ISCN

International Software Consulting Network

APS COMETT

Practical Improvement of Software Processes and Products

ISCN '94

4–5 May 1994, Dublin, Ireland

Principles

Goals
Quality
Metrics and
Processes

Approaches

SCOPE

ami

BOOTSTRAP!

metkit

ISO 9000

Benefits

Festo
Motorola
Alcatel
Brameur
Objectif Technologie
Siemens

Supported by :

APS Comett
European Software Institute (ESI)
Hibernia UETP
ISCN
K&M Technologies Ltd
National Software Directorate
Ovum Ltd
Q-SET Ltd
University of Technology,Graz

Who Should Attend?

The ISCN seminar and workshop will benefit all those concerned with management and improvement of software quality in their organisations.

Software quality has a marked effect on the effectiveness of the overall business processes, irrespective of being a software developer or a software user. Process improvement is applicable equally to manufacture and procurement of software, and facilitates the attainment of ISO 9000 certification.

Specifically the seminar and workshop addresses business development executives, IT managers, development managers, quality managers and software practitioners.

You cannot afford to miss this event.

Background

"Product quality is determined by process quality." This is just as true for business and manufacturing processes involving software as it applies to traditional manufacturing industries.

How do you determine the quality of products and processes involving software? What underlying techniques are promising? How do they relate to SEI's CMM? How can these effectively be integrated into a business and organisational framework—and produce a return on investment? How do these techniques relate to ISO 9000—and is there an additional benefit to be gained?

ESPRIT programmes have enhanced the state-of-the-art, but how usable are the results? What technology transfer initiatives are available to industry, and what is the strategy for Europe?

Approaches

ami provides an overall improvement framework based on the application of quantitative methods: the Assess-Analyse-Metricate-Improve paradigm.
Bootstrap is a method for assessing and quantitatively evaluating process quality attributes. It directly facilitates ISO 9000 certification programmes.
ESI–the **European Software Institute** was founded by industry in 1993 with the specific aim to improve software practice in European industry by using the methods presented here as its baseline technology.
Iscn is a network of independent process improvement consultants supporting analysis and installation of improvements.
Metkit has developed a training programme for software metric selection and installation in an organisation.
Scope has developed procedures for and a guide for the evaluation of software product quality.

Industrial Relevance

Benefits realised by the various approaches to improving the quality of software processes and products will be presented by a variety of speakers from companies committed to quality improvement.

In a workshop style atmosphere the experts for process and product improvement technology will answer questions and respond to problems formulated by the audience. The respresentatives from industry will act as referees for industrial relevance and realism in this part of the workshop.

Prospective delegates are invited to submit issues and queries with their registration.

Seminar Programme

Day One

Chairman of the Days Proceedings:

Takis Katsoulakos, *Lloyds Register, Chairman of ESI*

09:30	Official Opening
	"Europe – Lacking a Software Strategy" Barry Murphy, *National Software Directorate*
	"Framework Programme IV for R&D – Best Practise for Software Engineering" Rainer Zimmermann, *CEC DG III*
10:15	"European Strategies for Software Process and Product Improvement" Günter Koch, *Managing Director ESI*
10:45	Coffee
11:15	"BOOTSTRAP and ISO 9000: A Quantitative Approach to Objective Quality Management" Richard Messnarz, *ISCN*
12:00	"ami: A New Paradigm for Software Process Improvement" Christophe Debou, *Alcatel*
12:45	Lunch
14:15	"SCOPE: A Guide for Software Product Quality Evaluation" Jørgen Bøegh, *DELTA Software Engineering*
15:00	"METKIT: How to Cope with Software Complexity?" Horst Zuse, *Technische Universitat Berlin*
15:30	Coffee
16:00	"Process Improvement: How Much Can the Organisation Endure?" Hans-Jürgen Kugler, *ISCN*
16:30	Open Forum – "The Improvement Soapbox" open to seminar participants, 5-10 minutes each

Day Two

Chairman of the Days Proceedings:

Günter Koch, *Managing Director ESI*

09:00	"Process and Product Quality Improvement: Approaches, Experience, Results – Part 1 Christophe Debou, *Alcatel* Mike Kelly, *Brameur* Roberto Galimberti, *BOOTSTRAP Institute* Manfred Koch, *FESTO* plus seminar participants
10:30	Coffee
11:00	"Process and Product Quality Improvement: Approaches, Experience, Results – Part 2 John Sheehy, *Motorola* Annie Combelles, *Objectif Technologie* Axel Völker, *Siemens* plus seminar participants
12:30	Lunch
14:00	"But – will this work for me?" Discussion and Workshop Questions and Problems: seminar participants Answers and Solutions: technology experts and industrial users
15:00	Coffee
16:00	Summary and Close of Workshop Günter Koch, *Managing Director ESI* Hans-Jürgen Kugler, *ISCN*

BOOTSTRAP and ISO 9000:

A Quantitative Approach to Objective Quality Management

Richard Messnarz†
Hans-Jürgen Kugler‡
Volkmar Haase †

Abstract

The aim of the BOOTSTRAP project was to develop a method for software process assessment, quantitative measurement, and improvement [3]. BOOTSTRAP enhanced and refined the SEI method ([11], [13]) for software process assessment and adapted it to the needs of the European software industry including non-defence sectors to make it applicable to all kinds of SPUs [4]. An SPU (Software Producing Unit [4]) is a small or middle sized software producing company or a software department of a large company that runs projects to develop software products. The SPU defines quality policies and guidelines, estimates and provides resources, manages the training of the engineers, and defines standards, practices and methods that have to be employed by the projects. The BOOTSTRAP method assesses both the SPU and the projects of the SPU: does the SPU provide the necessary resources and how efficiently do the projects use these resources? A detailed process quality attribute hierarchy ([9], [4]) enhancing the SEI questionnaire and taking into account ISO 9000-3 guidelines [12] for software quality assurance and the ESA PSS 05 software engineering standards [6] forms the basis of the BOOTSTRAP method. The SEI maturity level algorithm [9] was refined to be able to calculate a maturity level for each of the individual process quality attributes resulting in a *quantitative process quality profile* [9] that provides a representation of the strengths and weaknesses. This quality profile serves as a quantitative basis for making decisions about process improvements and allows to evaluate the degree of satisfaction of about 85% of the ISO attributes.

1. Process Measurement Approach

A quality model starts with the definition of a quality goal. Quality attributes are assigned to this goal, and quality factors are then assigned to each quality attribute. Quality factors might again recursively consist of quality factors leading to a *quality attribute hierarchy* is (Fig. 1, [14]).

If software metrics are assigned to the leaves of this quality attribute hierarchy [14] and if these quality factors are evaluated according to the defined metrics a set of measured values will be obtained which represents the quality of the process. As a metric we assign the BOOTSTRAP maturity level algorithm (section 3, [9]) to each process quality factor and thus obtain a *quantitative process quality/maturity profile* (see Fig. 2, Fig. 3).

---

† University of Technology, Graz, Austria

‡ K&M Technologies Ltd, Bray, Ireland

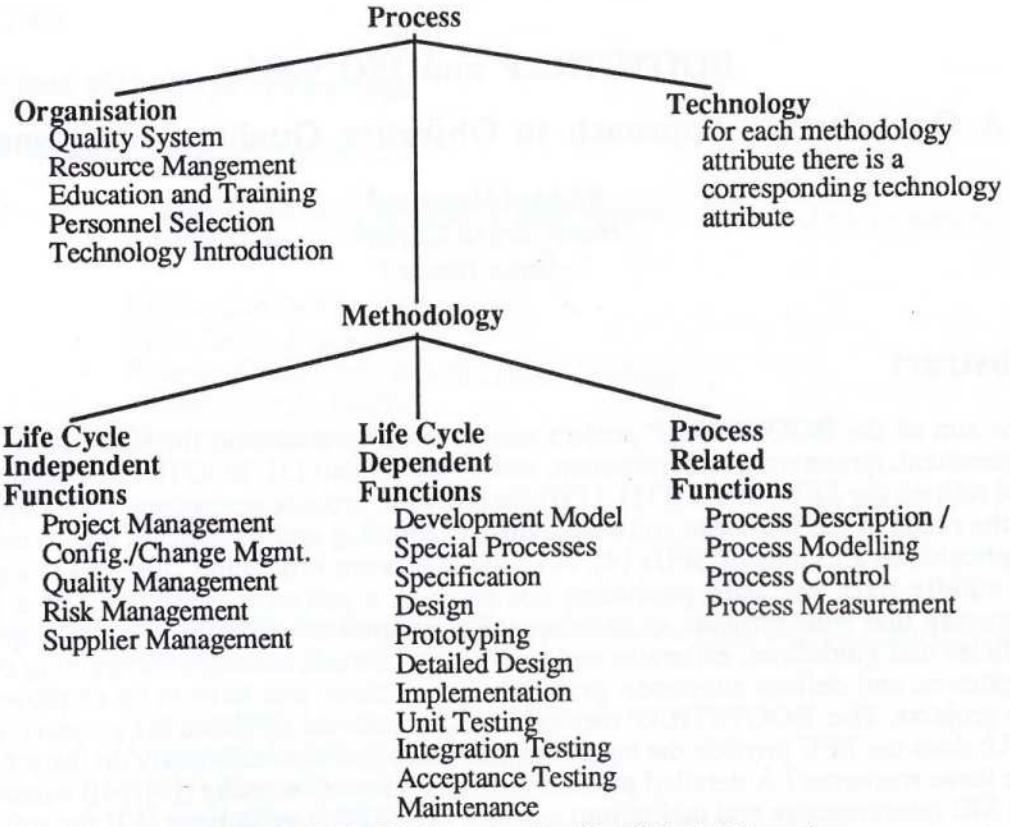


Fig. 1: BOOTSTRAP´s Process Quality Attribute Hierarchy

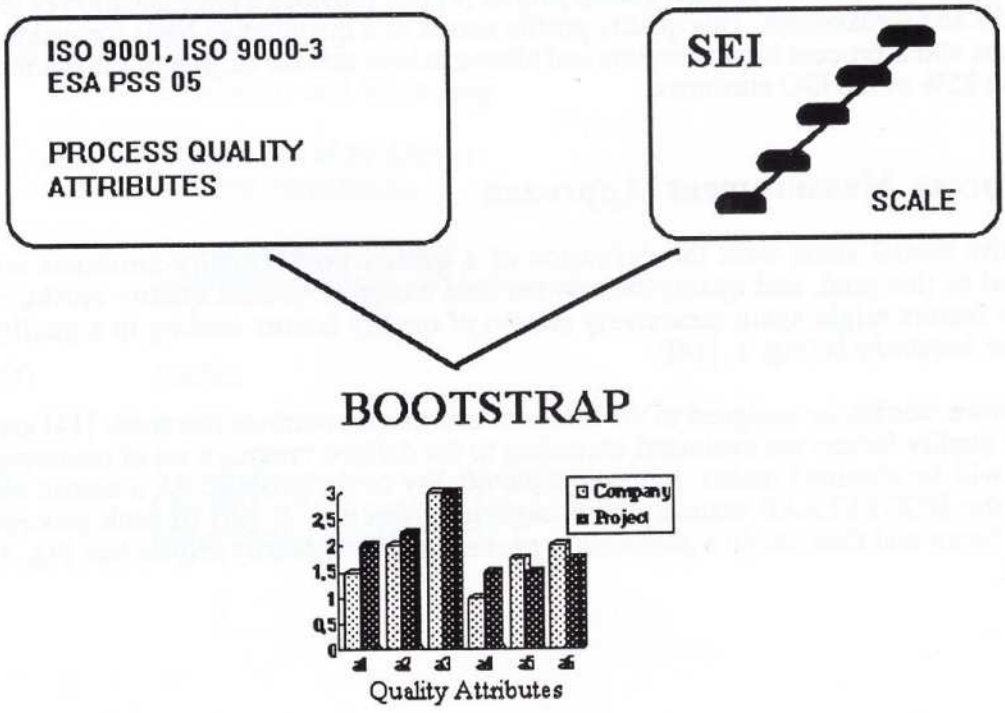


Fig. 2: BOOTSTRAP's Process Quality/Maturity Profile [4], [9]

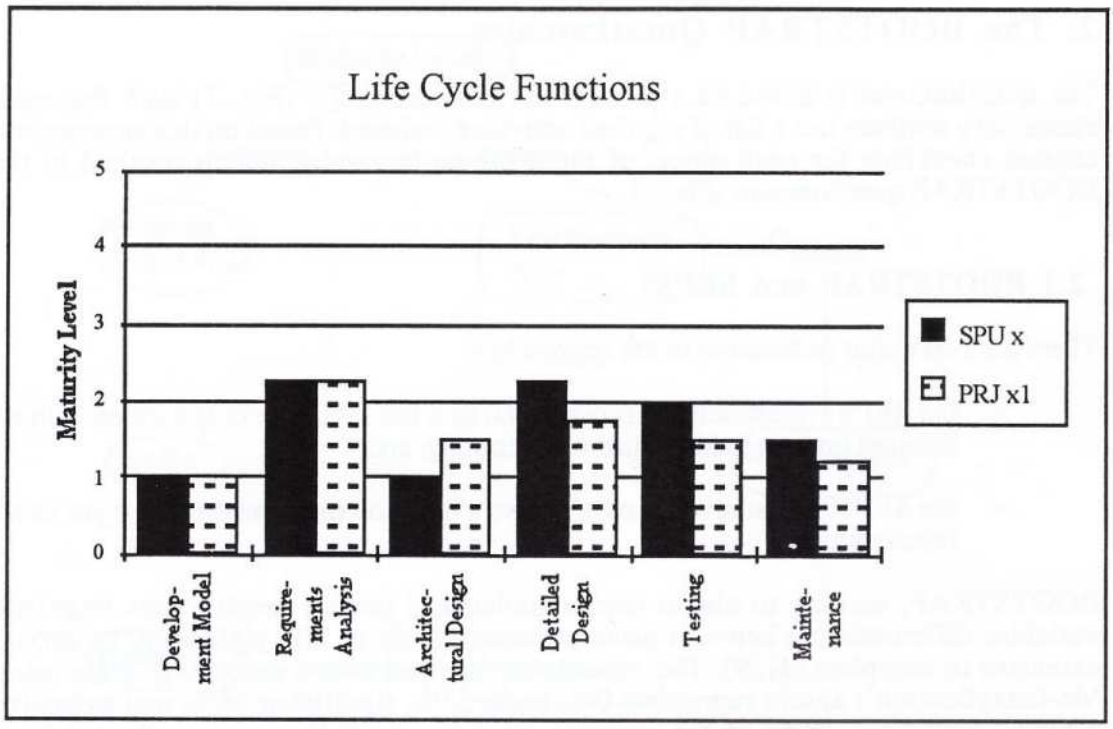


Fig. 3: Calculated Maturity Levels for Attribute Cluster "Life Cycle Functions" of SPU_X and its project PRJ_{X1}

The CMM model [13] of the SEI (Software Engineering Institute) differentiates between five different maturity levels of software processes: Initial Process (1), Repeatable Process (2), Defined Process (3), Managed Process (4), and Optimising Process (5). The SEI method is based on a questionnaire (see section 2.1) in which all questions are assigned to maturity levels from 2 to 5. After the assessment the completed questionnaire is mapped onto a maturity level scale [2] and the calculated maturity level of an SPU is regarded as an indicator of the SPU's process quality. However, a single maturity measure for an entire process does not sufficiently support a quantitative analysis of all the strengths and weaknesses ([2],[9]).

The ISO 9000-3 guidelines [12] define a number of process quality attributes that are essential for setting up a quality system within a software management and development organisation. The SEI'87 questionnaire has only six attributes concerning organisation and methodology. Taking into account the ISO 9000-3 guidelines and the ESA PSS 05 software engineering standards we identified about 25 additional attributes to be addressed by a number of additional of questions.

The ISO 9000-3 guidelines check, for instance, if reviews—joint reviews, design reviews, management reviews, progress control reviews—are conducted. However, the performance of a review presumes that a standard software engineering process is already in place so that the reviews can check if the defined methods and procedures are effectively used and followed within the projects. If there is, for example, no standard structure and format for design descriptions and if there is no checklist to investigate the design, the output of a design review will be a matter of discussion, but not the result of an objective evaluation. You first have to create a software engineering process model including standards and methods for both management and development before setting up a quality system that checks whether these standards and methods are effectively followed and employed. The ESA PSS 05 software engineering standards [6] describe such a software engineering process model, and the ISO 9000-3 guidelines specify the attributes required to set up a quality system.

2. The BOOTSTRAP Questionnaire

The questionnaire is based on a process attribute hierarchy (Fig. 1) such that each elementary attribute has a list of required activities assigned. Based on this structure we created checklists for each object of the attribute hierarchy, which resulted in the BOOTSTRAP questionnaire (Fig. 6).

2.1 BOOTSTRAP and SEI'87

There are two major differences in the approaches:

- the SEI'87 questionnaire is organised as a flat sequence of questions with no detailed process quality attribute hierarchy, and
- the SEI'87 questionnaire only allows a question to be answered by yes or no (black/white).

BOOTSTRAP, seeking to obtain more detailed and precise results, uses linguistic variables differentiating between weak or absent, basic or fair, significant or strong, extensive or complete [4],[9]. The answers are mapped onto a percentage scale using 'de-fuzzyfication': absent represents 0%, basic 33%, significant 66%, and extensive 100%, based on a study of the relevant interviewing techniques conducted by a psychologist in BOOTSTRAP. Some example questions are shown in (Fig. 4).

ATTRIBUTE: RISK MANAGEMENT		
Number	*Level*	*Text / Answers*
2219	3	Adoption of requirements to identify, assess, document risks to project and product associated with modifying SLC or non-SLC activities. Answers: absent/basic/significant/extensive
2220	2	Adoption of a requirement for identifying the parts of a specification more likely to show instability. Answers: absent/basic/significant/extensive
.....	.....	.....

Fig. 4: Sample BOOTSTRAP Questions

2.2 BOOTSTRAP and CMM (Capability Maturity Model, 1991)

CMM [13] is a comprehensive framework which provides guidelines for improvements recommended for software organisations wanting to increase their software process capability. CMM describes the five maturity levels of software processes, for each of which a number of key process areas are defined. Each key process area contains a number of key practices that have to be performed. A key practice specifies a key indicator which directly relates to at least one question of the SEI questionnaire (see Fig. 5). Thus CMM provides useful guidelines for any organisation wanting to find out what has to be done to reach the next higher level of maturity.

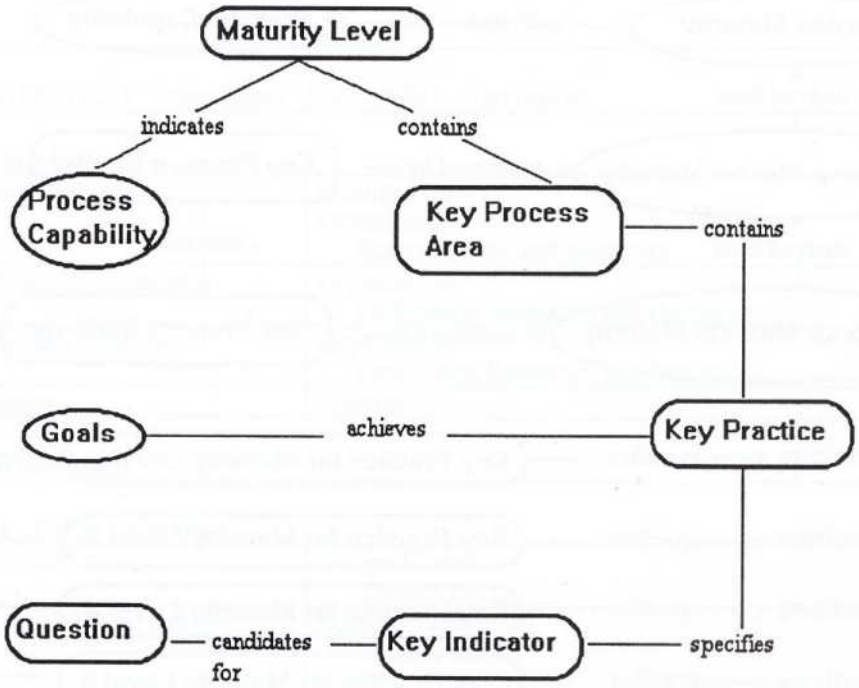


Fig.5: The CMM Structure

CMM's Structure

CMM defines different key process areas for each maturity level. The disjunction of the sets of attributes belonging to different levels is always empty. Each CMM key process area—corresponding to an attribute—is valid only within one maturity level. Thus it is not possible to map a single CMM attribute onto a maturity level scale from 2 to 5. This means that the whole improvement approach is maturity level and not attribute based (Fig. 5), because CMM's structure supports the calculation of only one maturity measure for the entire process. However, an attribute based evaluation is more detailed in its approach and provides a clearer understanding of strengths and weaknesses.

