they can be useful tool to understand and control the development of complex software application.

The development of software metrics is about 40 years old [cf. Ebert 2007]; nevertheless, software developers and managers in most companies don’t use it or don’t have enough knowledge and experiences in software measurement.

Software developers and managers should have some knowledge about measurement theory and about its techniques, e.g. what metrics are measuring and how to interpret the results. This measurement theory can be provided in the development tool itself. One of the problems with measuring software projects is to understand what the values really mean. There exist many techniques to support this. Of course, there are many development tools available which provide techniques for software measurement, but most of these techniques are difficult to adjust and ambiguous to understand.

We presented in this thesis a framework that could be helpful in order to determine the current (quality) measurement situation. Furthermore we can identify the possibilities of measurement improvement and can plan the meaningful extensions in an industrial environment. Our description of the existing software process evaluations in chapter four gives more information extending our framework with process-based experiences and knowledge as we have used in this thesis.

The application of our framework for existing industrial processes and used CASE tools shows the appropriateness of our approach and gives essential kinds of quality measurement improvement obviously.

Next steps of research in order to improve the quality assurance situation in the CASE-based software development could be

- The refinement of the CASE tool evaluation in order to identify more extensions for quality improvement in the software development
- The prototypical implementation of the CASE tool extension and their practical application
- The extension of the ordinal-based values used in the framework in order to consider more kinds of practical situations.

- The consideration of further software processes including maintenance, evolution and migration in order to build essential experience bases for quality measurement and evaluation

- The investigation of further kinds of CASE tool including modern infrastructures and services

The consideration of the CASE-based software development from a process point of view should also improve the CASE tool situation themselves.

References

[Abran 2006] Abran, A. et al.: *Applied Software Measurement*. Shaker Publ., Aachen, 2006

[Albrecht 1983] Albrecht, A. J.; Gaffney, J. E.: *Software Function, Source Lines of Code, and Development Effort Prediction.* IEEE Transactions on Software Engineering, 9(183)6, pp. 639-648

[Aloisio 2006] Aloisio, G.; Caffaro, M.; Epicoco, I. : *A Grid Software Process.* In: Cunha/Rana: Grid Computing – Software Environments and Tools, Springer Publ., 2006, pp. 75-98

[April 2005] April, A.: *S3m-Model to Evaluate and Improve the Quality of Software Maintenance Process.* Shaker Publ., Aachen, Germany 2005

[Armour 2004] Armour, P. G.: *The Laws of Software Process – A New Model for the Production and Management of Software.* CRC Press, 2004

[Augustine 2005] Augustine, S.; Payne, B.; Sencindiver, F.; Woodcock, S.: *Agile Project Management: Steering from the Edges.* Comm. Of the ACM, 8(2005)12, pp. 85-89

[Basili 2001] Basili, V. R.; Boehm, B. W.: *COTS-Based Systems Top 10 List.* IEEE Computer, May 2001, pp. 91-95

[Basili 2007] Basili, V. R. et al.: *GQM+Strategies.* In: Büren/Bundschuh/Dumke: Praxis der Software-Messung, Shaker Publ., Aachen, 2007, pp. 253-266

[Basili 1986] Basili, V. R.; Selby, R. W.; Hutchens, D. H.: *Experimentation in Software Engineering.* IEEE Transactions on Software Engineering, 12(1986)7, pp. 733-743

[Bauer 1999] Bauer, M.: *Analyzing Software Systems by Using Combinations of Metrics*. Technical Report, Forschungszentrum Informatik Karlsruhe, 1999

[Bergstra 2001] Bergsta, J. A.; Ponse, A.; Smolka, S. .: *Handbook of Process Algebra.* Elsevier Publ., 2001

[Bielak 2000] Bielak, J.: *Improving Size Estimate Using Historical Data.* IEEE Software, Nov./Dec. 2000, pp. 27-35

[Biffl 2000] Biffl, S.: *Using Inspection Data for Defect Estimation.* IEEE Software, Nov./Dec. 2000, pp. 36-43

[Blazey 2002] Blazey, M.: *Softwaremessansätze für komponentenbasierte Produkttechnoogien am Beispiel der EJB.* Diploma Thesis, University of Magdeburg, Dept. of Computer Science, 2002