BOOTSTRAP's Structure

The BOOTSTRAP method ([4], [9]) evaluates each attribute separately on a five point maturity scale. For each attribute the BOOTSTRAP assessment team checks a number of activities that have to be performed on levels 2 to 5 (Fig. 6). Thus BOOTSTRAP can concentrate on single attributes, evaluate each attribute on a five point maturity level scale, identify weaknesses and strengths, and establish attribute based improvement plans (Fig. 3).

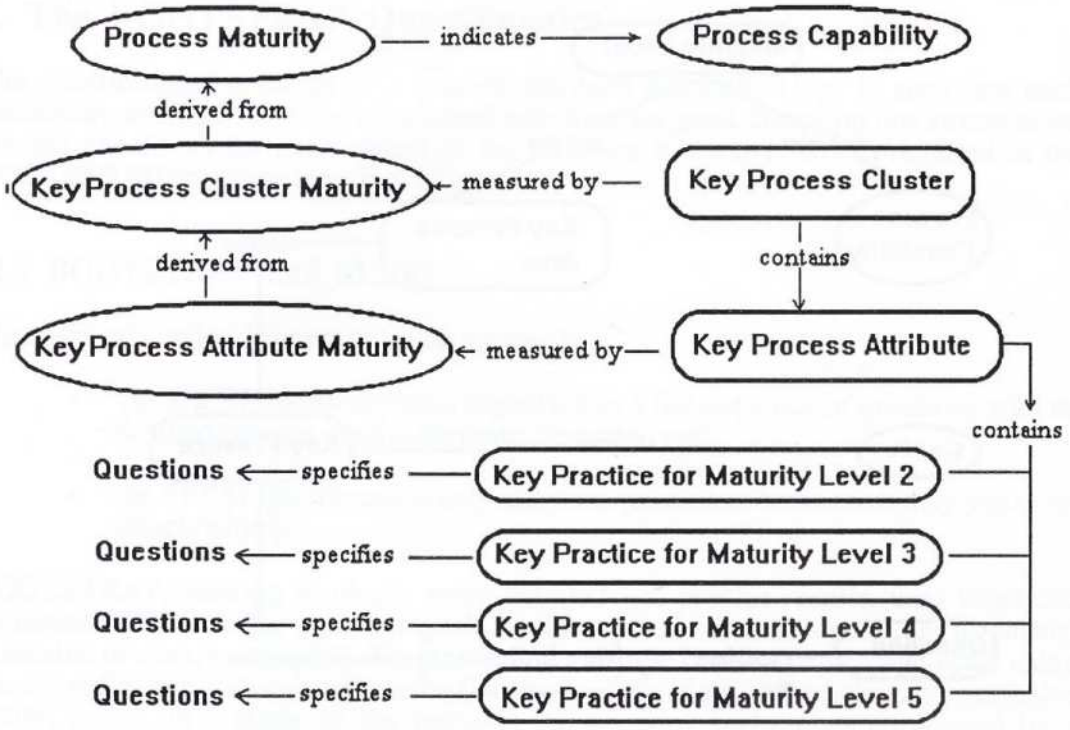


Fig. 6: BOOTSTRAP's Attribute Based Structure

2.3 BOOTSTRAP and ISO 9000 [12]

The ISO 9000 standard for quality management describes a number of main process quality attributes such as *quality policy, strategic planning, resource allocation, quality planning, quality control, and quality assurance*. For each quality attribute an organisation has to demonstrate that a method is used and that all projects employ this method. Additionally ISO 9000 provides a selection procedure for choosing the appropriate *quality system*: ISO 9001 (Design, Development, Production, Installation, Servicing); ISO 9000-3 (Guidelines for the Application of ISO 9001 to the Development, Supply and Maintenance of Software).

We analysed the attribute structures of ISO 9001 and ISO 9000-3 and assigned all BOOTSTRAP questions to ISO attributes, deriving information about the comparability of BOOTSTRAP and ISO. As a result we can determine for about 85% of the ISO 9001 and the ISO 9000-3 attributes whether or not they are satisfied.

In Tab. 1 we have mapped all ISO 9001 attributes onto BOOTSTRAP's attribute clusters as presented in Fig. 1. For instance, the ISO 9001 attributes *quality policy* and *responsibility and authority* concerning the organisation relate to BOOTSTRAP's attribute *quality system*. And the ISO 9001 attributes *verification resources and personnel* and *training* relate to BOOTSTRAP's attributes *personnel selection* and *training*. Additionally we assigned all BOOTSTRAP questions to ISO attributes and thus obtained a coverage measure (Tab. 1).

BOOTSTRAP Attributes	ISO 9001 Attributes	% Coverage
Organisation		
Quality System	Quality Policy	30%
	Organisation Responsibility and Authority	80%
Resource Management	Organisation Verification Resources and Personnel	100%
Personnel Selection	Organisation Verification Resources and Personnel	100%
Training	Training	100%
Methodology		
Life Cycle Functions		
Specification	Design Control Design Input	100%
Design, Detailed Design	Design Control Design Output	100%
	Design Verification	100%
Unit Testing	Inspection and Testing Receiving Inspection & Test	100%
	Inspection & Test Records	100%
Integration Testing	Inspection and Testing In-Process Inspection & Test	100%
	Inspection & Test Records	100%
Acceptance Testing	Inspection and Testing Final Inspection & Test	100%
	Inspection & Test Records	100%
Transfer	Handling, Storage, Packaging and Delivery Handling, Storage, Packaging	0%
	Delivery	100%
Maintenance	Servicing	70%
Life Cycle Independent Functions		
Quality Management	Contract Review	50%
	Management Responsibility Management Review	100%
Project Management	Design Control Design and Development Planning	90%
Configuration & Change Mgmt.	Design Control Design Changes	100%
	Document Control Document Issue	10%
	Document Modifications	100%
	Product Identification & Traceability	100%
	Inspection & Test Status	0%
Supplier Management	Purchasing Assessment of Subcontractors	100%
	Purchasing data	100%
	Verification of Purchased Product	100%
	Purchaser Supplied Product	0%

Tab. 1: ISO 9001 Attributes Mapped onto BOOTSTRAP's Attribute Clusters & Percent Coverage of ISO 9001 Attributes (to be continued)

BOOTSTRAP Attributes	ISO 9001 Attributes	% Coverage
Process Related Functions		
Process Control	Process Control	
	General Processes	100%
	Special Processes	100%
	Control of Non conforming Product	30%
	Corrective Actions	100%
Process Measurement	Quality Records	100%
	Internal Quality Audits	100%
	Statistical Techniques	100%

Tab. 1: ISO 9001 Attributes Mapped onto BOOTSTRAP's Attribute Clusters & Percent Coverage of ISO 9001 Attributes (continued)

The assignment of BOOTSTRAP questions to ISO 9001 attributes were also based on the ISO 9000-3 guidelines to be able to correctly interpret the ISO 9001 attributes. BOOTSTRAP checks, for example, if contract reviews are performed and if quality requirements are defined, but it currently does not specifically investigate if the customer is integrated into this review procedure, although the customer's integration into reviews is addressed by the ISO 9000-3 attributes *joint reviews*, and *mutual co-operation*. Thus the coverage measure concerning contract reviews (Tab. 1) we obtained a value of less than 100%. However, for most of the attributes (e.g. life cycle functions, process related functions) BOOTSTRAP checks more methods and activities than it is required for the ISO 9001 certification. Moreover BOOTSTRAP provides attributes such as *risk management* which are not covered by ISO.

3. BOOTSTRAP's Evaluation Method

A mathematical description and analysis of the algorithm was presented at the SPSE'92 conference in Klagenfurt, Austria and is discussed in [4], [9]. The following summarises a few main points.

Four Point Fuzzy Reply Set

Each question is answered on a four point scale of fuzzy terms that we use instead of a yes/no option (section 2.1).

Key Attributes

The SEI algorithm uses key questions which have to be satisfied to reach a certain level [2]. BOOTSTRAP does not use single questions but key clusters of questions (key attributes). Quality management, for example, is a key attribute evaluated by 10 questions that have to be satisfied by a threshold percentage to fulfil a certain level. This attribute based evaluation allows to identify weak and strong process attributes and to establish attribute based improvement plans.

Innovations

The SEI algorithm is strictly sequential. Only if level i is satisfied by a minimum of about 80 percent, and if nearly all key questions on level i are answered by yes, does

the SEI algorithm take into account the scores on level i+1 [2]. This does not favour SPUs and projects which plan and phase improvement over a period of time. BOOTSTRAP takes into account scores which the SPU or project gained on the next higher level.

Dynamic Step Scale

If the evaluation is only based on percentages, then it would appear as if there were equal distances between the levels of the maturity scale, although there are different numbers of questions for each level. This holds for SEI and BOOTSTRAP and is due to the fact that only few SPUs on levels 4 and 5 have been found and characterised so far. BOOTSTRAP treats the distances between the levels as variable—expressed by the number of questions associated with each level.

This 'step scale' ensures that we do not obtain a too high a maturity level when taking into account the scores on higher levels.

4. Quality Profiles [4], [9] and Action Plans [3]

Maturity level 2 means that effective methods are in place, level 3 means that the most effective methods are documented and standardised across all projects, level 4 means that the efficiency of these methods, the productivity of major process steps and the quality of the products are quantitatively measured, and level 5 means that the quantitative feedback is analysed and action plans to improve the process are established.

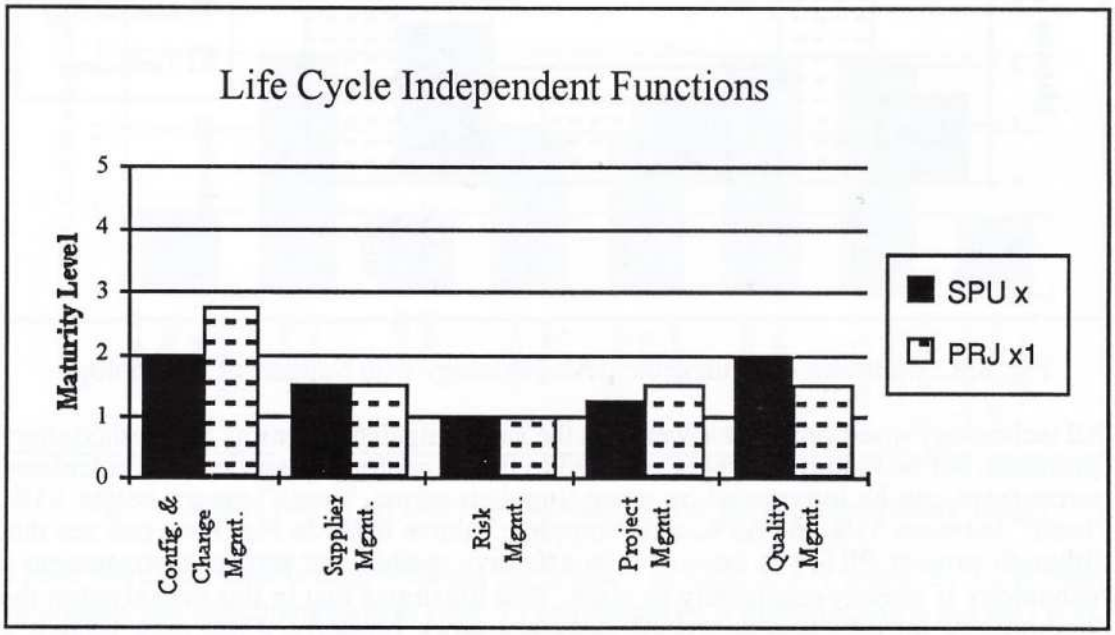


Fig. 7: Sample Part of a Process Quality Profile of SPU_X and its Project PRJ_{X1}

All quarters (e.g. 1.75, 2.25, 3.5) can be interpreted in linguistic terms: the method employed is weak at 1.25, basic at 1.5, significant at 1.75, and effectively employed at 2; the method is effectively used but weakly documented and standardised at 2.25, basically documented and standardised at 2.5, etc.

Fig. 7 represents a part of BOOTSTRAP process quality profiles of an anonymous SPU_X and one of its projects PRJ_{X1}. SPU_X does not provide a method for risk management and therefore project PRJ_{X1} lacks a method. Concerning project management the SPU only provides a very weak method and the project does not use an effective method either. In quality management the SPU provides an effective method, but the resources could be better exploited by the project. SPU_X provides an effective method for configuration management and project PRJ_{X1} is efficiently using this method as a standard. SPU_X has so far not standardised the methods for quality management and configuration management across all projects (no value above 2).

The SEI method rates technology with A (low) and B (high). However, where is the difference if an SPU has fulfilled 49% instead of 51% of the technology questions? In the first case we would achieve technology level A, in the second technology level B. We tried to solve this problem by detailed comparison of methodology and technology (Fig. 8). For each methodology attribute there is a technology attribute. Each methodology attribute is mapped onto a maturity level scale (see left hand scale of Fig. 8) whereas the technology attribute is mapped onto a percentage scale (see right hand scale of Fig. 8).

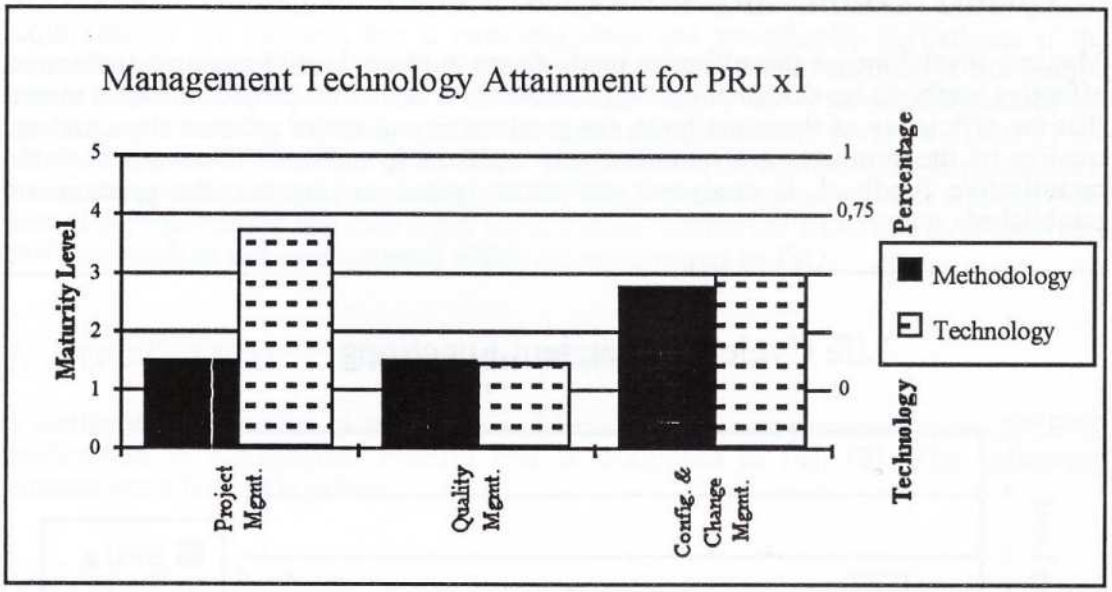


Fig. 8: Comparison of Management Methodology with Support of Technology

All technology questions are answered in the same linguistic terms as the methodology questions, but no maturity level is assigned to the technology questions. The calculated percentages can be interpreted by using linguistic terms, "weak" means below 33%, "basic" between 33% and 66%, and "complete" above 66%. In Fig. 8 we can see that although project PRJ_{X1} is not using an effective method for project management a technology is already completely in place. This illustrates that in this organisation the "technology introduction function" does not work properly. First they bought a technology and then they started to learn the associated methodology. And if we look at the project management maturity of the same SPU (Fig. 7), which is about 1.25, we see that the SPU bought the project management tool without knowing anything about project management methods and procedures. Concerning configuration and change management project PRJ_{X1} uses an effective method which is basically supported by technology (Fig. 8).

4.1 ISO Certification Profiles

For about 85% of the ISO 9001 and ISO 9000-3 attributes we can evaluate whether or not they are satisfied. We are currently developing prototype tools that can calculate a certification level for 85% of the ISO attributes. For each ISO attribute we calculate a certification level on a five point scale:

0 for this attribute we cannot calculate a certification level

1 failed the certification

2 will pass the certification with modifications in the process

3 passed the certification

4 expert fulfilling additional BOOTSTRAP issues (note: maturity level 3 is the goal of an ISO 9001 certification [5]).

A definition of the algorithm used to calculate ISO certification profiles is given in the appendix. Applying this transformation to the data of the anonymous SPU_X, we obtain the ISO 9001 certification profile shown below in Fig. 9.

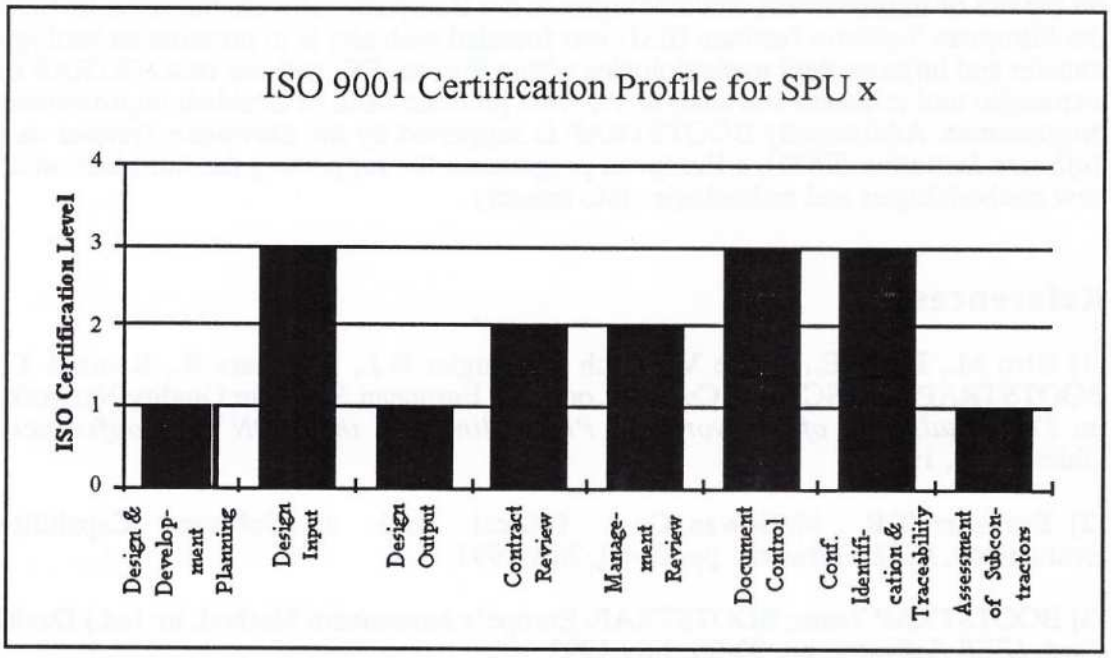


Fig. 9: Sample Part of an ISO Certification Profile for the anonymous SPU_X

There is a direct relationship between the profiles in Figs. 3, 7, 7 and 9. SPU_X is weak in project and risk management (Fig. 7) and therefore it does not fulfil the ISO 9001 attribute *design and development planning*. SPU_X provides an effective method for requirements analysis (Fig. 3) but does not provide a method for performing the design based on the specified requirements. Thus the SPU fulfils the ISO 9001 requirement *design input,* but fails in the attribute *design output*.. SPU_X provides an effective method for quality management (Fig. 7) including the performance of reviews, but does not standardise and document this method across all projects. Thus the ISO 9001 attributes *contract review* and *management review* are on certification level 2 and by standardising the most effective methods for all projects the certification level 3 could be achieved. SPU_X provides an effective method for configuration management (Fig. 7)

and project PRJ_{X1} is efficiently using this method as a standard. Thus the ISO 9001 attributes *document control* and *configuration identification and traceability* are satisfied. Concerning supplier management SPU_X is not efficient (Fig. 7) and fails the ISO 9001 attribute *assessment of subcontractors*.

5. Conclusion and Future Outlook

The field testing and benchmarking for BOOTSTRAP was provided by Robert BOSCH GmbH of Germany, a corporation in the electronics and telecommunications sector with about 180,000 employees. Additionally we have gathered considerable experience in the assessment and improvement of software divisions of leading banks, telecommunications companies, administrative institutions and small or middle sized software companies.

The *BOOTSTRAP method* is based on practical experience and feedback from customers. The recently founded *BOOTSTRAP Institute* will ensure that the method is maintained and enhanced, and its consultants are continuously trained. The BOOTSTRAP Institute actively contributes to the SPICE initiative.

The co-operation with ESPRIT projects in the software metrics field through the *International Software Consulting Network* (ISCoN) [1] will have a multiplying effect on the use of improvement methodologies in the European software industry. In 1993 the *European Software Institute* (ESI) was founded with aim is to promote technology transfer and improvement methodologies within Europe. ESI will use BOOTSTRAP as a strategic tool to assess and analyse software processes and to establish improvement programmes. Additionally BOOTSTRAP is supported by the *European Systems and Software Initiative* (ESSI), a European programme for supporting the introduction of new methodologies and technologies into industry.

References

[1] Biro M., Feuer E., Haase V., Koch G., Kugler H.J., Messnarz R., Remszö T., BOOTSTRAP and ISCN - A Current Look at a European Software Quality Network, in: *The Challenge of Networking, Proceedings of the CON'93 Conference*, Oldenbourg, 1993

[2] Bollinger T.B. , McGowan C.: A Critical Look at Software Capability Evaluations, *IEEE Software*, pp. 25-41, July 1991

[3] BOOTSTRAP Team, BOOTSTRAP: Europe's Assessment Method, in: (ed.) David Card, *IEEE Software*, pp. 93-95, July 1993

[4] Cachia R. M., Maiocchi M., *Middle Management Briefing*, Deliverables 10 and 20, BOOTSTRAP ESPRIT 5441, Commission of European Communities, 1992

[5] Coallier F., Canada B., How ISO 9001 Fits Into the Software World, *IEEE Software*, pp. 98-100, January 1994

[6] ESA Board for Standardisation and Control, *ESA PSS 05 Software Engineering Standards*, European Space Agency, Paris, 1991

[7] Grady R. B., *Practical Software Metrics for Project Management and Process Improvement*, Prentice Hall, Englewood Cliffs, London, Sydney, Tokyo, 1992

[8] Grady R. B., Caswell D. C., *Software Metrics: Establishing a Company-Wide Program*, Prentice Hall, Englewood Cliffs, New Jersey, 1986

[9] Haase V., Messnarz R., Cachia R. M., Software Process Improvement by Measurement, in: (ed.) Mittermeir R., *Shifting Paradigms in Software Engineering*, Springer Verlag, Wien, New York, September 1992

[10] Huber A., A Better Way to Represent BOOTSTRAP Data, *IEEE Software*, p. 10, September 1993

[11] Humphrey W. S., *Managing the Software Process*, (ed.) Software Engineering Institute (USA), Addison-Wesley Publishing Company, New York, Wokingham, Amsterdam, Bonn, Madrid, Tokyo, 1989

[12] ISO 9000-3, *Quality Management and Quality Assurance Standards*, Part 3: Guidelines for the Application of ISO 9001 to the Development, Supply, and Maintenance of Software, 1991

[13] Paul M. C., Curtis B., Chrissis M. B., *Capability Maturity Model for Software*, (ed.) Software Engineering Institute (USA), Carnegie Mellon University, Pittsburgh, 1991

[14] Schneidewind N. F., *Standard for a Software Quality Metrics Methodology*, Unapproved Draft, IEEE Computer Society, September 1991

Appendix: Calculation of ISO Certification Profiles

The definition of the algorithm is as follows:

Q	...	Set of all BOOTSTRAP questions
$L = \{1..5\}$	...	Maturity levels assigned to questions
$S = \{0, 33, 66, 100\}$	...	Scoring of questions
$B_Q \subseteq Q \times L \times S$	...	Completed BOOTSTRAP Questionnaire
$q \varepsilon B_Q$	...	Element of B_Q
I	...	Set of all ISO 9001/9000-3 attributes
$i \varepsilon I$	...	Represents an ISO 9001/9000-3 attribute
$P(I)$	...	Set of all possible subsets of I

F is a function which describes the mapping of BOOTSTRAP questions onto ISO attributes. The result of F(q) is a subset V of I because one and the same BOOTSTRAP question sometimes relates to more than one ISO attribute (section 2.3).

$$F: B_Q \Rightarrow P(I), \quad F(q) = V, \text{ with } V \subseteq I$$

We differentiate between three different categories of questions:

$C(q) = C_1$ ... a question which is directly related to an ISO attribute and has to be satisfied to fulfil the ISO certification requirements;

$C(q) = C_2$ ... a question which covers additional issues that are checked by BOOTSTRAP but are not required for the ISO certification

$C(q) = C_3$ ... a question which is related to an ISO attribute that is not checked effectively enough by BOOTSTRAP.

E is an evaluation function which maps a set of n ISO attributes $\{i_1, i_2, .. , i_n\}$ onto a set of pairs $\{(i_1, v_1), (i_2, v_2), .. , (i_n, v_n)\}$ with $v_i \varepsilon I_L = \{0..4\}$ and I_L representing the ISO certification levels.

$A \varepsilon P(I)$ ... The set of those ISO attributes for which we can calculate a certification value

Q_i ... All $q \varepsilon B_Q$ with $i \varepsilon F(q)$

E is then defined as follows:

$$E: P(I) \Rightarrow P(I \times I_L)$$

$$E(A) = \{(i_1, v_1), (i_2, v_2), .. , (i_k, v_k)\}, \text{ with}$$

$$i_m \varepsilon A, v_m \varepsilon I_L - \{0\}, \text{ for } m = 1..k, \text{ and } |A| = k, |I| = n$$

and

$$E(A) = \{(i_{k+1}, v_{k+1}), (i_{k+2}, v_{k+2}), .. , (i_n, v_n)\}, \text{ with}$$

$i_m \varepsilon A, v_m = 0$, for $m = k+1..n$

where:

$E(i) = (i,1)$ if there are questions $q \varepsilon Q_i$ with $C(q)=C_1$ which were answered with less than *significant* and the total satisfaction percentage for C_1 questions is lower than 50%

$E(i) = (i,2)$ if there are questions $q \varepsilon Q_i$ with $C(q) = C_1$ which were answered with less than *significant* and the total satisfaction percentage for C_1 questions is greater than 50%

$E(i) = (i,3)$ if each question $q \varepsilon Q_i$ of category C_1 was answered with *significant* or *extensive* and the total satisfaction percentage for questions of category C_2 is below 50%

$E(i) = (i,4)$ if each question $q \varepsilon Q_i$ of category C_1 was answered with significant or extensive and the total satisfaction percentage for questions of category C_2 is greater than 50%

Questions with C(q)=C3 are related to those attributes for which we cannot calculate a certification level. Using this first version of a transformation algorithm and based on the data of the anonymised SPU_X we obtain the ISO 9001 certification profile shown in Fig. 9.

BOOTSTRAP's Goals

1. Design and Development of an Assessment Method Providing

* an assessment process model

* a questionnaire taking into account European approaches and standards (ISO 9000, ESA PSS 05)

* a detailed quantitative feedback about strengths and weaknesses (quality/maturity profiles)

* guidelines for action plan generation

2. Design and Development of a European Database

* storing all general data and quality/maturity profiles

* allowing to perform evaluations for certain market segments, sizes, platforms, etc.

* providing information about the state of the art

3. Establishment of an Institute (BI) responsible

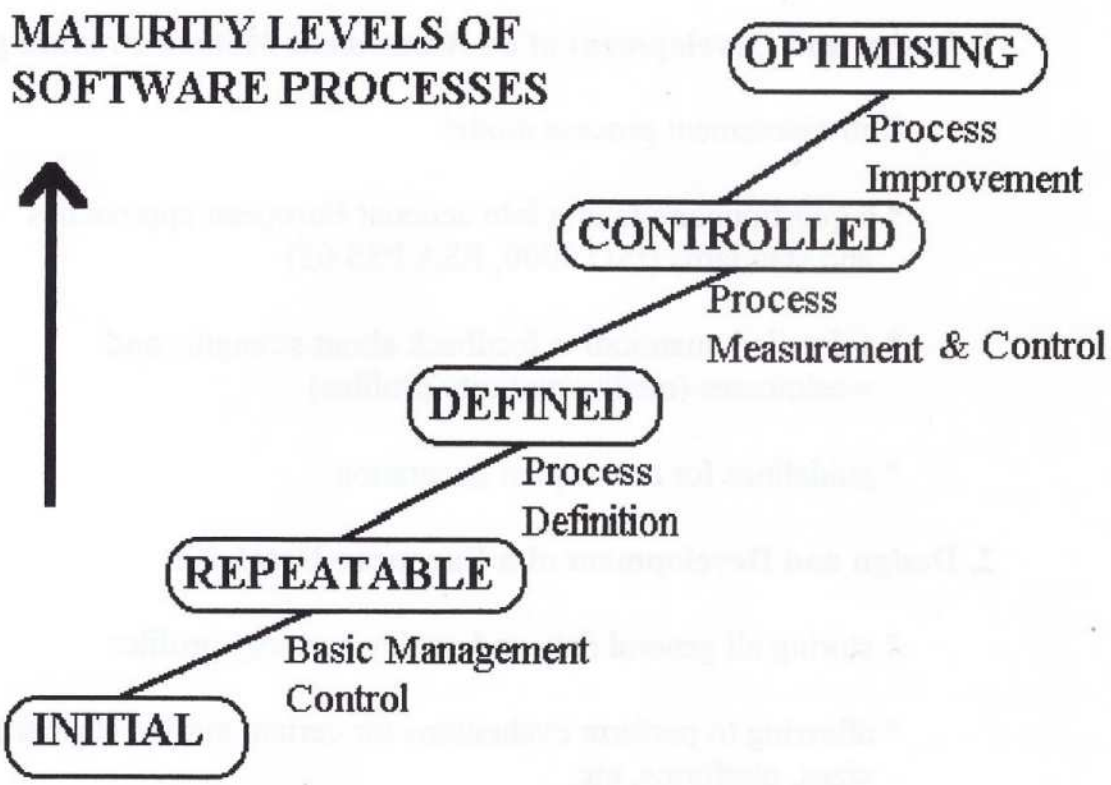
* for continuing research on software process evaluation

* for managing database service

* for training BOOTSTRAP assessors

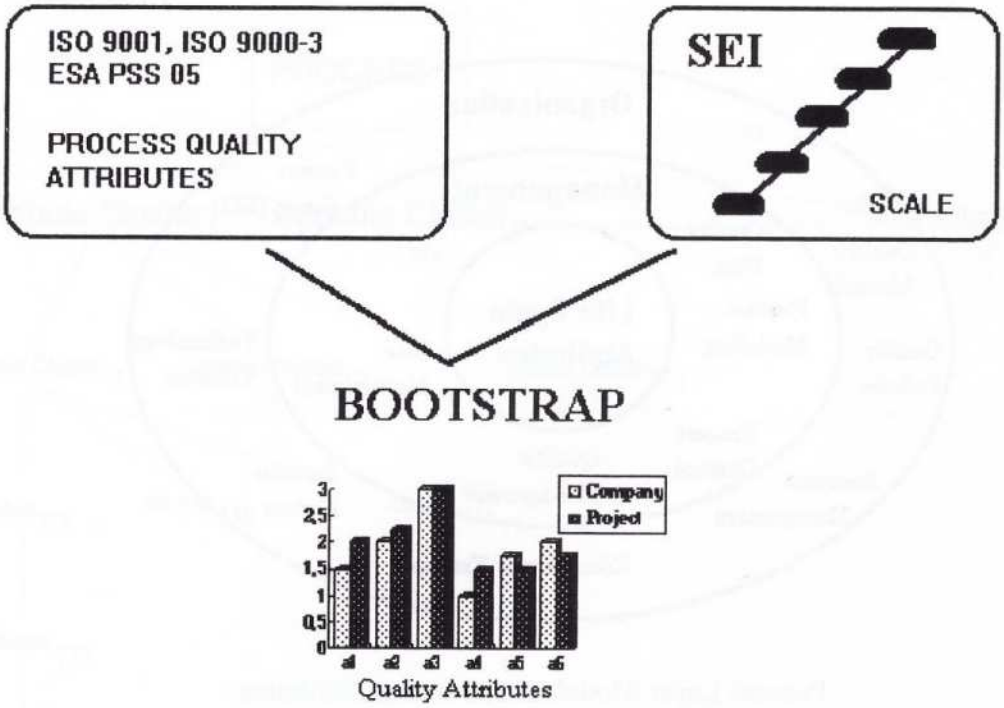
* for managing the licensing policy

BOOTSTRAP ESPRIT 5441: A Quantitative Approach to Objective Quality Management

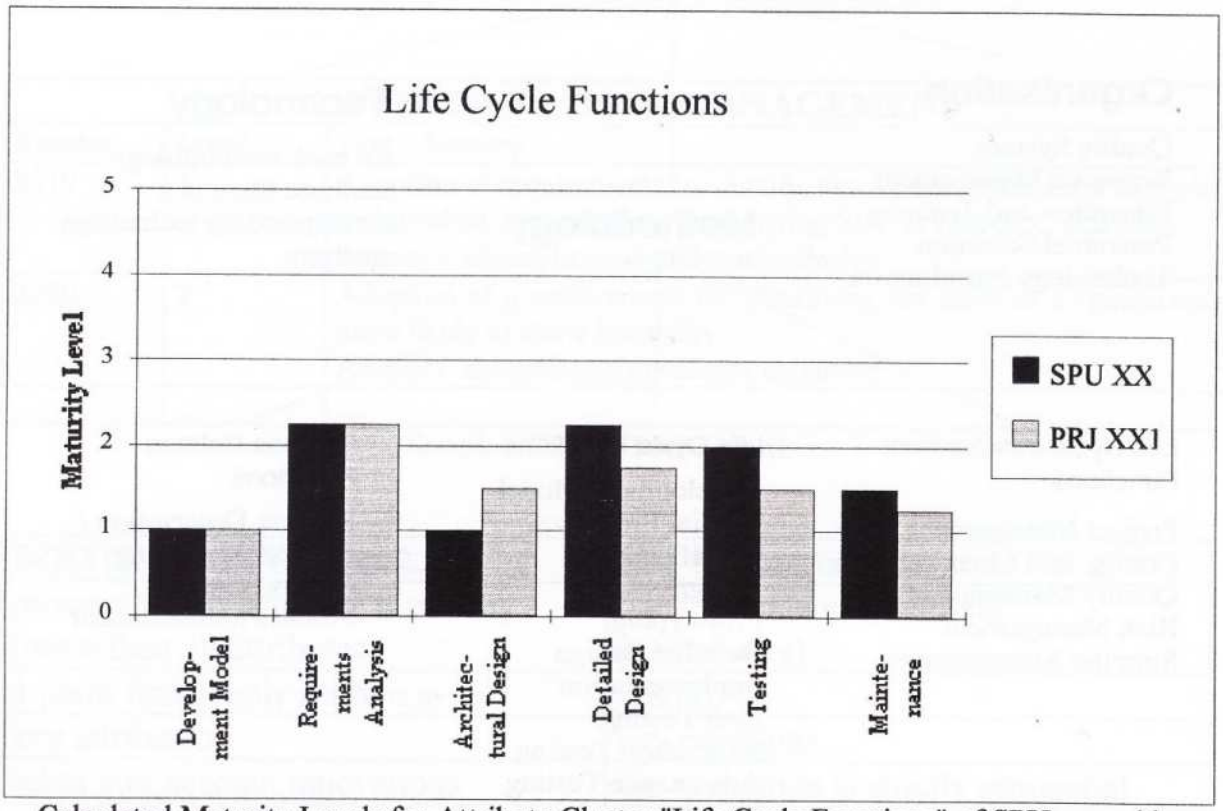


Authors: R. Messnarz, H.J. Kugler, V. Haase

BOOTSTRAP ESPRIT 5441: A Quantitative Approach to Objective Quality Management



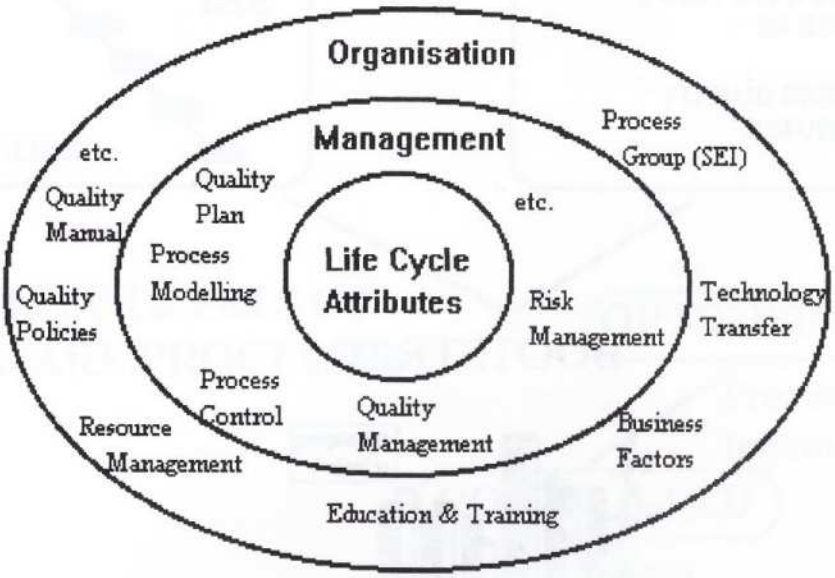
BOOTSTRAP´s Process Quality/Maturity Profile



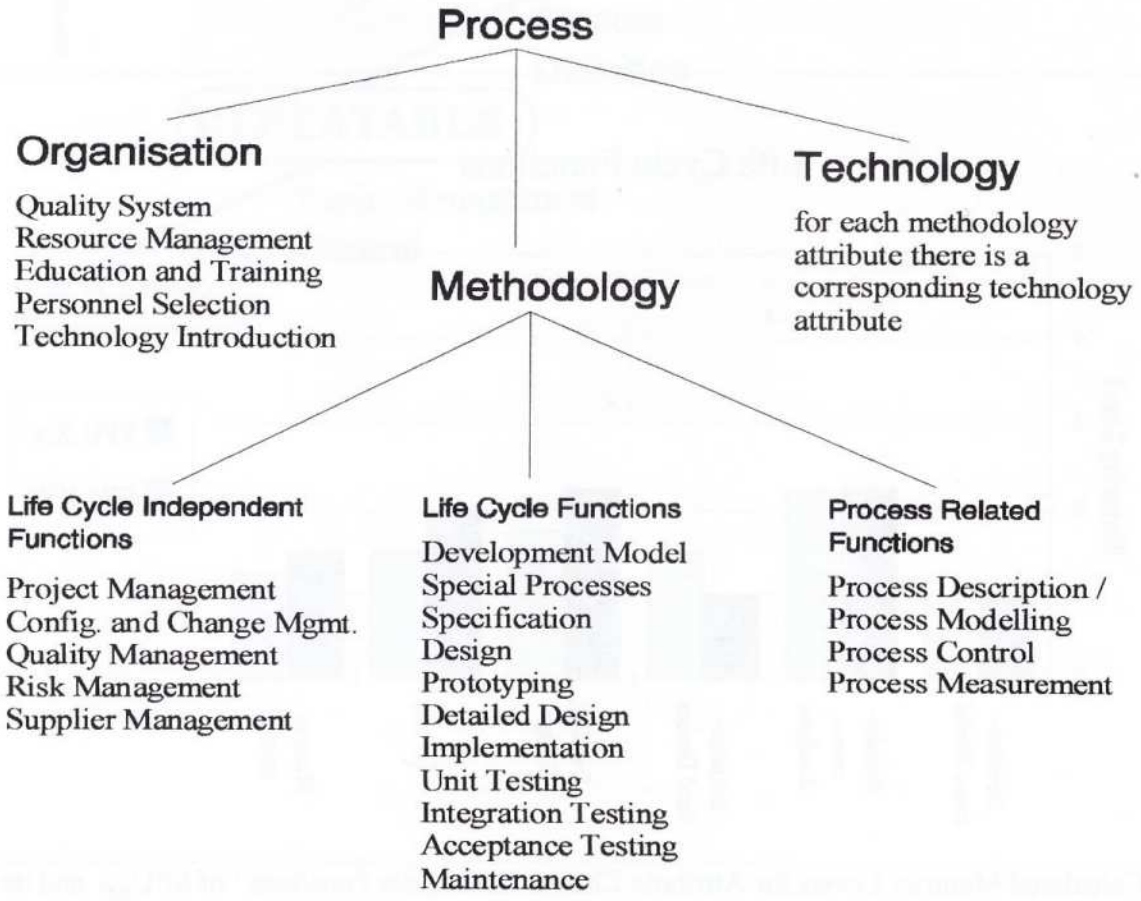
Calculated Maturity Levels for Attribute Cluster "Life Cycle Functions" of SPU_{XX} and its project PRJ_{XX1}