[Boehm 2000] Boehm, B. W.: *Software Cost Estimation with COCOMO II.* Prentice Hall, 2000

[Boehm 1984] Boehm, B. W.: *Software Engineering Economics.* IEEE Transactions on Software Engineering, 10(19841, pp. 4-21

[Boehm 1989] Boehm , B.W.: *Software Risk Management*. IEEE Computer Society Press, 1989

[Boehm 2000a] Boehm, B. W.: *Software Estimation Perspectives.* IEEE Software, Nov./Dec. 2000, pp. 22-26

[Boehm 2000b] Boehm, B. W.; Basili, V. R.: *Gaining Intellectual Control of Software Development.* IEEE Software, May 2000, pp. 27-33

[Boehm 2005] Boehm, B. W.; Turner, R.: *Management Challenges to Implementing Agile Processes in Traditional Development Organizations.* IEEE Software, Sept./Oct. 2005, pp. 30-39

[Braungarten 2006] Braungarten, R.; Kunz, R.; Dumke, R.: *Service-orientierte Software-Messinfrastrukturen.* Presentation at the Bosch Metrics Community, Stuttgart, March 2006

[Braungarten 2005] Braungarten, R.; Kunz, M.; Dumke, R.: *An Approach to Classify Software Measurement Storage Facilities.* Preprint No 2, University of Magdeburg, Dept. of Computer Science, 2005

[Braungarten 2005a] Braungarten, R.; Kunz, M.; Farooq, A.; Dumke, R.: *Towards Meaningful Metrics Data Bases.* Proc. of the 15th IWSM, Montreal, Sept. 2005, pp. 1-34

[Bundschuh 2000] Bundschuh M.: *Aufwandschätzung von IT-Projekten,* MITP Publ., Bonn, 2000

[Bundschuh 2008] Bundschuh, M.; Dekkers, C.: *The IT Measurement Compendium.* Springer Publ., 2008

[Chang 2000] Chang, S. K.: *Multimedia Software Engineering.* Kluwer Academic Publisher, 2000

[Chidamber 1994] Chidamber S. R.; Kemerer, C. F.: *A metrics suite for object oriented design*. IEEE Transactions on Software Engineering, 20(6), pp. 476-98, 1994

[Chrissis 2003] Chrissis, M. B.; Konrad, M.; Shrum, S.: *CMMI – Guidelines for Process Integration and Product Improvement.* Addison-Wesley 2003

[Chung 2000] Chung, L.; Nixon, B. A.; Yu, E.; Mylopoulos, J.: *Non-Functional Requirements in Software Engineering.* Kluwer Academic Publ., 2000

[Clements 2005] Clements, P. C.; Lawrence, G. J.; Northop, L. M.; McGregor, J. D.: *Project Management n a Software Product Line Organization,.* IEEE Software, Sept./Oct. 2005, pp. 54-62

[Davis 1995] Davis, A. M.: *201 Principles of Software Development.* McGraw Hill Publ., 1995

[Deek 2005] Deek, F. P.; McHugh, J. A. M.; Eljabiri, O. M.: *Strategic Software Engineering – An Interdisciplinary Approach.* Auerbach Publications, Boca Raton London New York, 2005

[Donzelli 2006] Donzelli P.: *A Decision Support System for Software Project Management.* IEEE Software July/August 2006, pp. 67-74

[Dreger 1989] Dreger, J. B.: *Function Point Analysis.* Prentice Hall, 1989

[Dumke 1999] Dumke, R.; Foltin, E.: *An Object-Oriented Software Measurement and Evaluation Framework.* Proc. of the FESMA, October 4-8, 1999, Amsterdam, pp. 59-68

[Dumke 2000] Dumke, R.; Foltin, E.; Schmietendorf, A.: *Metriken-Datenbanken in der Informations-verarbeitung.* Preprint No 8 University of Magdeburg, Dept. of Computer Science, 2000

[Dumke 2003] Dumke, R.; Lother, M.; Wille, C.; Zbrog, F.: *Web Engineering.* Pearson Education Publ., 2003

[Dumke 2003a] Dumke, R.: *Software Engineering – Eine Einführung für Informatiker und Ingenieure.* (4th edn) Vieweg Publ., 2003

[Dumke 2004] Dumke, R.; Cotè, I.; Andruschak, O.: *Statistical Process Control (SPC) – A Metrics-Based Point of View of Software Processes Achieving the CMMI Level Four.* Preprint No. 7, University of Magdeburg, Fakultät für Informatik, 2004

[Dumke 2005] Dumke, R.; Schmietendorf, A.; Zuse, H.: *Formal Descriptions of Software Measurement and Evaluation - A Short Overview and Evaluation.* Preprint No. 4, Fakultät für Informatik, University of Magdeburg, 2005

[Dumke 2005a] Dumke, R.; Richter, K.; Fetcke, T.: *FSM Influences and Requirements in CMMI-Based Software Processes.* In: Abran et al.: Innovations in Software Measurement. Shaker Publ., 2005, pp. 179-194

[Dumke 2005b] Dumke, R.; Kunz, M.; Hegewald, H.; Yazbek, H.: *An Agent-Based Measurement Infrastructure.* Proc. of the IWSM 2005, Montreal, Sept. 2005, pp. 415-434

[Dumke 2005c] Dumke, R.: *Software Measurement Frameworks.* Proc. of the 3rd World Congress on Software Quality, Munich, Sept. 2005, Online Proceedings

[Dumke 2006] Dumke, R.; Blazey, M.; Hegewald, H.; Reitz, D.; Richter, K.: *Causalities in Software Process Measurement.* Accepted to the MENSURA 2006, Cardiz, Spain, Nov. 2006

[Dumke 2008] Dumke, R.;Kunz, M.; Farooq, A.; Georgieva, K.; Hegewald, H.: *Formal Modeling of Software Measurement Levels of Paradigm-Based Apporaches.* Technical Report FIN-013-2008, University of Magdeburg, Germany

[Dumke 2008a] Dumke, R. et al.: *Software Process and Product Measurement.* LNCS 5338, Springer Publ, 2008

[Dumke 2010] Dumke, R.; Mencke, S.; Wille, C.: *Quality Assurance of Agent-Based and Self-Managed Systems.* CRC Press, Boca Raton 2010

[EA 2010] Enterprise Architect 8.0: http://www.sparxsystems.de/ (accessed 2010-07)

[Ebert 2004] Ebert, C.; Dumke, R.; Bundschuh, M.; Schmietendorf, A.: *Best Practices in Software Measurement.* Springer Publ., 2004

[Ebert 2005] Ebert, C.: *Systematisches Requirements Engineering.* dpunkt.Verlgag, Germany, 2005

[Ebert 2007] Ebert, C.; Dumke, R.: *Software Measurement.* Springer Publ., 2007

[Eclipse 2010] Metrics Eclipse Plug-in: http://metrics.sourceforge.net/ (accessed 2010-07)

[Emam 2005] Eman, K. E.: *The ROI from Software Quality.* Auerbach Publ., 2005

[Emam 1998] Emam, K. E.; Drouin, J. N.; Melo, W.: *SPICE – The Theory and Practie of Software Process Improvement and Capability Determination* IEEE Computer Society Press, 1998

[Endres 2003] Endres, Albert; Rombach, D.: *A Handbook of Software and System Engineering*. Pearson Education Limited, 2003

[Farooq 2008] Farooq, A.; Kernchen, S.; Dumke, R. R.; Wille, C.: *Web Services based Measurement for IT Quality Assurance.* In: Cuadrado-Gallege t al.: Software Product and Process Measurement. LNCS 4895, Springer Publ., Berlin Heidelberg, 2008

[Fenton 1997] Fenton, N. E.; Pfleeger, S. L.: *Software Metrics - a rigorous and practical approach.* Thompson Publ., 1997

[Ferguson 1998] Ferguson, J.; Sheard, S.: *Leveraging Your CMM Efforts for IEEE/EIA 12207.* IEEE Software, September/October 1998, pp. 23-28

[Fetcke 1999] Fetcke, T.: *A Generalized Structure for Function Point Analysis.* Proc. of the 11th IWSM, Lac Superieur, Canada, Sept. 1999, pp. 143-153

[Florac 1999] Florac, W. A.; Carleton, A. D.: *Measuring the Software Process – Satistical Process Conrol for Software Process Improvement.* Addison-Wesley Publ., 1999