Authors: R. Messnarz, H.J. Kugler, V. Haase

BOOTSTRAP ESPRIT 5441: A Quantitative Approach to Objective Quality Management



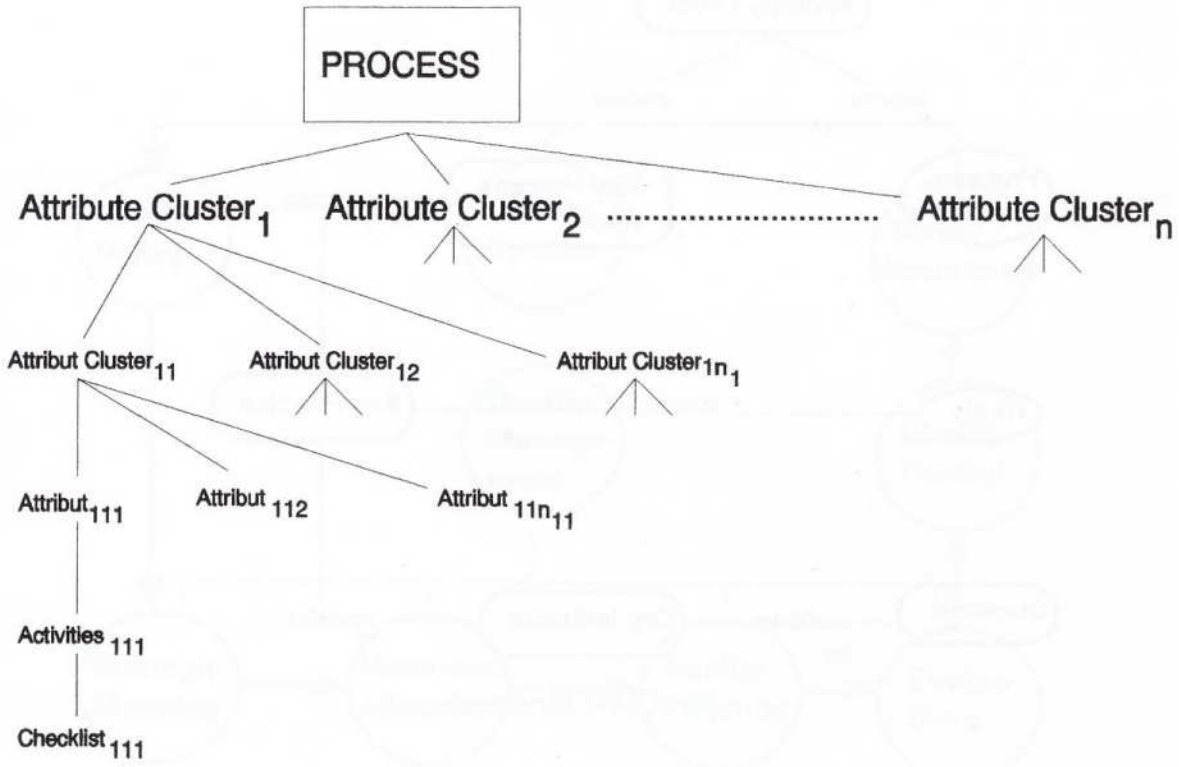
Process Layer Model / Key Process Attributes



BOOTSTRAP´s Process Quality Attribute Hierarchy

Authors: R. Messnarz, H.J. Kugler, V. Haase

BOOTSTRAP ESPRIT 5441: A Quantitative Approach to Objective Quality Management



Attribute Based Software Process Model

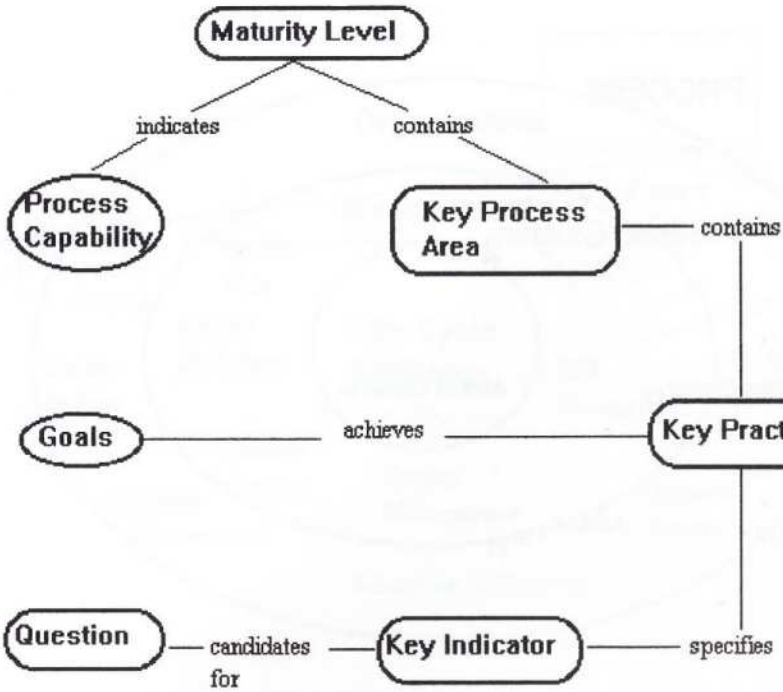
ATTRIBUTE: RISK MANAGEMENT		
Number	*Level*	*Text / Answers*
2219	3	Adoption of requirements to identify, assess, document risks to project and product associated with modifying SLC or non-SLC activities. Answers: absent/basic/significant/extensive
2220	2	Adoption of a requirement for identifying the parts of a specification more likely to show instability. Answers: absent/basic/significant/extensive
.....	.....	.....

Sample BOOTSTRAP Questions

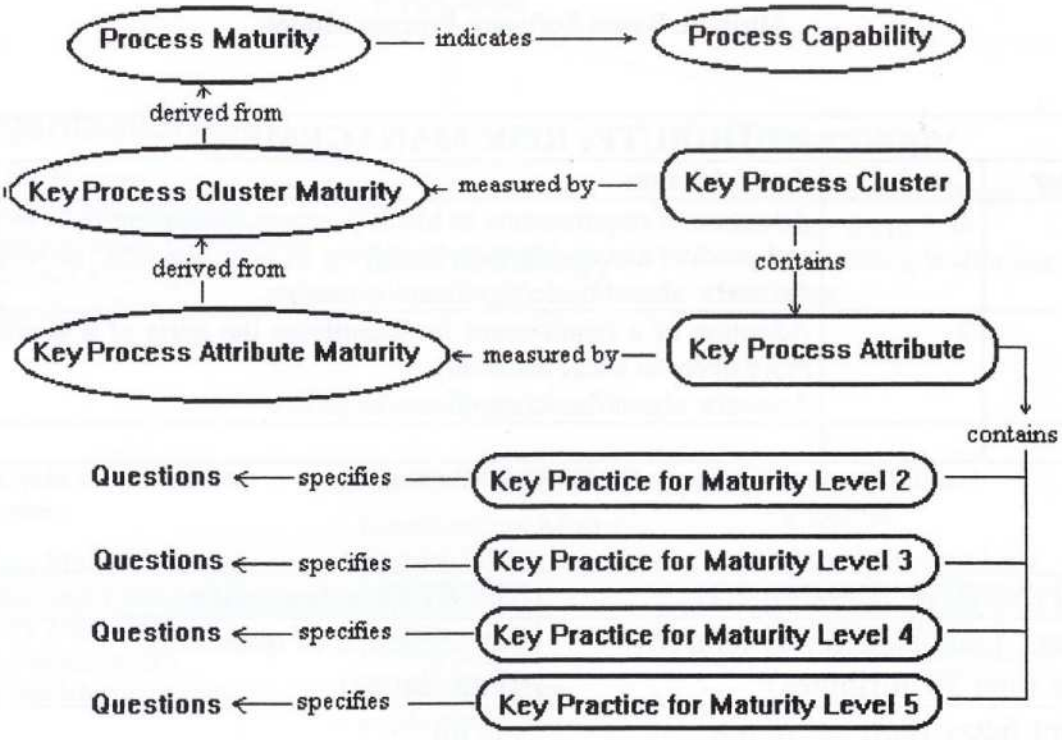
BOOTSTRAP Version 2.22	SEI´87 Questionnaire
process quality attribute hierarchy (more than 30 attributes)	flat sequence of questions (6 attributes)
4 point fuzzy reply set & n.a.	yes/no
key attributes	key questions
takes into account innovations	the evaluation is strictly sequential
attributes are as much as possible independent from each other	dependencies between the attributes

Authors: R. Messnarz, H.J. Kugler, V. Haase

BOOTSTRAP ESPRIT 5441: A Quantitative Approach to Objective Quality Management



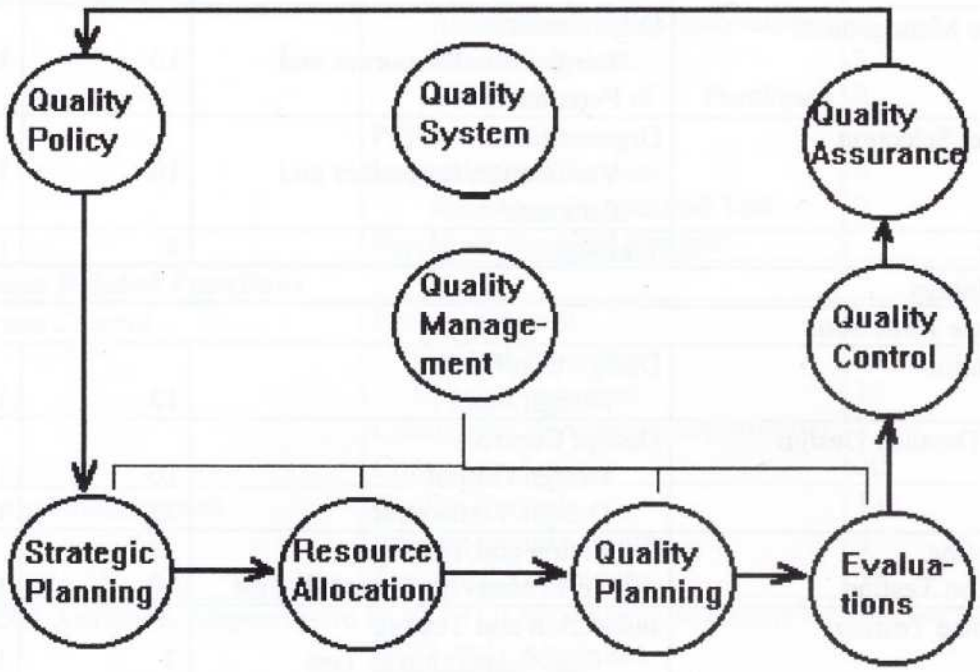
The CMM Structure



BOOTSTRAP´s Attribute Based Structure

Authors: R. Messnarz, H.J. Kugler, V. Haase

BOOTSTRAP ESPRIT 5441: A Quantitative Approach to Objective Quality Management



The ISO 9000 Improvement Cycle

Quality System	Application Area
ISO 9001	Design, Development, Production, Installation, Servicing
ISO 9000-3	Guidelines for the Application of ISO 9001 to the Development, Supply and Maintenance of Software

ISO Quality Systems Applicable for Software Management & Development

BOOTSTRAP ESPRIT 5441: A Quantitative Approach to Objective Quality Management

BOOTSTRAP Attributes	ISO 9001 Attributes	#questions	%Coverage
Organisation			
Quality System	Quality Policy	2	30%
	Organisation Responsibility and Authority	15	80%
Resource Management	Organisation Verification Resources and Personnel	10	100%
Personnel Selection	Organisation Verification Resources and Personnel	10	100%
Training	Training	4	100%
Methodology			
Life Cycle Functions			
Specification	Design Control Design Input	12	100%
Design, Detailed Design	Design Control		
	Design Output	10	100%
	Design Verification	25	100%
Unit Testing Integration Testing	Inspection and Testing In-Process Inspection & Test	10	100%
Acceptance Testing	Inspection and Testing		
	Final Inspection & Test	7	100%
	Inspection & Test Records	4	100%
Transfer	Handling, Storage, Packaging and Delivery		
	Handling, Storage, Packaging	0	0%
	Delivery	2	100%
Maintenance	Servicing	13	70%
Life Cycle Independent Functions			
Quality Management	Contract Review	4	50%
	Management Responsibility Management Review	5	100%
Project Management	Design Control Design and Development Planning	34	90%
Configuration & Change Mgmt.	Design Control Design Changes	10	100%
	Document Control		
	Document Issue	2	10%
	Document Changes	10	100%
	Product Identification & Traceability	18	100%
	Inspection & Test Status	2	0%

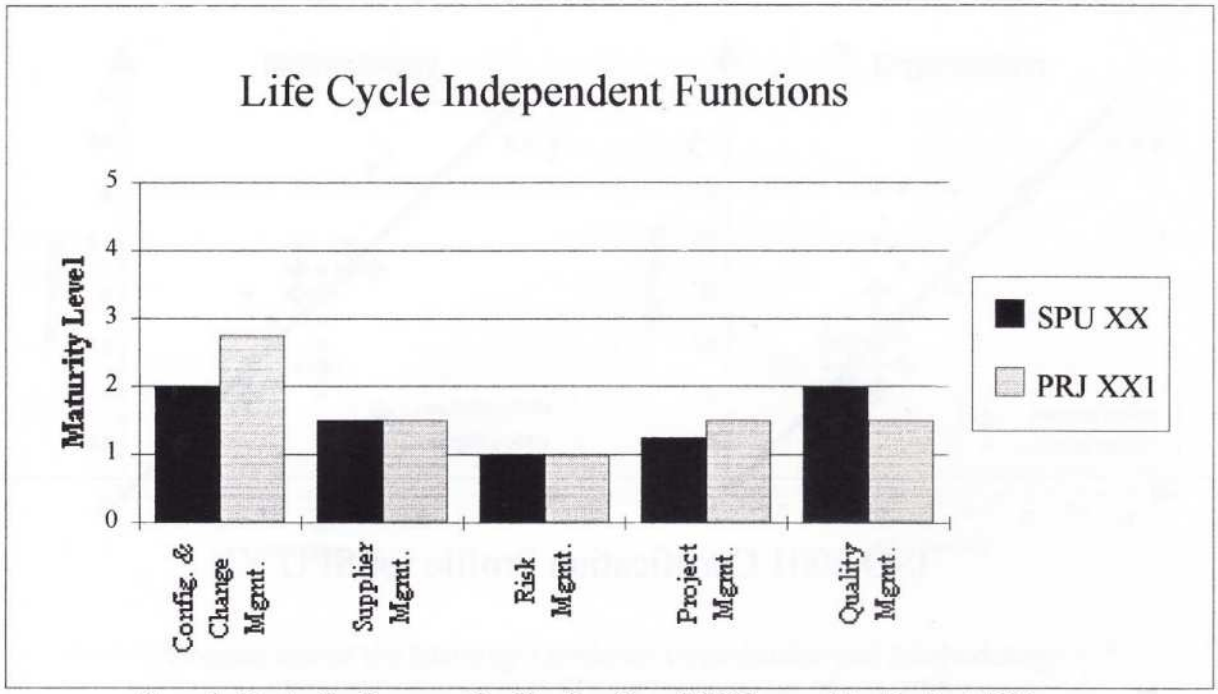
Supplier Management	Purchasing		
	Assessment of Subcontractors	2	100%
	Purchasing data	2	100%
	Verification of Purchased Product	2	100%
	Inspection and Testing	4	100%
	Receiving Inspection&Test	0	0%
	Purchaser Supplied Product		
Process Related Functions			
Process Control	Process Control		
	General Processes	23	100%
	Special Processes	14	100%
	Control of Nonconforming Product	3	30%
	Corrective Actions	20	100%
Process Measurement	Quality Records	13	100%
	Internal Quality Audits	4	100%
	Statistical Techniques	33	100%

ISO 9001 Attributes Mapped onto BOOTSTRAP´s Attribute Clusters & Percent Coverage of ISO 9001 Attributes

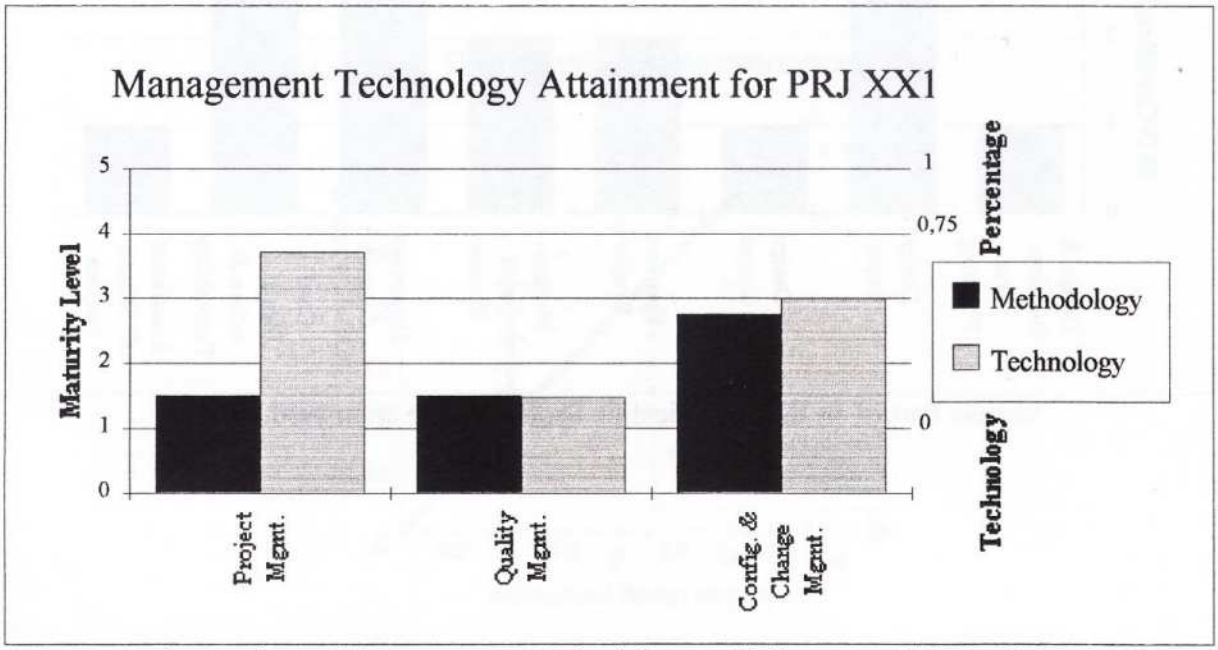
BOOTSTRAP ESPRIT 5441: A Quantitative Approach to Objective Quality Management

Correct Interpretation of ISO 9001 Attributes through ISO 9000-3		
ISO 9000-3 Attributes that represent an interpretation of ISO 9001 attributes	**ISO 9001 Attributes Related to the ISO 9000-3 Attributes**	**Meaning**
joint reviews, mutual cooperation	contract reviews management reviews design verification	* more emphasis on cooperation with the customer * the customer is to be integrated into contractual, design, and management reviews * the cooperation with the customer is explicitely taken into account when planning the project
purchaser´s management responsibility	responsibility and authority	* the customer is part of the organigram * the role of the customer within the project must be defined
contract items on quality	contract reviews	* each contract must clearly define the quality of the product which will be developed * a list of quality items should be part of the contract
purchaser´s requirements specification	design input	* ISO 9001 directly starts with design * software business demands a closer cooperation with the customer (see above) and thus a specification of the customer´s wishes is required before the design phase starts
development planning	design and development planning	* ISO 9001 demands an "activity assignment" and the definition of the "organisational and technical interfaces" . * ISO 9000-3 refines this requirement by including the issues: - Phases - Input to Phases - Output to Phases - Management - Methods and Tools - Progress Control
quality system documentation		* to provide a quality manual
quality plan, quality planning		* to establish a quality plan for each project based on the quality manual * quality planning covers the contents of a quality plan
acceptance planning and testing	inspection and testing	* test plan, test specification, test documentation
field testing	inspection and testing	* testing on the user´s site
maintenance	servicing	* the ISO 9001 attribute "servicing" shall also include: - maintenance plan - differentiation between types of maintenance activities - maintenance records and reports