[Florac 2000] Floac, W. A.; Carleton, A. D.; Barnard, J. R.: *Statistical Process Control: Analyzin a Spac Shuttle Onboard Software Process.* IEEE Software, July/August 2000, pp. 97-106

[Gadatsch 2005] Gadatsch, A.; Mayer, E.: *Masterkurs – IT Controlling.* Vieweg Publ., 2005

[Garcia 2005] Garcia, S.: *How Standards Enable Adoption of Project Management Practice.* IEEE Software, Sept./Oct. 2005, pp. 22-29

[Hale 2000] Hale, J.; Parrish, A.; Dixon, B.; Smith, R. K.: *Enhancing the Cocomo Estimation Models.* IEEE Software, Nov./Dec. 2000, pp. 45-49

[Halstead 1977] Halstead, M. H.: *Elements of Software Science.* Prentice Hall, New York, 1977

[Hansen 2006] Hansen, K. T.: *Project Visualization for Software.* IEEE Software, July/August 2006, pp. 84-92

[Harrison 1998] Rachel Harrison R; Counsell, S. J.: *An Evaluation of the MOOD Set of Object-oriented software metrics*. IEEE Transactions on Software Engineering, Vol. 24, No. 6, June 1998

[Haywood 1998] Haywood, M.: *Managing Virtual Teams – Practical Techniques for High-Technology Project Managers.* Artech House, Boston, London, 1998

[Hegewald 1991] Hegewald, H.: *Implementation des Prototyps eines Softwarebewertungsplatzes.* Diploma Thesis, University of Magdeburg, Dept. of Computer Science, 1991

[Henderson-Sellers 1996] Henderson-Sellers, B.: *Object-Oriented Metrics, measures of Complexity*. Prentice Hall, 1996

[Hill 1999] Hill, P.: *Software Project Estimation.* KWIK Publ., Melbourne, 1999

[Horn 2002] Horn, E.; Reinke, T.: *Softwarearchitektur und Softwarebauelemente.* Hanser Publ., 2002

[Humphrey 2000] Humphrey, W. S.: *The Personal Software Process: Status and Trends.* IEEE Software, Nov/Dec. 2000, pp. 71-75

[ISBSG 2003] ISBSG Software Project Estimation – *A Workbook for Macro-Estimation of Software Development Effort and Duration.* Melbourne, 2003

[ISO 15939] ISO/IEC 15939: *Information Technology – Software Measurement Process*. Metrics News, Vol. 6, No. 2, pp. 11-46, 2001

[ISO 9126] ISO/IEC 9126: *Software Engineering – Product Quality*. 2003

[ITIL 2006] The ITIL Home Page*, http://www.itil.org.uk/what.htm,* (see July 24, 2006)

[Johnson 2007] Johnson, B.; Higgins, J.: *ITIL and the Software Lifecycle: Practical Strategy and Design Principles.* Van Haren Publ., Netherlands, 2007

[Johnson 2005] Johnson, P. M.; Kou, H.; Paulding, M.;Zhang, Q.; Kagaw, A.; Yamashita, T.: *Improving Software Development Management through Software Project Telemetry.* IEEE Software, July/August 2005, pp. 78-85

[Jones 1991] Jones, C.: *Applied Software Measurement – Assuring Productivity and Quality* McGraw Hill Publ., 1991

[Juristo 2003] Juristo, N.; Moreno, A. M.: *Basics of Software Engineering Experimentation*. Kluwer Academic Publishers, Boston, 2003

[Kamatar 2000] Kamatar, J.; Hayes, W.: *An Experience Report on the Personal Software Process.* IEEE Software, Nov/Dec. 2000, pp. 85-89

[Kandt 2006] Kandt, R. K.: *Software Engineering Quality Practices.* Auerbach Publications, Boca Raton New York, 2006

[Kenett 1999] Kenett, R. S.; Baker, E. R.: *Software Process Quality – Management and Control.* Marcel Dekker Inc., 1999

[Keyes 2003] Keyes, J.: *Software Engineering Handbook* Auerbach Publ., 2003

[Kitchenham 2007] Kitchenham, B: *Empirical Paradigm – The Role of Experiments*. In: Basili et al.: Empirical Software Engineering, Springer Publ., pp. 25-32, 2007

[Kitchenham 1997] Kitchenham et al.: *Evaluation and assessment in software engineering.* Information and Software Technology, 39(1997), pp. 731-734

[Kulpa 2003] Kulpa, M. K.; Johnson, K. A.: *Interpreting the CMMI – A Process Improvement Approach.* CRC Press Company, 2003

[Kunz 2006] Kunz, M.; Schmietendorf, A.; Dumke, R.; Wille, C.: *Towards a Service-Oriented Measurement Infrastructure.* Proc. of the 3rd Software Measurement European Forum (SMEF), May 10-12, 2006, Rome, Italy, pp. 197-207

[Laird 2006] Laird, L. M.; Brennan, M. C*.: Software Measurement and Estimation – A Practical Approach*. IEEE Computer Science, 2006

[Lanza 2006] Lanza, M.; Marinescu, R.: *Object-Oriented Metrics in Practice*. Springer Pbl., Hdelberg, New York 2006

[Lecky-Thompson 2005] Lecky-Thompson, G. W.: *Corporate Software Project Management.* Charles River Media Inc., USA, 2005

[Lepasaar 2001] Lepasaar, M.; Varkoi, T.; Jaakkola, H.: *Models and Succes Factors of Process Change.* In: Bomarius/Komi-Sirviö: Product Focused Software Process Improvement. PROFES 2001, Kaiserslautern, Sept. 2001, LNCS 2188, Springer Publ., 2001, pp. 68-77

[Lewerentz 2000] Lewerentz C.; Rust, H.; Simon, F.: *Quality – Metrics – Numbers – Consequences*. In: Dumke, R.; Lehner, .F: Software-Metriken, DUV Publ., Wiesbaden. 2000

[Lewerentz 1998] Lewerentz C.; Simon, F.: *A Product Metrics Tool Integrated into a Software Development Environment*. Technical Report, University of Cottbus, 1998

[Lokan 2001] Lokan, C.; Wright, T.; Hill, P. R; Stringer, M.: *Organizational Benchmarking Using the ISGSG Data Repository.* IEEE Software, Sept./Oct. 2001, pp. 26-32

[Lorenz 1994] Lorenz M.; Kidd, J.: *Object-Oriented Software metrics – A practical guide*. Prentice Hall, New Jersey, 1994

[Lother 2004] Lother, M.; Braungarten, R.; Kunz, M.; Dumke, R.: *The Functional Size e-Measurement Portal (FSeMP).* In: Abran et al: Software Measurement – Research and Application, Shaker Publ., 2004, pp.27-40

[Lother 2001] Lother, M.; Dumke, R.: *Point Metrics – Comparison and Analysis.* In Dumke/Abran: Current Trend in Software Measurement, Shaker Publ., 2001

[Maciaszek 2001] Maciaszek, L. A.: *Requirements Analysis and System Design – Development Informatik Systems with UML.* Addison Wesley Publ., 2001

[Marciniak 1994] Marciniak, J. J.: *Encyclopedia of Software Engineering.* Vol. I and II, John Wiley & Sons Inc., 1994

[Marinescu 1998] Marinescu, R.: *An Object Oriented Metrics Suite on Coupling*. Universitatea „Politehnica“Timisoara, Facultatea de Automatica si Calculatoare, Departamentul de Calculatoare si Inginerie Software, Diploma Thesis, September 1998.

[Martin 2002] Martin, R. C.: *Agile Software Development. Principles, Patterns, and Practices*. Prentice Hall International, 2002

[McConnel 2006] McConnel, S.: *Software Estimation – Demystifying the Black Art.* Microsoft Press, 2006

[Messerschmitt 2003] Messerschmitt, D. G.; Szyperski, C.: *Software Ecosystem – Understanding an Indispensable Technology and Industry.* MIT Press, 2003

[Mikkelsen 1997] Mikkelsen, T.; Phirego, S.: *Practical Software Configuration Management.* Prentice Hall Publ. 1997

[Milner 1989] Milner, R.: *Communication and Concurrency.* Prentice Hall Publ., 1989

[Munson 2003] Munson, J., C.: *Software Engineering Measurement.* CRC Press Company, Boca Raton London New York, 2003

[Nidiffer 2005] Nidiffer, K. .; Dolan, D.: *Evolving Distributed Project Management.* IEEE Software, Sept./Oct. 2005, pp. 63-72