BOOTSTRAP ESPRIT 5441: A Quantitative Approach to Objective Quality Management



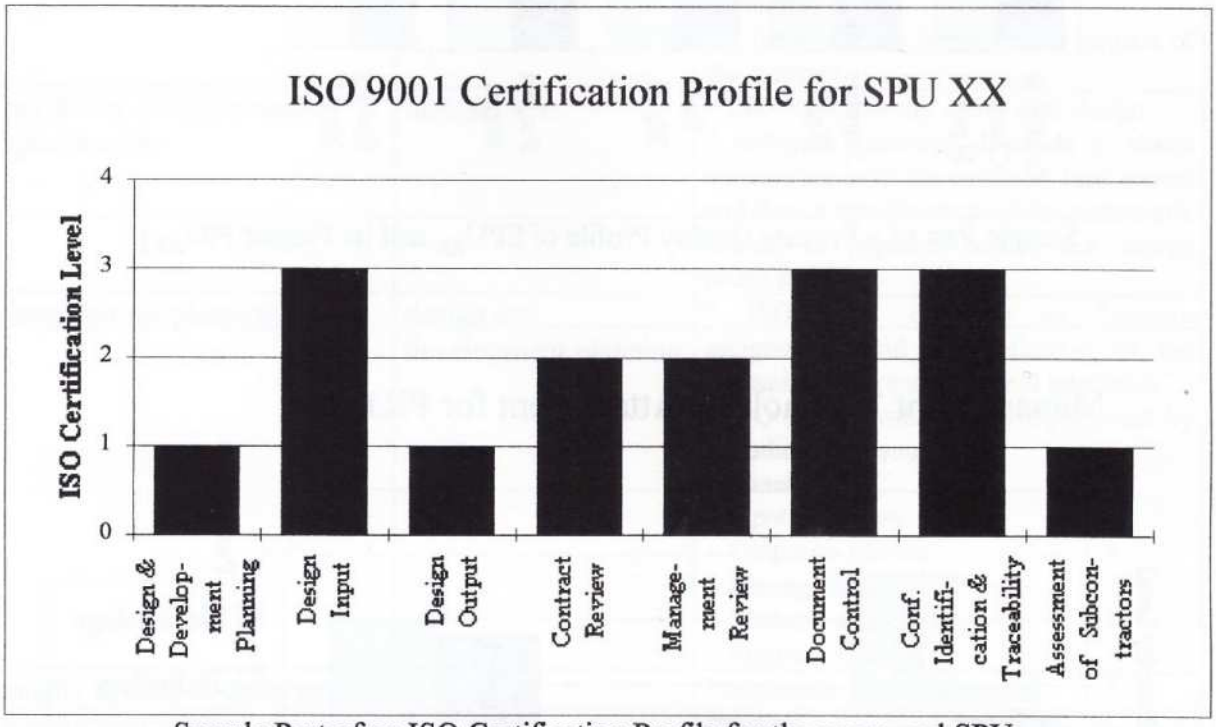
Sample Part of a Process Quality Profile of SPU_{XX} and its Project PRJ_{XX1}



Comparison of Management Methodology with Support of Technology

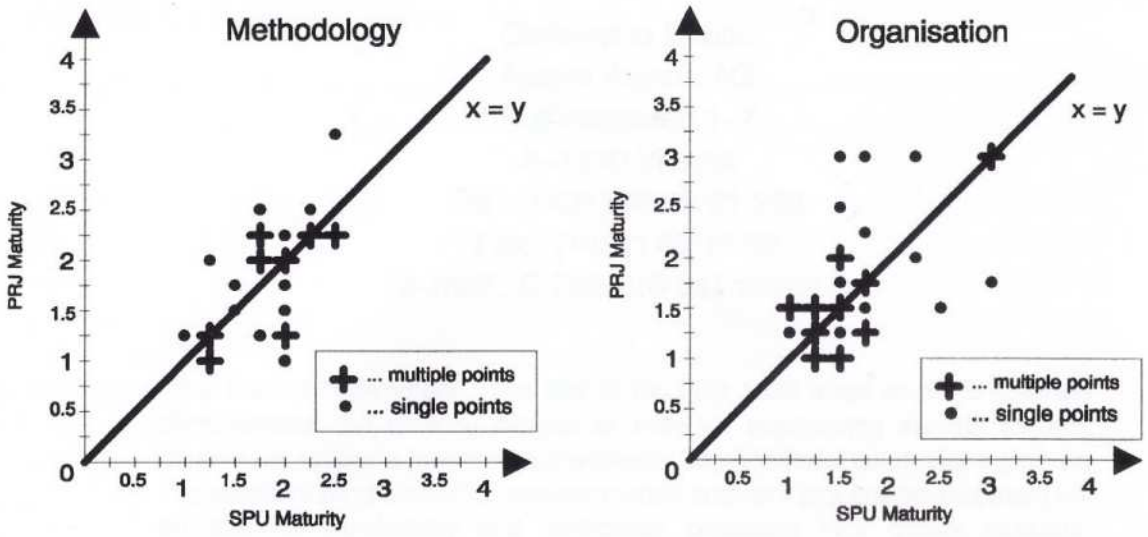
Authors: R. Messnarz, H.J. Kugler, V. Haase

BOOTSTRAP ESPRIT 5441: A Quantitative Approach to Objective Quality Management

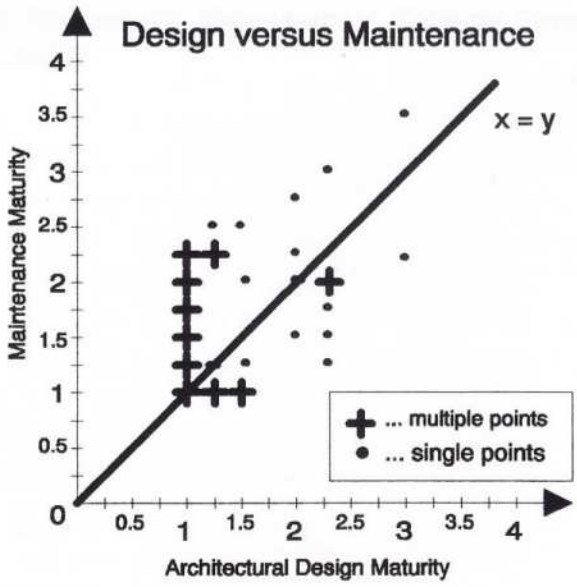


Sample Part of an ISO Certification Profile for the anomysed SPU_{XX}

BOOTSTRAP ESPRIT 5441: A Quantitative Approach to Objective Quality Management



Comparison of the Maturity Levels for Organisation and Methodology
(Refined Evaluation of 23 SPUs, 49 projects, Jan. 94, [3], [10])



Comparison of the Maturity Levels for Design and Maintenance Methodology
(Refined Evaluation of 49 projects, Jan. 94 [3], [10])

Authors: R. Messnarz, H.J. Kugler, V. Haase

ami:
a new paradigm for software process improvement

Christophe Debou
Alcatel Austria AG
Ruthnergasse 1–7
A–1210 Vienna
Tel : (+431) 39 16 21 268
Fax : (+431) 39 14 52
e–mail : C.Debou@aaf.alcatel.at

"The maturity movement". The title of the July 1993 issue of IEEE software demonstrates the growing interest of software engineering experts towards continuous software process improvement. Organizations put in place process improvement programmes to achieve mature software process development i.e. process for consistently and predictably producing high quality products. Measurement is an essential component of process improvement for understanding the process, monitoring the changes and evaluating the return on investment. **ami** (Application of Metrics in Industry) proposes an improvement paradigm based on the application of quantitative techniques. This paper describes the process improvement trend and the **ami** paradigm illustrated with a real case study.

Keywords: Assessment, Measurement, Software Process Improvement, Software Quality

1. Introduction

Software process improvement is certainly the keyword of the nineties within the software engineering community. Conferences, journals make their headlines with this subject though process improvement has always been implicitly performed, maybe not in an orderly way. Increased competition and the growing complexity of software systems and surrounding infrastructure to build it, is forcing more software organizations to improve their processes. People are aware that not only should the technological aspects of software engineering (e.g. CASE tool) be addressed but the process issues as well. Before you can automate software engineering, you must first define and streamline the software process.

Process improvement has its roots in manufacturing mainly from Japan (Kaizen movement) and from a few American quality Gurus such as W.E. Deming [Dem82], J.M. Juran [Jur88] or P.B. Crosby [Cro79], who advocate continuous process improvement for improving products quality. The Shewhart cycle (Plan–Do–Check–Act) made popular by Deming is now referenced by many software process improvement initiatives [Hum92] [ami92].
The aim of this paper is to introduce a new paradigm for software process improvement which considers measurement as the principal component of an improvement approach. After a brief presentation of the necessary requirements for a process improvement approach, the **ami** paradigm is developed according to the following frame: origin, paradigm, underlying theory, organization and pros & cons. The stepwise paradigm is illustrated with a continuous case study.

2. Software process improvement: State of the Practice

2.1. The necessary components for a successful software process improvement approach

By analyzing successful existing software process improvement approaches, necessary activities/components can be defined:

- **Assessing:** The current development process has to be assessed to point out the problems and areas to be improved. The assessment can be conducted against a written process model or a process maturity model (SEI CMM). Many assessment procedures have been developed from the SEI CMM such as Trillium [CoD92] (Bell Canada) or BOOTSTRAP [Koc92]. Therefore, an **assessment procedure** and optionally an **evolutionary maturity model** or a **best practices list** or a **generic software development model** is needed.
- **Modelling:** Process improvement requires a basis for defining and analysing processes. In that context, process modelling can fulfill many roles [CKO92] such as the detection of problem areas, estimation of impact of potential changes to the software process (simulation), comparison of alternative software processes, ... A **process modelling formalism** is needed.
- **Improving**: Improvement actions resulting from the assessment are implemented and monitored (measuring). An **improvement framework** for quantitative evaluation of improvement goals is needed.
- **Measuring:** Measurement is considered as a monitoring or controlling tool on the one hand and as a support for decision making with respect to process improvement on the other hand. A **measurement framework** (guidelines for measurement plan production and implementation) is needed.

Beside these components, an improvement organization should be put in place for sponsoring, coordinating, implementing and promoting the improvement initiative.
All those components are drawn on Figure 1 to build the Modelling/Assessing/Measuring/Improving paradigm. The coverage of the **ami** paradigm is indicated as well i.e. the quantitative aspect and the assessment and improvement procedures. The next chapter briefly surveys existing software process improvement approaches.

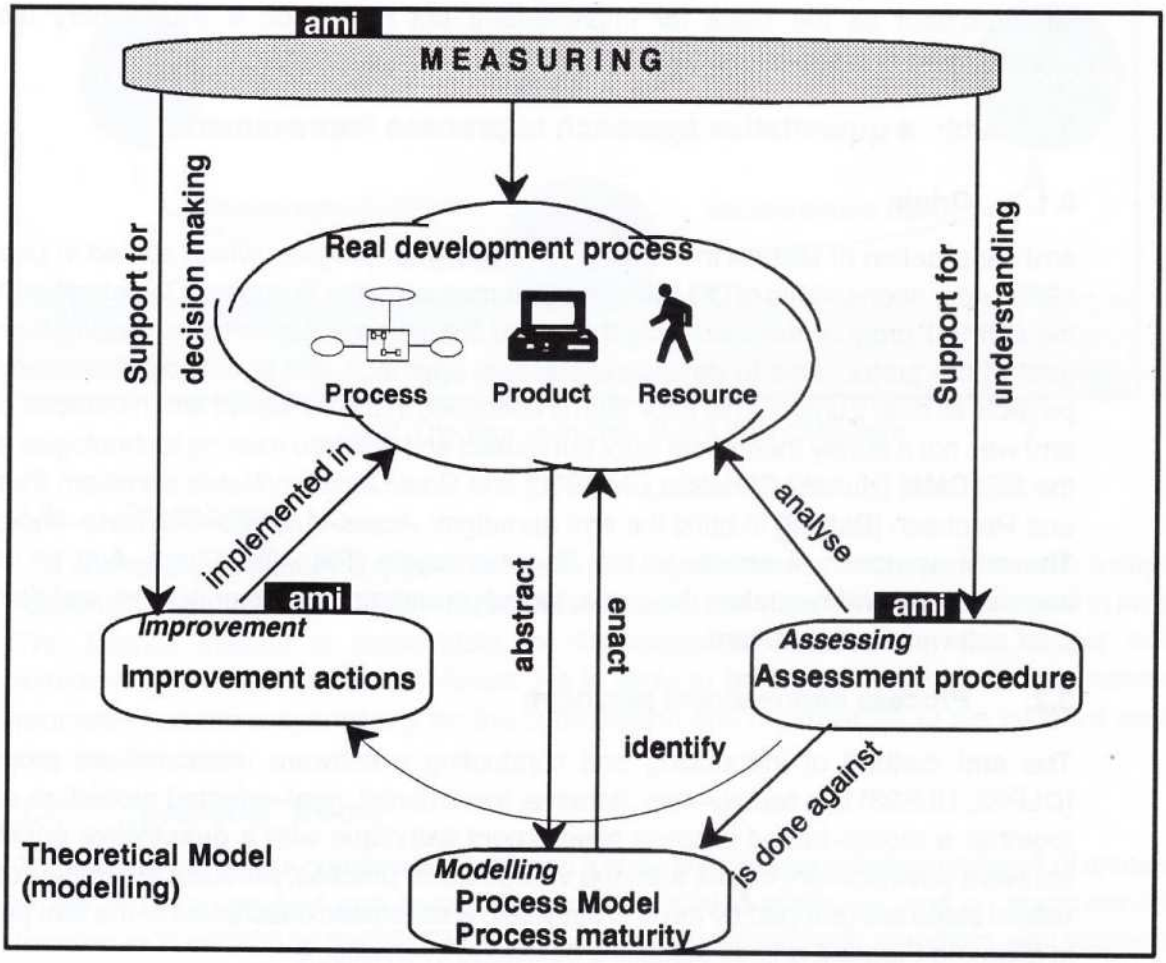


Figure 1: The Modelling/Assessing/Measuring/Improving paradigm

2.2. Taxonomy of software process improvement approaches

D. Card [Car92] identified two basic approaches to process improvement namely the ***analytic*** approach and the ***benchmarking*** approach.

1) The ***analytic*** approach relies on quantitative evidence to determine where improvements are needed and whether improvement initiative has been successful e.g. Shewhart cycle [Dem82] or the Software Engineering Laboratory (SEL) approach [BCM92]. The SEL has conducted successful experiments during the last 17 years evaluating promising technologies like Ada, Cleanroom, OO, ...The SEL combines a quality improvement paradigm with a measurement framework namely the Goal/Question/Metric paradigm[BaR88]. In addition, the concept of experience factory [Bas92] covers the reuse of experience and collective learning. Improvement means are determined through controlled experiments.

2) The ***benchmarking*** approach depends on identifying an "excellent" organisation and documenting its practices and tools. The most famous benchmarking approach is the Software Engineering Institute (SEI) Capability Maturity Model [CMM93a, CMM93b] which will be roughly described in the **ami** presentation (Chapters 4. and 3.4.). It works under the assumption that an organization who will adopt these practices and tools will become also excellent: *"Learn from the best".*

To this taxonomy, a **mixed** approach (e.g. the **ami** approach) can be added considering measurement as the basis for improvement but relying on a evolutionary model of development process capability and maturity.

3. ami: a quantitative approach to process improvement

3.1. Origin

ami (Application of Metrics in Industry)[1] is a two–year program which started in December 1990 under sponsorship of DG XIII of the Commission of the European Communities through the ESPRIT programme promoting the use of measurement in software development. The goal of the project was to develop a practical approach and to validate it on a variety of projects all over Europe .This approach is described in the so–called **ami** handbook [ami92]. **ami** was not a purely theoretical work but reused and adapted existing technologies such as the SEI CMM [Hum89,CMM93a,CMM93b] and Goal/Question/Metric paradigm from Basili and Rombach [BaR88] to build the **ami** paradigm: *Asses–Analyse–Metricate–Improve.*
The **ami** approach is similar to the Shewhart cycle (Plan–Do–Check–Act) for process improvement. **ami** has taken this cycle, based on common sense principles, and developed it for software measurement.

3.2. Process improvement paradigm

The **ami** method of introducing and conducting a software measurement programme [DLP92, DLS93] is a twelve–step, iterative, incremental, goal–oriented procedure coupling together a model–based process assessment technique with a quantitative approach to software development issues from the viewpoints of process, products and resources. The twelve steps are grouped by three in activities. A schematic description of the **ami** paradigm is given on Figure 2 and all steps are described in chapter 4.

Why to consider **ami** as a process improvement paradigm? **ami** can be applied for many purposes when measurement is used for management or decision making and therefore also for process improvement. The whole approach offers a complete framework for process improvement while fulfilling the main requirements for such approach: Iterative (continuous), goal–oriented, quantitative, involvement of everyone, necessary management commitment. This paradigm encompasses the main components for process improvement as defined in chapter 2. However, some of them have to be instantiated (e.g. assessment procedure, process modelling technique).

1. The **ami** consortium comprises: GEC Marconi Software Systems (UK), Alcatel Austria Forschungszentrum (Austria), Bull AG (Germany), Objectif Technologie (France), GEC Alsthom (France), ITS (Spain), Olivetti Group (Italy), RWTÜV (Germany) and South Bank Polytechnic (UK).

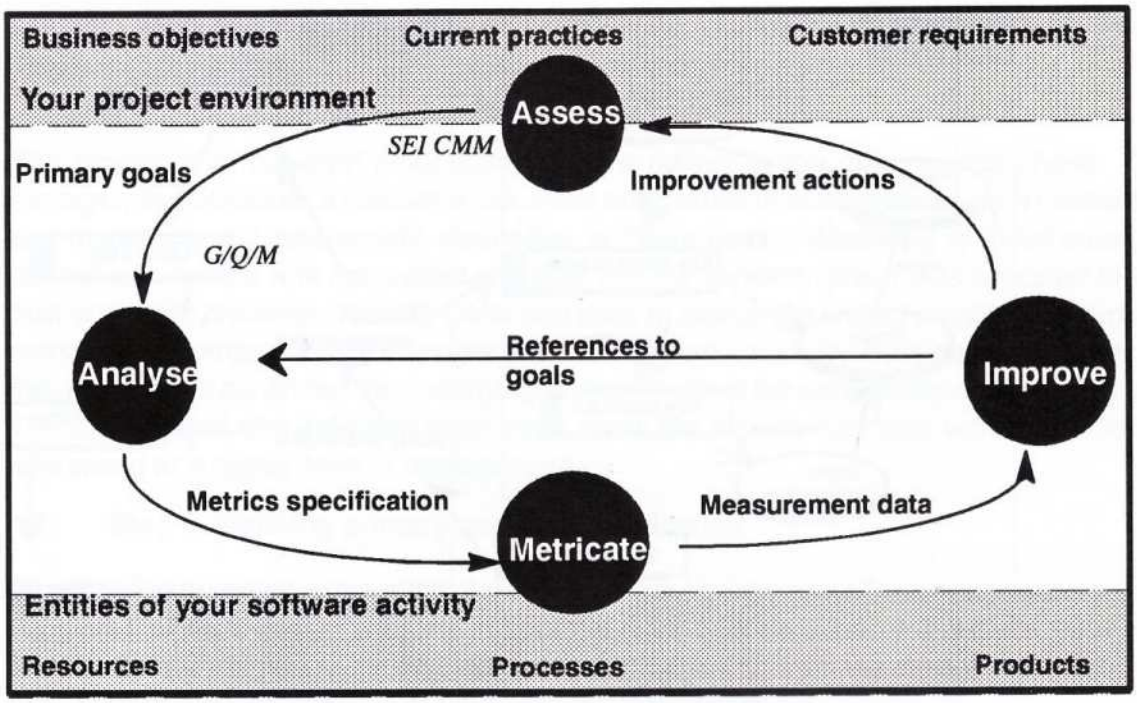


Figure 2: The four activities of the **ami** paradigm

3.3. Organization

It is necessary to identify the participants in the improvement initiative and then to assign responsibilities. Besides concerned project members, two main roles are considered in **ami**. The Metrics initiator is responsible for the organization, allocating the budget and establishing the strategical orientation. He is likely to be a senior manager. The metrics promoter has the responsibility for the coordination and organization of the initiative on a day–to–day basis.

3.4. Underlying "theory"

The **ami** paradigm is a combination between a software process definition in terms of entities of the process, product and resources and their related attributes, and an improvement framework illustrated by the SEI CMM and the Goal/Question/Metric paradigm. **ami** attempts to cover the whole improvement cycle (alike the Shewhart cycle Plan/Do/Check/Act described in [Dem82]). It is slightly different from the SEL approach since a maturity model is advised for validating the feasibility of a goal.
The relationships between **ami** and other process improvement approaches are twofold:

- **ami** integrates and enhances to its purpose existing approaches (SEI CMM, Goal/Question/Metric paradigm). Concerning the CMM, the measurement scope is extended to support extensively, decision making at all maturity levels.
- **ami** is a "tool" for controlling (and improving afterwards) the conduct of process model techniques, i.e. a given phase, the passage from a maturity level to another (SEI CMM) or the introduction of ISO 9000.

One the hand, **ami** would like to stay generic and let the user choose his own model of software process development or process maturity improvement. But too much abstraction does not lead to a DE facto acceptance of the approach. Therefore, the SEI CMM is advocated mainly for validating the primary goals. The relationships between the CMM and the **ami** *model* is depicted in Figure 3.

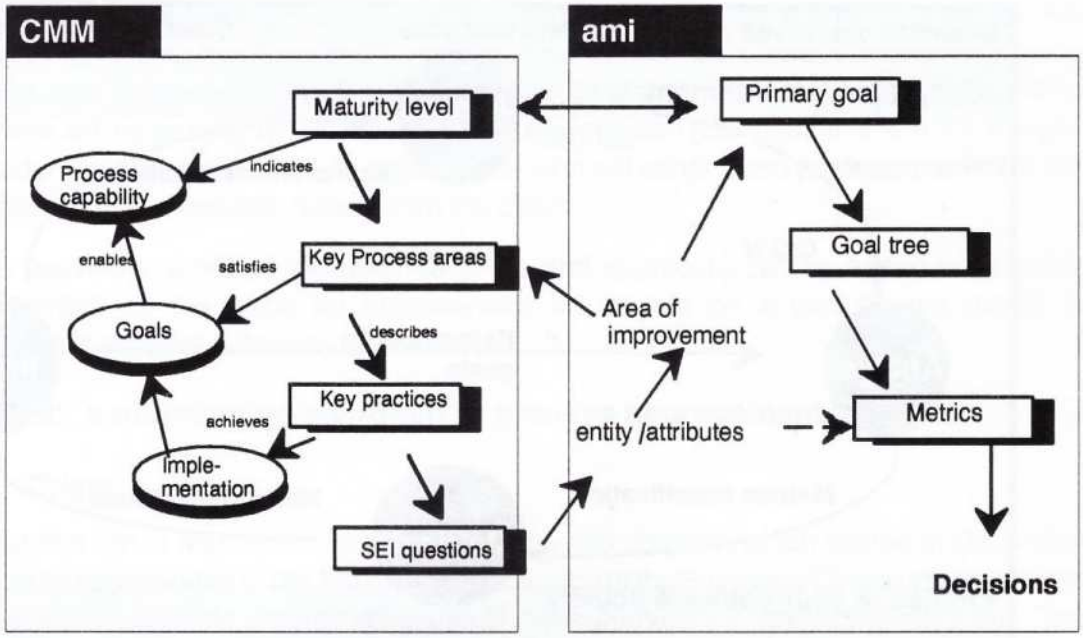


Figure 3: Coupling **ami** with SEI CMM

ami runs top–down i.e. the primary goals are mainly management goals and are broken down into more manageable goals until directly quantifiable goals (directly definable with metrics) are defined. The goal tree whose leafs are metrics stimulates decision making necessary for improving the process. On the other hand, from the SEI questionnaire, weaknesses and area of improvement are highlighted, the latter being related with the key process areas. This is used as input to the definition of the primary goals whose ambition level has to relate with the global level of the maturity of the software development process. An empirical relationship between the goal and the level of maturity is established. The higher the level of maturity is, the more ambitious the goals can be. Two classes of goals are considered: *Knowledge goals (*to evaluate, to monitor, to understand,...) requiring SEI level 2 upwards and *Change goals* (to increase, to reduce, to achieve, to change,to improve) requiring at least level 3.

4. "Assess" activity: assessing the project environment.

To illustrate the description of each step, a continuous case study is added. This project is one of the 20 validation projects performed during the **ami** project. This is a real–time embedded software project in a maintenance phase.

4.1. Step 1: Assessing the project environment

The aim of this step is to point out weak areas, critical and problematic parts of the development process. The **ami** consortium advises the use of Capability Maturity Model [CMM93a, CMM93b] defined by the Software Engineering Institute. The outcome of this particular procedure is not only the scaled result of the assessment in terms of a maturity level (from 1 to 5), but an overview of critical items that serve as an input for defining the primary goals for improvement of the process and measurement of their characteristics. Although recommended, the assessment is not restricted to the SEI CMM procedure. A quality audit or similar investigation is valid as long there is a guarantee of getting an objective and sufficiently exhaustive view of the development process. A European variant of the SEI CMM

has also been developed during the ESPRIT project 5441 BOOTSTRAP [Koc92] which includes ISO 9000 issues in the assessment method.

Case study:
The project is a real–time embedded software project in the maintenance phase. The management perceived a number of risk areas that the use of metrics would aid in controlling and monitoring e.g. productivity, estimating, software quality. According to those areas, a budget was agreed with the project manager and the senior engineer was allocated to the task of metrics promoter. The SEI CMM was used to assess the project involving the project manager, the project quality manager and the department quality manager. The results of the assessment confirmed the underlying feeling of where the weak areas of the project lay. The assessment also indicated other weak areas like training but these were in fact to be addressed at a higher level of management.

4.2. Step 2: Defining primary goals for metrication

The knowledge of status quo within the development environment allows one to express the objectives of participating subjects in terms of primary goals. They can include business objectives, environment–specific objectives, goals resulting from the assessment step etc. At the beginning a reasonable number is 1 or 2 per project.

4.3. Step 3: Validating primary goals

This is a very important and sensitive step which may result into success or failure of the entire improvement programme. Goals established in the previous step are to be validated in order to ensure their consistency in terms of:

- Consistency between goals and your assessment conclusions,
- Consistency between selected goals and time scales,
- Consistency between goals and budget.

The degree of ambition of the goals should also be checked in two ways:

- Check the difficulty of the goal against the ability of the development environment to fulfil it. From this viewpoint, there are two classes of goals: *knowledge goals* (to evaluate, to monitor, to understand, to predict...) and *change goals* (to increase, to reduce, to achieve, to change, to improve...). The first class is applicable from maturity level 2 upwards, the other requires at least level 3. Therefore, metrication has to be first supported by knowledge goals before making use of improvement goals.
- Check the timeliness (short/mid/long term) of the goals resolving inconsistencies and giving priorities based on agreement of all participants.

Case study:
Interviews with the main project actors allowed the identification of primary goals. At the start, there was a great temptation to go straight for improvement goals. But, it became apparent that one need first to establish baselines against which improvement could be measured. This was reinforced by the level of maturity. From an initial set of 10 goals, 6 were selected from project members at different levels and areas of interest. In examining them, certain goals were identified as being sub–goals of other goals. So a goal tree could be produced in anticipation. We will focus on the following goal: ***G1 To gain a better understanding of project software quality?***

5. "Analyse" activity: Defining goals and metrics.

5.1. Step 4: Breakdown of goals to sub–goals

This step is a simplified adaptation of Basili's Goal/Question/Metric paradigm [BaR88], a flexible method for refining goals of any level to metrics using process and product related questions. The thinking process for decomposing goals into sub–goals is documented with a table of questions and a list of related entity–attribute pairs. The decomposition process runs as follow:

```
goal_level = primary
Repeat
    define the list of entities related to each goal of goal_level
    For each entity of the list
        Ask questions related to the goal
    Decomposition in sub-goals
    decrement goal_level
Until quantifiable sub-goals are attained
```

Case study:

The metrics promoter collected a lot of information on goals at a number of levels in the organisation. The project perceived that there were three areas that we needed to consider in the context of quality:

1. Where did the errors occur in the code? Did some software parts cause more errors than others?

2. What about test coverage? Were some code parts missed during testing?

3. What sort of turn around were given to report software errors?

Other issues could have been raised to gain further insight into this goal e.g. to measure the complexity of each module. But, it was decided to restrict on these three sub–goals because they could be achieved with little change to the project organization and little additional load. One sub–goal is considered for our purpose:

G1.1 Provide information on the location of software errors?

5.2. Step 5: Verifying the goal tree

The verification includes checks of:

- Homogeneity of levels of detail in each branch of the tree,
- the internal consistency of the tree i.e. no contradiction between goals and sub–goals.
- the external consistency of the tree i.e. the relevance of the goal tree to primary goals.

5.3. Step 6: Identification of metrics by questions

An additional questioning procedure with the aim to describe sub–goals in a quantitative way is applied on a validated goal tree with elementary sub–goals (expressible with metrics) on the leaf level. The outcome is a set of metrics covering the goals selected in the appropriate step of the measurement programme. **ami** offers a so–called *basic metrics set* which has been proved as useful if starting a measurement programme for the first time.

Case study:

The questions derived from the sub–goal G1.1 are:

Q1.1.1 In which modules are software errors located and what type are they?
Q1.1.2 In which functions are software errors located and what type are they?
The derivation of the metrics was quite simple:
M1: Software error location function and module reference.
M2: Software error classification

6. "Metricate" activity: Implementing the measurement plan.

6.1. Step 7: Writing the measurement plan

The measurement plan should contain all information of interest for the metric data collection procedure as objectives of the measurement plan, metrics definitions, metrics analysis procedures, responsibilities, timescales, references and a logbook for recording measurement activities. In addition of being a plan for the collection of data, the measurement plan records information of the software development environment, the software development process, the strengths and weaknesses of the environment and process, and a standard for communication between the participants. The Handbook includes precise templates for the measurement plan and metrics definition.

6.2. Step 8: Collecting primitive data

Collection is performed manually (e.g. using collection forms) or automatically by support of tools for static analysis (size, structure of code, documents, etc.), dynamic analysis (statement coverage during testing, etc.), configuration management (number of faults, run times, change requests, etc.), project management (schedule, costs, etc.) and data management (measurement data). A metrics database (or simple spreadsheet) belongs to the kernel of the system.

Case Study:
The majority of data was collected manually and drawn from existing sources within the projects such as library systems, quality assurance reports and configuration control. Overall, there was a feeling that using tools to automate the collection process would have reduced the collection effort and increase the data accuracy.

6.3. Step 9: Verifying the primitive data

The metrics data should be verified as they become available in order to allow corrections and additions. Furthermore, the accuracy of the data collection process may be quantified. Motivating feedback to the data collectors can be provided. During data verification any obviously unusual data (outliers) should be detected and the reasons should be identified.

7. "Improve" activity: Exploiting the measures.

7.1. Step 10: Data presentation and utilisation

Any controlling, changing and/or improvement activity starts with an appropriate presentation of the measurement data. Advantages of graphical presentation aids against statistical techniques are robustness (because of no underlying statistical assumptions) and their user–friendliness as information is conveyed in a more clear way.

7.2. Step 11: Validating metrics

Validation of metrics means showing that they are adequate for the purpose required for them. This can be done through a procedure that is either objective or subjective. For an

objective validation we compare the measurement data with an expected trend or with an expected correlation to other data. In a subjective validation people who are involved in the process give an opinion as to whether the metrics corresponds to the sub–goal.

7.3. Step 12: Relating the data to goals

Goals for metrics within the knowledge class are categorized between evaluation (understanding) and prediction goals. When dealing with prediction goals, the interpretation of data may require a model (e.g. cost, reliability, quality models) to relate an inherent property of the software (metrics) and the final performance of the product.
By relating data to goals, the objective is to determine if goals are fulfilled, how quickly they are fulfilled and if not fulfilled, then why. This last step involves producing an action plan based on the collected data which can involve improvements to the development process. From this action plan, this step will also include a modification of the primary goals. In the case of improvement actions, new goals will cover the monitoring of those actions.

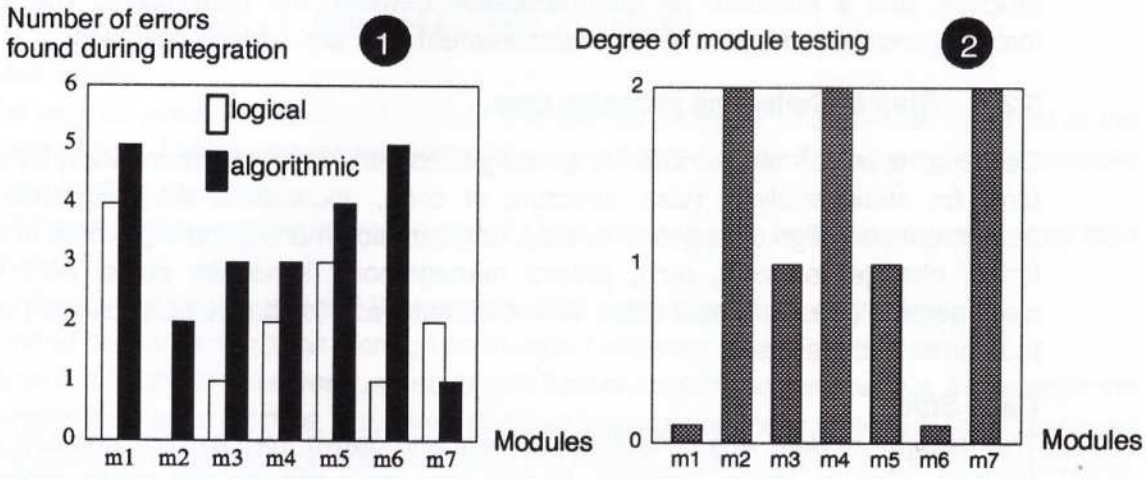


Figure 4 : Organization of the measurement process

Case Study:

The metrics promoter assembled the results and presented them to the project manager in a combination of tabular and graphical diagrams. This was prepared 6 months after the start of the initiative when results were sufficient to show trends. At this early stage,the project realised from the reports that some of the metrics were not as valuable as initially thought. Some of them were dropped while others were collected in a modified form. This number could have been greater if the verification steps had not been carried out. A simple example of interpretation is given on Figure 4. The left chart provides the distribution of errors (logical and algorithmic) within modules (a significant sample). It shows which parts of the system are particularly error–prone and the types of errors and therefore give insights for the goal G1.1. This can be also related with another goal dealing with the identification of well–tested part in the software. A comparison can then be made to see if the more poorly tested parts of the software are the part where the most errors are found. The right chart gives per module the degree of module test (0 means minimum testing, 1 average testing and 2 maximum testing). It appears that more errors, essentially algorithmic, occurred in less tested modules. Finally, a better view of the software in terms of error–proneness and testing was provided and will drive the implementation of improvement actions mainly concerning the module test phase.

The step 12 is the last step of the **ami** loop, often seen as a feedback step, since the method is iterative and incremental in its nature. The Figure 5 summarizes the 12 steps.
After stepping through the loop, it is worth quantifying the initial benefits of the improvement programme. The iterative aspect of the method permits either, by a reassessment, the refinement of goals and metrics set, but also the improvement of the metrication process (data collection, storage, analysis...). Having an appropriate degree of experience and an appropriate amount of information from the measurement process, improvement action can be taken depending on the goals fulfilled.

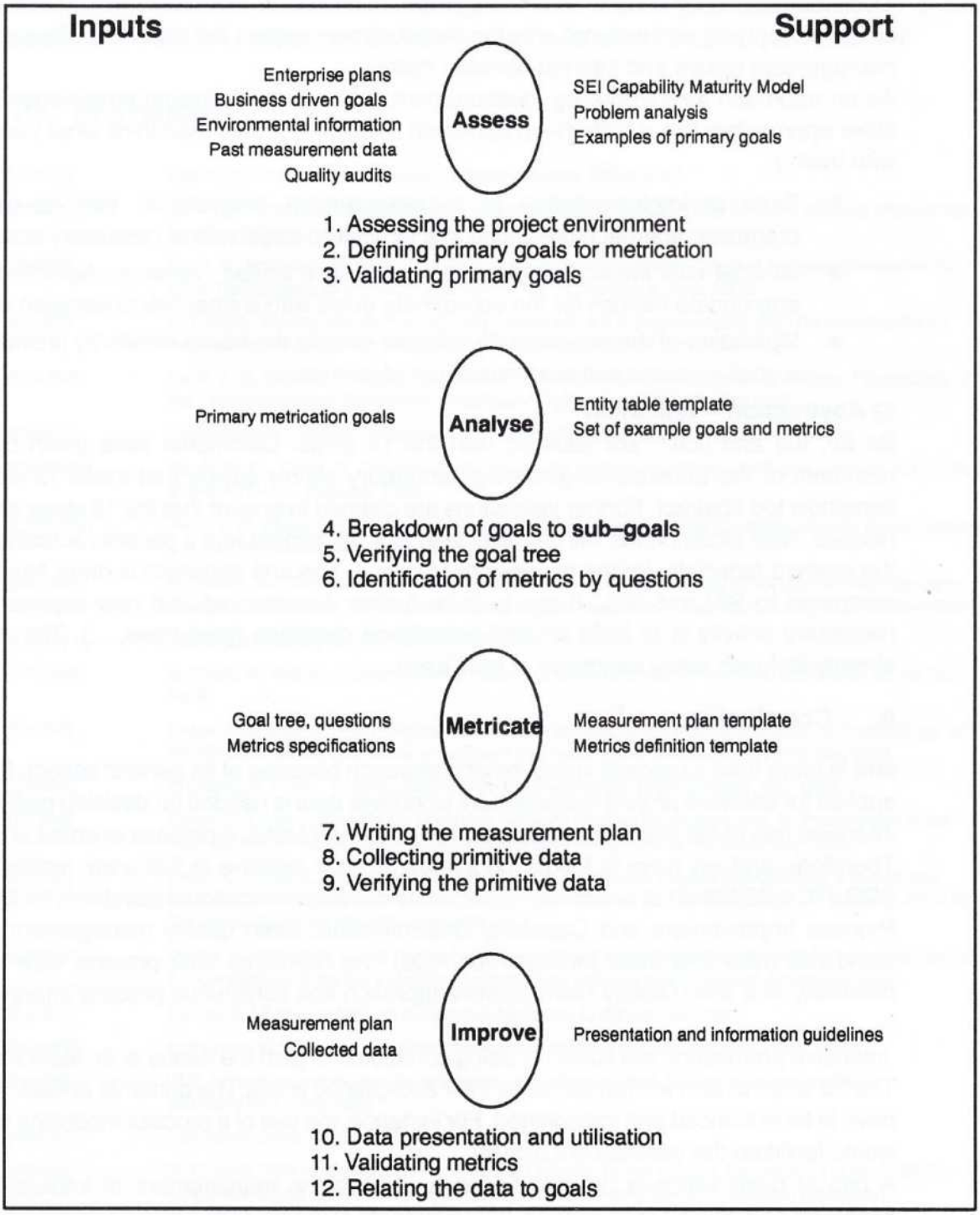


Figure 5: The Structure of the **ami** Method

8. Pros & Cons

⊕ Pragmatic and economic approach for software measurement & process improvement

The ami approach is an excellent means for selling software measurement and its necessity for process improvement. This goal–oriented approach is rather straight–forward to apply and self–contained. The efficient use of measurement is demonstrated through a wide range of examples and case studies [DLS93]. Several experiences have already been published in the telecommunications [DPF92, DeS93, DFS93] and Defense [PFS92] areas.

This was necessary to convince management to introduce the measurement activity. The scope of applying ami was not only the measurement aspect but also the software project management issues and internal decision making.

As an approach for introducing measurement, ami has the following advantages against other approaches like a bottom–up approach (First collect data then think what you can do with them!):

- Ease of implementation of a measurement programme; easy–to–use and pragmatic approach described in a structured sequence of necessary actions
- Minimal cost impact, predictable costs with a limited variance; Selection of the appropriate metrics for the appropriate goals with a clear link to decision making.
- Motivation of the project staff; Address directly the user's needs by providing him a goal–oriented and customized set of metrics.

⊖ Abstraction – validation

So far, the ami users are satisfied with the 12 steps. Comments were given by some members of the Software Engineering Laboratory mainly saying that these 12 steps are somehow too abstract. Further validations are claimed to ensure that the 12 steps are really needed. New experiments will certainly lead to a refinement and a periodic actualization of the method (specially for the assessment activity). The ami approach is more higher level compared to SEI and SEL. It needs to be further documented with new experiences. A necessary activity is to build an **ami** experience database (goal trees, ...). The **ami** tool already includes many examples of goal trees.