[Pandian 2004] Pandian, C. R.: *Software Metrics – A Guide to Planning, Analysis, and Application.* CRC Press Company, 2004

[Pfleeger 1998] Pfleeger, S. L.: *Software Engineering – Theory and Practice.* Prentice-Hall Publ., 1998

[Putnam 2003] Putnam, L. H.; Myers, W.: *Five Core Metrics – The Intelligence Behind Successful Software Management.* Dorset House Publishing, New York, 2003

[Putnam 1992] Putnam, L. H.; Myers, W.: *Measures for Excellence – Reliable Software in Time, within Budgets.* Yourdon Press Publ., 1992

[RAD 2010] Embarcadero RAD Studio 2010**:** http://www.embarcadero.com/products/rad-studio (accessed 2010-07)

[Rational 2004] metricsOne Rational Rose Plug-in: Tools and associated documentation are available at the University Magdeburg, 2004

[Reitz 2005] Reitz, D.; Schmietendorf, A.; Dumke, R.: *Tool supported monitoring and estimations in EAI multi projects.* Proc. of the IWSM 2005, Montreal, Sept. 2005, pp. 53-66

[Reitz 2003] Reitz, D.; Schmietendorf, A.; Dumke, R.; Lezius, J.; Schlosser, T.: *Aspekte des empirischen Software Engineering im Umfeld von Enterprise Application Integration.* Preprint No 5, University of Magdeburg, Dept. of Computer Science, 2003

[Richter 200] Richter, K.: *Softwaregrößenmessug im Kontext von Software-Prozessbewertungsmodellen.* Diploma Thesis, University of Magdeburg, 2005

[Royce 1998] Royce, W.: *Software Project Management.* Addison-Wesley, 1998

[Royce 2005] Royce, W.: *Successful Software Management Style: Steering and Balance.* IEEE Software, Sept./Oct. 2005, pp. 40-47

[Schmietendorf 2003] Schmietendorf, A.; Dumke, R.: *Performance analysis of an EAI application integration.* Proc. of the UKPE, Warwick, July 2003, pp. 218-230

[Schmietendorf 2004] Schmietendorf, A.; Reitz D.; Dumke, R.: *Project reporting in the context of an EAI project with the aid of Web-based portal.* Proc. of the CONQUEST 2004, Nuremberg, Sept. 2004, pp. 47-57

[SEI 2002] SEI: *Capability Maturity Model Integration (CMMI^{SM}),* Version 1.1, Software Engineering Institute, Pittsburgh, March 2002, CMMI-SE/SW/IPPD/SS, V1.1

[Singpurwalla 1999] Singpurwalla, N. D.; Wilson, S. P.: *Statistical Methods in Software Engineering.* Springer Publ., 1999

[Skyttner 2005] Skyttner, L.: *General Systems Theory – Problems, Perspectives, Practice*. World Scientific Publ., New Jersey, 2005

[Sneed 1990] Sneed, H.: *Die Data-Point-Methode.* Online, DV Journal, May 1990, pp.48

[Sneed 1996] Sneed, H.: *Schätzung der Entwicklungskosten von objektorientierter Software.* Informatik-Spektrum, 19(1996)3, pp. 133

[Sneed 2005] Sneed, H.: *Software-Projektkalkulation.* Hanser Publ., 2005

[Solingen 1999] Solingen, v. R.; Berghout, E.: *The Goal/Question/Metric Method.* McGraw Hill Publ., 1999

[Sommerville 2007] Sommerville, I.: *Software Engineering*. Addison Wesley – Eighth Edition , 2007

[SPICE 2006] The SPICE Web Site, *http://www.sqi.gu.edu.au/spice/* (seen July 24, 2006)

[Tayntor 2003] Tayntor, C. B.: *Six Sigma Software Development*. CRC Press, 2003

[Together 2010] Together: http://techpubs.borland.com/together/ (accessed 2010-07)

[Ullwer 2006] Ullwer, C.: *Konzeption und prototypische Realisierung einer Telemetrie-basierten Mess-Architektur.* Diploma Thesis, University of Magdeburg, Dept. of Computer Science, July 2006

[Venugopal 2005] Venugopal, C.: *Single Goal Set: A New paradigm for IT Megaproject Success.* IEEE Software, Sept./Oct. 2005, pp. 48-53