9. Conclusion

ami is more than a process improvement approach because of its generic aspect. It can be applied for software project management whenever data is needed for decision making. The intensive use of the SEI CMM makes **ami** in its current status a process oriented approach. Therefore, **ami** will have to follow the standardization initiative in this area, mainly SPICE (ISO JTC1/SC1/WG10) which was approved to develop international standards for Software Process Improvement and Capability Determination. Even quality management system standards (new ISO 9000 foreseen for 1996) will reenforce their process view and the necessity of a Total Quality Management approach and continuous process improvement.

Intensive promotions are currently being conducted to gain the status of de facto standard. The 12 steps of **ami** will remain since their acceptance is big. The contents of each step will have to be enhanced and instantiated. For instance, the use of a process modelling technics would facilitate the assessment activity.

A critical point which is not addressed by **ami** is the management of knowledge and experiences within an organization. Improvements of the process are to be documented and promoted across the company. Bill Curtis, former director of SEI software process program

raised this issue and considers that sharing the knowledge within an organization is a dominant factor in increasing the software development productivity. The SEL experience factory is certainly the best existing example of organizational knowledge database.
At least, **ami** users are able to share their experiences in the **ami** newsletter and very soon through an **ami** user group. Even a mailing list has been recently set up (send subscription to ami–request@aaf.alcatel.at).

Acknowledgments:
The author would like to thank all partners involved in the **ami** project who permits this work to be carried out. Thanks also to all ICSN reviewers who helped in the refinement of this paper. **ami** was partly supported by the Austrian Innovations und Technologiefonds"

10. References

[AMI92] Esprit II 5494 ami: ami Handbook, Published version, March 1992.

[Art93] L.J. Arthur, Improving software quality, an insider's guide to TQM, Wiley series in software engineering practice, 1993

[BaR88] Basili, V., Rombach, H.: The TAME Project: Towards Improvement–Oriented Software Environments, IEEE Transactions on Software Engineering, Vol 14, Number 6, June 1988.

[Bas92] V.R. Basili, The experience Factory: Can it make you a 5 ?, proceedings of the 17th annual software engineering workshop dec 1992, Goddard NASA center, Maryland (USA).

[BCM92] Basili V., G. Caldiera, F. McGarry and al.: An Operational Software Experience Factory, Proceedings of the 14th International Conference on Software Engineering (ICSE 92, May 1992.

[Car91] Understanding process improvement, D. Card, IEEE software, July 1991.

[CMM93a] Paulk M C, Curtis B, Chrissis M B: Capability Maturity Model for Software, version 1.1, CMU/SEI–93–TR–24, February 1993.

[CMM93b] Paulk M C, Weber C V, Garcia S M, Chrissis M : Key practices of the Capability Maturity Model, version 1.1, CMU/SEI–93–TR–25, February 1993.

[CoD93] F. Coallier, J–N Drouin, Bell Canada, Developing an assesment method for telecom software system: an experience report, Proceedings of 3rd european conference on software quality, Madrid, Nov 1992.

[Cro79] P.B. Crosby, Quality is free, McGraw–Hill, 1979

[CKO92] B. Curtis, M. Kellner, J. Over, Process modelling, Communication of the ACM, September 92, vol 35, No 9.

[DeS93] Debou C, Stainer S.: Improving the maintenance process: a quantitative approach, In: Proceedings of the 6th international conference on software engineering and its application, Paris, Nov 1993.

[DFS93] C.Debou, N. Fuchs, H. Saria, Selling believable technology, IEEE Software, Nov 1993.

[DLP92] Debou C, Lipták J, Pescoller L: Managing Software Process by Applying **ami**. In: Proceedings of the MSP–92 IFAC – annual review in Automatic programming, volume 16, Graz, May 1992.

[DLS93] Debou C, Lipták J, Shippers H.: Decision making for software process improvement: a quantitative approach, In: Proceedings of the 2nd internation conference on "achieving quality in software" ACQUIS 93, Venice (Italy), pp 363–377, Oct 1993.

[DPF92] Debou, C., Pescoller, L. and Fuchs, N.: Software Measurements on telecom systems – Success stories ? : Proceedings of the 3rd European conference on software quality, Madrid, November 1992.

[Fen91] Fenton, N.: Software Metrics: A Rigorous Approach, Chapman Hall, 1991

[Hum89] Humphrey, W.: Managing the Software Process, Addison–Wesley, Reading, Mass., 1989.

[Hum92] W.S. Humphrey, Introduction to software process improvement, SEI technical report, CMU/SEI–92–TR–7, June 1992

[Jur88] J.M. Juran, Juran on planning for quality, MacMillan, 1988

[Koc92] G. R. Koch, THe bootstrap initiative – reported benefits for the industry, Proceedings of the IPSS–Europe International conference on lean software development, Oct 1992, Stuttgart (Germany).

[PFS92] I. Perez, P. Ferrer, A. Fernandez: Application of Metrics in Industry : Proceedings of the 3rd European conference on software quality, Madrid, November 1992.

SCOPE: A guide for Software Product Quality Evaluation

Jørgen Bøegh,
DELTA,
Venlighedsvej 4, DK-2970 Hørsholm, Denmark
tel. +45 42 86 77 22 fax. +45 42 86 58 98

1. Introduction

The methods and techniques developed to support process analysis and assessment are very usefull in order to achieve software products of good quality. However, with our present knowledge it is not sufficient to control the process. It is also necessary to control the product, both the intermediate products and the end product. Methods for software quality measurement are therefore a necessary supplement to process measurement. This paper describes a scheme for independent third party demonstration of software product quality.

The scheme for quality evaluation of software products was developed by the ESPRIT II Project 2151 SCOPE (Software CertificatiOn Programme in Europe). The main results from SCOPE has been adopted by ISO and is currently in the process of becomming international standards.

2. International standard ISO/IEC 9126

Currently the relevant international standard for software evaluation and metrics is ISO/IEC 9126: Information technology - Software product evaluation - Quality characteristics and guidelines for their use [1]. The main purpose of the standard is to identify the six important software quality characteristics:

Functionality - A set of attributes that bear on the existence of a set of functions and their specified properties. These functions are those that satisfy stated or implied needs.

Reliability - A set of attributes that bear on the capability of software to maintain its level of performance under stated conditions for a stated period of time.

Usability - A set of attributes that bear on the effort needed for use, and on the individual evaluation of such use, by a stated or implied set of users.

Maintainability - A set of attributes that bear on the effort needed to make specified modifications.

Portability - A set of attributes that bear on the ability of software to be transferred from one environment to another.

Efficiency - A set of attributes that bear on the relationship between the level of performance of the software and the amount of resources used, under stated conditions.

In order to apply ISO/IEC 9126 in practice guidance is needed on how to state quality requirements and specifications, how to develop and select indicators and metrics, and how to interpret and report evaluation results. This information will be provided in guidelines to the standard. The most important guide in this context is the evaluators guide [2]. It takes a quantitative approach to evaluation and makes use of rigorous metrics. The evaluation metrics are structured in evaluation modules. They are described in the guide to producing evaluation modules [3].

In the following the SCOPE evaluation scheme as described in the evaluators guide and the guide to producing evaluation modules is introduced. A more thorough description can be found in [4] and [5].

3. The Evaluators Guide

The evaluators guide guidance for those who perform independent software product evaluation professionally and those who demand independent quality evaluation. It defines a procedure for obtaining a quantitative statement about the quality of a software product. The quality aspects refered to are those identified in the standard ISO/IEC 9126. The evaluation procedure described in the evaluators guide is intended for:

* testing laboratory evaluators, when they provide software product evaluation services,
* software producers, when they plan independent evaluation of their products,
* software customers, when they specify acceptance evaluation in their contracts with software producers,
* software users, when they study evaluation reports.

The evaluation procedure divides the evaluation into discrete activities which are performed by the evaluator. The first step is the analysis of the requirements of the evaluation. In the next step the evaluation specification is developed. The third step is the design of the evaluation plan including the selection of evaluation modules and the fourth step is the conduct of the evaluation. The final step collects the results of applying the evaluation modules in an evaluation report. Figure 1 gives an overview of the evaluation procedure.

4. Analysis of evaluation requirements

Before the evaluation can start the quality requirements of the software product must be carefully analysed. This analysis must take into account the demands of the environment where the software is to be used. Also relevant laws and regulations must be considered. The quality requirements are formulated in terms of the quality characteristics in ISO/IEC 9126.

All six characteristics are not equally relevant for all software products. Therefore, all characteristics need not be evaluated for all products. For example, in mission critical software reliability is most important whereas efficiency is a major concern for time-critical, real-time systems. Interactive end-user software has high requirements to usa-

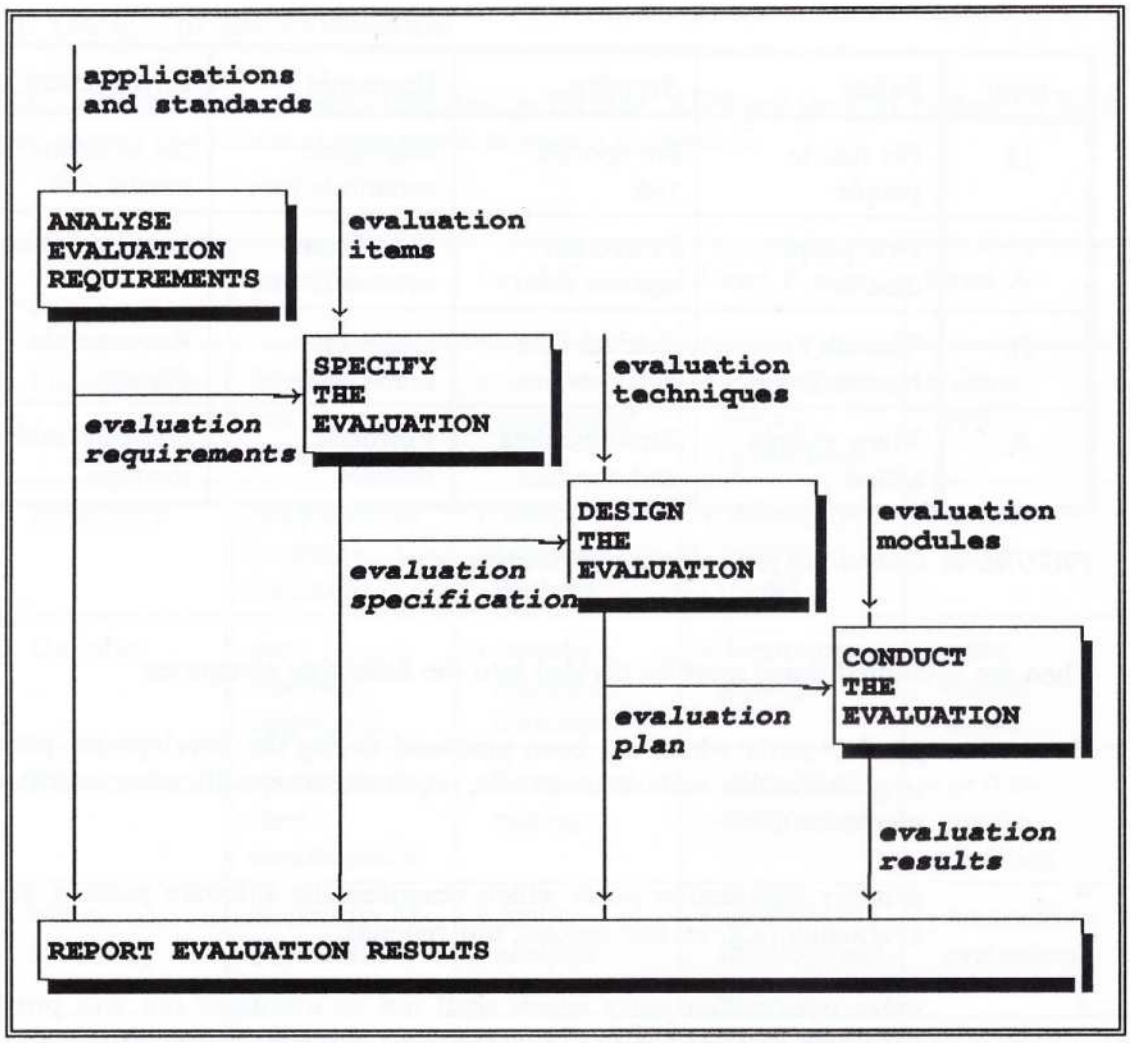


FIGURE 1: The information flow and main activities of the evaluation.

bility. For most types of software the functionality and maintainability characteristics must be evaluated.

The needed stringency of the evaluation will depend on the criticality of the application of the software product. This includes safety, security, economic and environmental considerations. It is part of the analysis to decide the right evaluation level. There are four levels, named A, B, C and D. The levels constitute a hierarchy with A as the highest level (the most stringent evaluation). At the D level less stringent evaluation techniques are used. The selected quality characteristics need not all be evaluated at the same level. Figure 2 gives some guidance on how to select evaluation level.

5. Specification of the evaluation

The evaluation specification is derived from the evaluation requirements. Thus the evaluator needs to perform a careful analysis of the documentation submitted for the evaluation. First the documentation is formally identified. The identification includes document identifier, title, special characteristics and legal implications of their handling such as confidentiallity requirements.

level	Safety	Security	Economic	Environment
D	No risk to people	No specific risk	Negligible economic loss	No environmental risk
C	Few people disabled	Protection against error	Significant economic loss	Local polution
B	Threath to human lives	Critical data and services	Large economic loss	Recoverable damage
A	Many people killed	Strategic data and services	Financial disaster	Unrecoverable damage

FIGURE 2: Guidelines for selecting evaluation level.

Then the submitted items must be divided into the following categories:

* *product parts* which has been produced during the development project (e.g. executable code, source code, requirements specification, user documentation),

* *product information parts* which describes the software product under evaluation (e.g. review reports, test reports),

* *other information parts* which shall not be evaluated but will provide necessary information (e.g. programming language manuals).

This categorisation is useful when deriving quality attributes. Quality attributes are quality properties of the software product. Examples are specific functionality or reliability properties, general properties of the code and its documentation implying good workmanship, or conformance to a particular standard.

The evaluation specification must state unambigously the quality attributes to be evaluated. They should be ordered according to the quality characteristics identified in the evaluation requirements. The evaluation specification comprises

* characteristics to be evaluated,

* the level of evaluation,

* attributes of the product to be evaluated formulated as exact references to the product, to relevant standards and/or supplementary quality requirements.

Finally it must be checked that the product parts, product information parts and other available informations are sufficiently detailed for conducting the evaluation.

6. Design of the evaluation

The software characteristics identified in ISO/IEC 9126 and the four evaluation levels are related to evaluation techniques as shown in figure 3.

Charac-teristic	Level D	Level C	Level B	Level A
Functionality	functional testing (black box)	+ review of documents	+ component testing (glass box)	+ formal proof
Reliability	programming language facilities	+ fault tolerance analysis	+ reliability growth model	+ formal proof
Usability	user interface inspection	+ interface standards conformity	+ laboratory testing	+ user mental model
Efficiency	execution time measurement	+ benchmark testing	+ algorithmic complexity	+ perfor-mance profiling
Main-tainability	inspection of documents (checklists)	+ static analysis	+ analysis of development process	+ traceability evaluation
Portability	analysis of installation	+ program-ming rules conformity	+ constraints evaluation	+ program design evaluation

FIGURE 3: Relation between quality characteristics, evaluation levels and evaluation techniques.

The evaluation techniques identified in the table of figure 3 are used to select a set of evaluation modules. They are chosen from a library of evaluation modules. This involves searching the library for evaluation modules which can be applied in the evaluation of the relevant attribute of the software product. In particular, each selected evaluation module must be applicable to the product part it is to be used on.

At the moment no publicly available evaluation module libraries exists. It is the responsibility of the testing laboratory to develop and maintain such libraries. However, it is envisioned that ISO will take the responsibility of producing a set of standardised evaluatin modules. Evaluation modules are discussed more detailed in sections 9 and 10.

When the relevant set of evaluation modules has been identified the final evaluation plan is produced. This includes allocation of manpower and other resources and fixing the time schedule for performing the evaluation. At this point it is also possible to derive a precise cost estimate for the evaluation.

7. Conducting the evaluation

Conducting the evaluation means applying the selected evaluation modules according to the evaluation plan. Results of applying the modules must be documented, both for the evaluation report and for the internal records of the evaluator. For independent testing laboratories the reporting must be in accordance with ISO Guide 25 [6].

8. Reporting of results

The evaluation report represents the major deliverable to be supplied by the evaluator. The table of contents of the report closely reflects the steps of the evaluation procedure. Each of the main steps is documented separately. The recommended table of contents is the following:

1. Preface
   Identification of producer and evaluator

2. Evaluation requirements
   Product overview, quality characteristics, evaluation level

3. Evaluation specification
   Identification and classification of items, detailed specification

4. Evaluation plan
   Selected evaluation modules, evaluation process planning

5. Evaluation results
   Results of applying the individual evaluation modules

6. Conclusion
   Evaluation result

9. Evaluation modules

The evaluation procedure described relies extensively on evaluation modules. Evaluation modules provide a flexible and structured approach to making software indicators and metrics applicable for evaluation of processes, products and intermediate products as described in the guides to ISO/IEC 9126. The use of evaluation modules produced and validated according to the evaluation module guide ensures that software evaluations can be repeatable, reproducible and impartial.

The definition of an evaluation module is short: "A structured set of instructions and data used for evaluation." An evaluation module specifies an evaluation method applicable to evaluate a quality characteristic and identifies the evidence needed. It also identifies the elementary evaluation procedure and the format for reporting the measurements resulting from the application of the technique.

10. Format of evaluation modules

An evaluation module has six parts each serving different purposes. Part 1 is used for administrative purposes. Part 2 gives the high level selection criteria and part 3 provides information for the selection of evaluation modules in actual evaluations. Part 4

defines the data to be collected and the metrics to be calculated. Part 5 explains how to interpret the measurement results and finally part 6 contains detailed procedures for applying the evaluation module. The format is as follows:

EM1: Identification
Name, Version, Author, Status

EM2: Type of evaluation
Evaluation object, Quality model

EM3: Evaluation requirements
Characteristics, Evaluation level, Technique, Input

EM4: Evaluation specification
Definitions, Principles, Scope, Data, Metrics, Specializations

EM5: Interpretation of results
Mapping of metrics, Acceptance criteria, Reporting, Specializations

EM6: Application procedure
Definitions, Resources, Procedures, Documentation

11. Experiences with software quality evaluations

In the SCOPE project 27 trial evaluations were conducted. The main objectives were to demonstrate the practical feasibility of the evaluation method and to obtain some feedback from real evaluations. The trial evaluations covered a wide range of applications including administrative and technical systems, software tools, communication protocols, and embedded systems. Furthermore, commonly applied development approaches were covered such as standard third generation life-cycles, prototypes and systems developed in 4GL. The set of trial evaluations includes the applications listed in figure 4.

Process control	Accounting
Electronic mail	Traffic control
Medico	Stock management
Phone exchange	Operating systems
Desktop publishing	Management info systems
EPOS	Process monitoring
Picture generation	Compiler
Message handling	Image processing
Graphical analysis	Fire alarm

Figure 4: Application areas in trial evaluations.

The trial evaluations gave useful information on many aspects of the evaluation procedure. The efficiency and effectiveness of the evaluation method was assessed by monitoring the effort incurred from applying the evaluation modules as well as their impact on the result of the evaluation.

In general, all companies involved in the trial evaluations were positive towards the evaluation process, the results, and the experience they had gained through their participation. The experiment showed that it is feasible to carry out software product evaluation according to the evaluators guide.

Based on the evaluators guide DELTA has implemented a software product evaluation service called MicroScope. This service is offered as an independent third party evaluation. As part of MicroScope DELTA has developed and maintains a set of evaluation modules. These evaluation modules are primarily improved versions of those applied in the SCOPE trial evaluations, and they have all been used in commercial evaluations. The current evaluation module library used by MicroScope is shown in figure 5. It is planned to extend the coverage of evaluation modules concurrently with market demands.

Functionality - Requirements specs.	Reliability - Source code
Functionality - Test documentation	
Functionality - Safety	Maintainability - Source code
Functionality - prEN 54	Maintainability - Design document
Functionality - Petri net	
	Efficiency - Source code
Usability - Users manual	
Usability - ISO 9241	Portability - Source code

FIGURE 5: MicroScope evaluation modules.