[Verzuh 2005] Verzuh, E.: *The Fast Forward MBA in Project Management.* John Wiley & Sons, 2005

[VS 2010] Microsoft Visual Studio 2010: http://www.microsoft.com/visualstudio/en-us (accessed 2010-07)

[Walter 2006] Walter, Z.; Scott, G.: *Management Issues of Internet/Web Systems.* Comm. of the ACM, 49(2006)2, pp.87-91

[Wang 2000] Wang, Y.; King, G.: *Software Engineering Processes – Principles and Applications.* CRC Press, Boca Raton London New York, 2000

[Wangenheim 2006] Wangenheim, C. .v.; Anacleto, A.; Saliano, C. F.: *Helping Small Companies Assess Software Processes.* IEEE Software, Jan./Febr. 2006, pp. 91-98

[Wasserman 1990] Wasserman, A.: *Tool Integration in Software Engineering Environments.* Proc. of the Int. Workshops on Environments, LNCS 322, Springer Publ., 1990, pp. 137-149

[White 2004] White, S.A.: *Introduction to the BPMN.* IBM Corporation, 2004

[Whitmire 1997] Whitmire, S.A.: *Object Oriented Design Measurement.* John Wiley & Sons, 1997

[Whitmire 1992] Whitmire, S.: *3-D Function Points: Scientific and Real-time Extensions of Function Points.* Proc. of the Pacific Northwest Software Quality Conference, 1992

[Wille 2006] Wille, C.; Braungarten, R.; Dumke, R.: *Addressing Drawbacks of Software Measurement Data Integration.* Proc. o the SMEF 2006, Rome, Italy, May 2006

[Wohlin 2000] Wohlin, C, Runeson, P, Höst, M, Ohlsson, M, Regnell, B, Wesslén, A.: *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, Boston, 2000

[Wong 2001] Wong, B. Jefferey, R.: *Cognitive Structures of Software Evaluation: A Means-End Chain Analysis of Quality.* In: Bomarius/Komi-Sirviö: Product Focused Software Process Improvement. PROFES 2001, Kaiserslautern, Sept. 2001, LNCS 2188, Springer Publ., 2001, pp. 6-26

[Yazbek 2007] Yazbek, H.: *Metrikenkonzepte von CASE-Tools am Beispiel von Together.* Master Thesis, University of Magdeburg, Faculty of Computer Science, 2007

[Yazbek 2010] Yazbek, H.: *A Conept of Quality Assurance for Metrics in CASE Tools.* Software Engineering Notes, Sept. 2010, p. 32

[Yazbek 2010a] Yazbek, H.: *Metrics Support in Industrial CASE Tools.* Software Measurement News, 15(2010)2, pp. 13-26

[Yazbek 2010b] Yazbek, H.: *Service-oriented Measurement Infrastructure.* In: O. Ormandjieva; C. Constantinides; A. Abran; R. Lee: IEEE-SERA 2010, IEEE Computer Society Los Alamitos, California, pp. 303-308

[Zelkowitz 207] Zelkowitz, M. V.: *Techniques for Empirical Validation*. In: Basili et al.: Empirical Software Engineering, Springer-Publ., pp. 4-9, 2007

[Zelkowitz 1997] Zelkowitz, M. V.; Wallace, D. R.: *Experimental Models for Validating Technology.* IEEE Computer, May 1998, pp. 23-31

[Zettel 2001] Zettel, J.; Maurr, F.; Münch, J.; Wong, L.: *LIPE: A Lightweight Process for E-Business Startup Companies Based on Extreme Programming.* In: Bomarius/Komi-Sirviö: Product Focused Software Process Improvement. PROFES 2001, Kaiserslautern, Sept. 2001, LNCS 2188, Springer Publ., 2001, pp. 255-270

[Zhong 2000] Zhong, X.; Madhavji, N. H. Emam, K. E.: *Critical Factors Affecting Personal Software Processes.* IEEE Software, Nov./Dec. 2000, pp. 76-83

[Zuse 1998] Zuse, H.: *A Framework of Software Measurement*. de Gruyter Publ., Berlin, 1998

[Zuse 2003] Zuse, H.: *What can Practioneers learn from Measurement Theory.* In: Dumke et al.: Investigations in Software Measurement, Proc. of the IWSM 2003, Montreal, September 2003, pp. 175-176