A MicroScope evaluation of e.g. a fire alarm, which is a typical example of a small evaluation, will require in the order of two to four weeks effort. Such an evaluation will usually require the application of the following five evaluation modules: Usability - User manual, Maintainability - Design document, Maintainability - Source code, Functionality - Safety, and Functionality - prEN 54. The purpose of the last evaluation module is to demonstrate compliance with the standard prEN 54: Components of automatic fire alarm detection systems: Part 2: Control and indicating equipment.

Similar evaluation services have also been set up by other European companies.

12. Conclusion

The approach to software product evaluation described here provides a baseline for improving software quality and for setting up a software product evaluation and certification scheme. It provides a useful supplement to the schemes for process measurement and certification.

The introduction of MicroScope as a commercial service based on this scheme has been very successful. The rapid growth in the number of customers shows that there is a real market need for independent third party quality evaluation of software products.

References

[1] ISO/IEC 9126, *Information technology - Software product evaluation - Quality characteristics and guidelines for their use*, International Standards Organisation, International Electrotechnical Commission, 1991.

[2] CD 9126-6, *Guide to software product evaluation - The evaluators guide*, Editor P. Robert, International Standards Organisation, 1993.

[3] *Guide to producing evaluation modules*, draft technical report, Editor J. Bøegh, International Standards Organisation, 1993.

[4] J. Bøegh, H.-L. Hausen, D. Welzel, *A practitioners guide to evaluation of software*, IEEE Software Engineering Standards Symposium, September 1993.

[5] J. Bøegh, *Standardisation of evaluation modules for software evaluation*, Sixth International Conference on Software Engineering & Its Applications, November 1993.

[6] ISO/IEC Guide 25, *General Requirements for the Competence of Calibration and Testing Laboratories*, International Standards Organisation, International Electrotechnical Commission, 1990.

How to Cope with Software Complexity?

Horst Zuse

Technische Universität Berlin (FR 5-3)
Franklinstraße 28/29
10587 Berlin
Germany
Phone: +49-30-314-73439
Fax: +49-30-314-21103
E-mail-Internet: zuse at tubvm.cs.tu-berlin.de

Abstract

Since the seventieth software measurement has received much attention as a tool supporting software quality. In this paper we give a brief history of software measures, discuss the goal of the ESPRIT Project METKIT, discuss several measures in order to analyze software complexity, discuss some foundations of measurement and show the idea of complexity which is hidden behind software complexity measures. Finally, foundations of the validation of software measures are discussed.

Keywords

Software measures, measurement theory, METKIT, software complexity, history.

1 Introduction

The groundwork for software measures was established in the sixties and mainly in the seventies, and from these earlier works, further results have emerged in the eighties and the beginning of the ninetieth. It may be, that the earliest paper about software complexity was published by Rubey et al. (Rubey, Hartwick, 1968) in 1968.

Today, we agree to the statement of Dieter Rombach, who said at the Eurometrics 1991 in Paris: *we should no longer ask if we should measure, the question today is how.*

For this reason we show some aspects of software measurement. We give a brief overview of the history of software measures, show, how we can analyze software complexity, discuss the term complexity related to software measures and present some foundations of the validation of software measures.

In detail, we consider the following: Section 2 gives a brief overview into the history of software measures, Section 3 presents the philosophy of the ESPRIT Project METKIT which was a three year collaborative research project funded by the European Commission under their ESPRIT II programme from 1989-1992, Section 4 discusses several software measures in order to analyze the complexity of programs and whole software systems, Section 5 shows the foundations of software measurement, Section 6 discusses the idea of software complexity behind the measures and illustrates this with the Measures of McCabe and LOC, Section 7 gives foundations the validation of software measures, Section 8 gives the conclusions and in Section 9 a list of references can be found.

2 History of Software Complexity Measurement

During the past more than 500 software (complexity) measures were proposed by researchers and practitioners, and till today more than 1000 papers about software complexity have been published. An overview of published papers can be found the following books or papers: (Conte, Dunsmore, Shen, 1986), (Grady, Caswell, 1987), (Ejiogu, 1991), (Dumke, 1992) (Mills, 1988), (Perlis, Sayward, Shaw, 1981), (Fenton, 1991), (Fenton, Littlewood, 1990), (Kitchenham, Littlewood, 1989), (Möller, Paulish, 1993), (Shepperd, 1993), (Shepperd, Ince, 1993a), (Jones, 1991), (Goodman, 1992), (Samadzadeh-Hadidi, 1987), (Grady, 1992), (DeMarco, 1982), (AMI, 1992), (Card, Glass, 1990), (Hetzel, 1993), (Shooman, 1983), (Mayrhauser, 1990 (Sommerville, 1992), (Pressmann, 1992), (Zuse, 1991) and (Zuse, 1994a). We only discuss some "milestones" in the development of software (complexity) measures.

The groundwork for software measures was established in the sixties and mainly in the seventies, and from these earlier works, further results have emerged in the eighties and the beginning of the ninetieth.

There is a confusing situation using the terms software "measures" or software "metrics". Using the terminology of measurement theory we use the term "measures". In literature the terms metric and measure are used as synonyms. A metric is here not considered in the sense of a metric space.

The earliest software measure is the Measure LOC, which is discussed and used till today (Park, 1992). In 1974 Wolverton (Wolverton, 1974) made one of the earliest attempts to formally measure programmer productivity using lines of code (LOC). He proposed object instructions per man-month as a productivity measure and suggested what he considered to be typical code rates. The basis of the Measure LOC (Shepperd, 1993), p.3, is that program length can be used as a predictor of program characteristics such as reliability and ease of maintenance. Despite, or possibly even because of, simplicity of this metric, it has been subjected to severe criticism. In 1983 Basili and Hutchens (Basili, Selby, Phillips, 1983) suggested that the Metric LOC should be regarded as a baseline metric to which all other metrics be compared. We should expect an effective code metric to perform better than LOC, and so, as a minimum, LOC offers a "null hypothesis" for empirical

It may be, that the earliest paper about software complexity was published by Rubey et al. (Rubey, Hartwick, 1968) in 1968. In 1979 Belady (Belady, 1979) mentioned a dissertation of Van Emden (Van Emden, 1971). The work of Van Emden was based on the concept of conditional probabilities on the formalism of information theory, and appeared suitable to model complexity of interconnected systems, such as programs built of modules.

Two other software complexity measures (Interval-Derived-Sequence-Length (IDSL) and Loop-Connectness (LC)) were proposed in 1977 by Hecht (Hecht, 1977) and are discussed in (Zuse, 1991), p.221. They are based on the reducibility of flowgraphs in intervals. However, they are not well known. The works of Rubey, Van Emden and the Measures of Hecht have been largely forgotten. However, in 1992 the Measure of Van Emden was used as a basis of a complexity measure by Khoshgoftaar et al. (Khoshgoftaar, Munson, 1992).

In 1977 Gilb (Gilb, 1977) published a book entitled "Tom Gilb: Software Metrics" which is one of the first books in the area of software measures.

The most famous measures, which are continued to be discussed heavily today and which were created in the middle of the seventies are the Measures of McCabe (McCabe, 1976) and of Halstead (Halstead, 1977). McCabe derived a software complexity measure from graph theory using the definition of the cyclomatic number. McCabe interpreted the cyclomatic number as the "minimum number of paths" in the flowgraph. He argued that the minimum number of paths determines the complexity (cyclomatic complexity) of the program: *The overall strategy will be to measure the complexity of a program by computing the number of linearly independent paths v(G), control the "size" of programs by setting an upper limit to v(G) (instead of using just physical size), and use the cyclomatic complexity as the basis for a testing methodology.* The Measures of Halstead are based on the source code of programs. Today the most used Measures of Halstead are the Measures Length, Volume, Difficulty and Effort.

In 1978 software design measures were proposed by Yin et al. (Yin, Winchester, 1978) and Chapin (Chapin, 1979). These measures maybe the first measures which could be used in the software design phase. From 1983 software measures, which can be used in the design phase, were proposed among others by Bowles (Bowles, 1983), McCabe et al. (McCabe, Butler, 1989), Card et al. (Card, Glass, 1990) and Ligier (Ligier, 1989).

In 1979 Albrecht (Albrecht, 1979) proposed the Function Point method. Function points are derived from requirements specification. This method, which can be used in the specification phase of the software life-cycle, is today widely used in USA and UK.

Aslo in 1979, Belady (Belady, 1979) proposed the Measure BAND which is sensitive to nesting.

In 1980 Oviedo (Oviedo, 1980) developed a "Model of Program Quality". This model treats control flow complexity and data flow complexity together. Oviedo defines the complexity of a program by the calculation of control complexity and data flow complexity with one measure.

Based on the works of Stevens, Myers and Constantine (Stevens, Myers, Constantine, 1974) much work has been done to create software complexity measures for the interconnectivity analysis of large software systems. Software systems contain multiple types of interrelations between the components, like data, control, and se-

quencing among others. In 1981 the "Interconnectivity Metric" of Henry and Kafura (Henry, Kafura, 1981) was proposed. This measure is based on the multiplication of the fan-in and fan-out of modules. At the end of the eighties a modification of this measure (Henry, Wake, 1988) was proposed by creating a hybrid measure consisting of a intra-modular measure, like the Measures of Halstead and McCabe and the "Interconnectivity Metric". Other approaches, for example of Selby et al. (Selby, 1992) and Hutchens et al. (Hutchens, Basili, 1985) base on the early works of Myers (Myers, 1976) and Stevens et al. (Stevens, Myers, Constantine, 1974), and propose software measures based on data binding.

In 1981 Harrison et al. (Harrison, Magel, 1981) presented software complexity measures which are based on the decomposition of flowgraphs into ranges. Using the concept of Harrison et al. it is possible to determine the nesting level of nodes in structured and especially unstructured flowgraphs. This is an important extension of the Measures of Dunsmore, Belady, etc. (Zuse, 1991), p.269, p.362, which were only defined for flowgraphs consisting of D-Structures (Dijkstra-Structures). In 1982 Piwowarski (Piwowarski, 1982) suggested a modification of the Measures of Harrison et al. because these measures have some disadvantages, for example unstructured flowgraph can be less complex than structured flowgraphs.

Troy et al. (Troy, Zweben, 1981) proposed in 1981 a set of 24 measures to analyze the modularity, the size, the complexity, the cohesion and the coupling of a software system. Measures for cohesion, which are based on the concept of slices of Weiser (Weiser, 1982), were discussed in 1984 by Emerson (Emerson, 1984), in 1986 by Longworth et al. (Longworth, Ottenstein, Smith, 1986) and in 1991 by Ott et al. (Ott, Thuss, 1991).

In 1981 a Study Panel (Perlis, Sayward, Shaw, 1981) consisting of people of the industry and universities (Victor Basili, Les Belady, Jim Browne, Bill Curtis, Rich DeMillo, Ivor Francis, Richard Lipton, Bill Lynch, Merv Müller, Alan Perlis, Jean Summet, Fred Sayward, and Mary Shaw) discussed and evaluated the current state of the art and the status of research in the area of software measures. During this panel DeMillo et al. (DeMillo, Lipton, 1981) discussed the problem of measuring software compared to other sciences. They discussed the problem of meaningfulness. However, they did not give conditions for the use of software measures as an ordinal and a ratio scale.

In 1984 Basili et al. (Basili, Weiss, 1984) proposed the GQM (Goal-Question-Metric) paradigm. GQM is used for defining measurement goals. The basic idea of GQM is to derive software measures from measurement goals and questions. The GQM approach supports the identification of measures for any type of measurement via a set of guidelines for how to formulate a goal comprehensiblly, what types of questions to ask, and how to refine them into questions. The idea that the use of software measures depends on the view of the humans is also supported by Fenton (Fenton, 1991a), p.253 (called: user-view), and Zuse et al. (Zuse, Bollmann, 1989) (called: a viewpoint of complexity).

Based on articles of Bollmann and Cherniavsky (Bollmann, Cherniavsky, 1981), and Bollmann (Bollmann, 1984), in which measurement theory was applied to evaluation measures in the area of information retrieval systems, in 1985 Bollmann and Zuse (Zuse, 1985), (Bollmann, Zuse, 1985) transfered this measurement theoretic approach to software complexity measures. They used measurement theory, as described by Roberts (Roberts, 1979), Krantz et al. (Krantz, Luce, Suppes, Tversky, 1971) and Luce et al. (Luce, Krantz, Suppes, Tversky, 1 which give conditions for the use of measures. In (Bollmann, Zuse, 1985), (Zuse, 1985), (Zuse, Bollmann, 1987), (Zuse, Bollmann, 1989), (Zuse, 1991), and (Zuse, Bollmann-Sdorra, 1992) the conditions for the use of software complexity measures on certain scale levels, like ordinal, interval or ratio scale were presented. Additionally, measurement theory gives an empirical interpretation of the numbers of software measures by the hypothetical empirical relational system and conditions for concatenation and decomposition operations, which are major strategies in software engineering. This concept is also applied in (Zuse, 1991) for more than 90 software measures. In 1993 followed papers which gave the foundations of prediction and software measures (Bollmann-Sdorra, Zuse, 1993) and about the validation of software measures (Zuse, 1994).
Similar approaches of using measurement theory followed from 1987 by the Grubstake Group (Baker, Bieman, Gustafson, Melton, Whitty, 1987), (Baker, Bieman, Fenton, Gustafson, Melton, Whitty, consisting of Norman Fenton (City University, London), Robin Whitty (CSSE, South Bank Polytechnic, London), Jim Bieman (Colorado State University), Dave Gustafson (Kansas State University), Austin Melton (Kansas State University), and Albert Baker. The Grubstake Group used measurement theory to describe the ranking order of programs as created by software meas-

ures. Measurement theory is also proposed as an appropriated theory for software measures by Fenton (Fenton, 1991), p.16.

In the eighties several investigations in the area of software measures were done by the Rome Air Development Center (RADC) (RADC, 1984). In this research institute the relationships of software measures and software quality attributes (usability, testability, maintainablity, etc.) were investigated. The goal of these investigations is the development of the Software Quality Framework which quantifies both user- and management-oriented techniques for quantifying software product quality.

In 1986 a research project started (Alvey-Project SE/069) entitled "Structured-Based Software Measurement" (Elliott, Fenton, Linkman, Markham, 1988). This project was intended to build on existing research into formal modelling, analysis and measurement of software structure. It was carried out at South Bank Polytechnic's Centre for Systems and Software Engineering in London, UK. Among others, results of this project can be found in Fenton (Fenton, 1991).

From 1989 till 1992, the Project METKIT (Metrics Educational Toolkit 2384) (METKIT, 1993) of the European Community was created. METKIT was a collaborative project part-funded by the European Commission under their ESPRIT programme. The aim of METKIT was to raise awareness and increase usage of software measures within European industry by producing educational material aimed to both industrial and academic audiences. An outcome of METKIT was the book of Fenton (Fenton, 1991) which gives an excellent overview of the area of software measures.

In the Project METKIT and by Fenton (Fenton, 1991), p.44, a simple classification of software measures was introduced. They classified software measures in resource, process and product measures. This classification can be also found in (IEEE Guide, 1989) and (IEEE, 1989a). It should be noted here that in literature many different classifications of software measures can be found. It would be beyond this article to mention all the classification schemes, but some of them can be found in (Zuse, 1991), Chapter 3.

Till the middle of the eighties mostly intra-modular software measures (Measures which are applied to modules or programs) were applied in the coding phase of the software life-cycle. From the middle of the eighties more attention has been directed to the measurement in the early phases of the software life-cycle, like the design phase. The idea of applying software measurement in the early phases of the software life-cycle is to improve software development in the software design phase by a feedback process controlled by software measures in order to get a better implementation of software in the coding phase and a less complicated and expensive maintenance.

In the middle of the eighties, researchers, like Kafura et al. (Kafura, Canning, 1988) and the NASA (NASA, 1984) tried to verify the hypothesis that there exists a correlation between software measures and development data (such as errors and coding time) of software. In order to verify this hypothesis they used the data from the SEL-Lab (NASA, 1981). Similar investigations can be found in (NASA, 1984) and (NASA, 1986). Rombach (Rombach, 1990) summarized some of the results doing measurement in the early phases of the software life-cycle. He points out that there is a high correlation (0.7, 0.8) between the architectural design documents and the maintainability (factors of maintainability). Architectural design can be captured with "architectural measures", (inter-modular measures) like system design and modularity measures. There is a low correlation between "algorithmic measures" (intra-modular measures: code complexity, like McCabe) and maintainability. There is also a high correlation (0.75, 0.8) between "hybrid measures" (algorithmic and architectural measures). From these results the idea is derived to get measurement values of the early phases of the software life-cycle in order to get a less complicated and less expensive maintenance. Other studies can be also found in (Conte, Dunsmore, Shen, 1986), Chapter 4.

Considering the software life-cycle and software measures many researchers proposed desireable properties for software complexity measures. Such desired properties can be found among others in Fenton (Fenton, 1991), p.218, Weyuker (Weyuker, 1988), Kearney et al. (Kearney, Sedlmeyer, Thompson, 1986), and Lakshmanan et al. (Lakshmanan, Jayaprakash, Sinha, 1991). Many of these required properties are contradictory. Some of them are identical with the conditions of the extensive structure in measurement theory (See also Section 4.2). A discussion of some of the desired properties can be found in (Zuse, 1991), Chapter 6.

At the end of the eighties two IEEE-Reports (IEEE Guide, 1989), (IEEE, 1989a) appeared which propose standardized software measures. Unfortunately, most of the discussed software measures in these reports are process measures and measures for the maintenance phase of the software life-cycle. Only some software complexity measures can be found there.

From 1989 the use of factor analysis in order to analyze the properties of existing software complexity measures and new dimensions of program complexity were used by Munson et al. (Munson, Khoshgoftaar, 1989) and Coupla et al. (Coupal, Robillard, 1990).

At the beginning of the nineties software measures for analyzing the complexity of object oriented systems were proposed. Four of the few authors are Bieman (Bieman, 1991), Lake et al. (Lake, Cook, 1992), and Rocacher (Rocacher, 1988), Chidamber et al. (Chidamber, Kemerer, 1993), and Morris (Morris, 1989). However, in the area of object oriented systems it is not clear what makes an object oriented program difficult or easy to understand, to test or to maintain.

3 METKIT

In order to promote the use of software measurement in industry and at unversities, among other ESPRIT-Projects, METKIT (METKIT, 1993) (Metrics Educational Toolkit) was founded in 1989. METKIT was a three year collaborative research project funded by the European Commission under their ESPRIT II programme from 1989-1991.

Other ESPRIT Project dealing with software engineering measurement were:

- **AMI:** Nov. 1990-92 (Applications of Metrics in Industry)
- **MUSIC:** Nov. 1990-93 (Metrics for Usability Standards in Computing)
- **MUSE:** 1987-90 (Software Quality and Reliability Metrics for Selected Domains: Safety Management and Clerical Systems).
- **PYRAMID:** Oct 1990-92 (Promotion for Metrics), Improvement of quality and productivity of European software-intensive systems development and maintenance by the use quantitative methods ... in the application of metrics by 1995.
- **COSMOS:** Febr. 1989-94 (Cost Management with Metrics of Specification)
- **MERMAID:** Oct. 1988-92 (Metrication and Resource Modelling Aid).

The primary objective of METKIT was to promote the use of measurement throughout Europe. The project developed an integrated set of educational materials to teach managers, software developers and academic students how to use measurement to understand, control and then improve software developments.

METKIT has produced, among others, the following material.

1. A set of 17 industrial modules.
   - IMMT: Measurement as a management tool.
   - IISM: Introduction to software engineering measurement.
   - IWDH: What can we measure in software engineering and how?
   - IPSS: Procuring software systems.
   - ISMI: Software engineering measurement in industry.
   - IIMP: How to implement a measurement programme?
   - IECD: Estimating the cost of software development.
   - IPRB: Process benchmarking.
   - ISMQ: Specifying and measuring software quality.
   - ICSP: Case study - setting up a measurement programme.
   - ICEM: Establishing a cost estimation measurement programme.
   - ICES: Cost estimation strategy.
   - IPOM: Process optimization measures.
   - IUSA: Usability assessment.
   - IDFA: Defect analysis.
   - IWBS: The case for a standard work breakdown structure.
   - IFPA: Principles of function point analysis.
2. A set of 7 academic modules.
   - AM0: Introduction to software engineering measurement.
   - AM2: What is measurement?
   - AM3: Principles of software engineering measurement.
   - AM4: Software engineering measurement in industry.
   - AM2.4: Experimental design for software engineering.
   - AM3.2: Software engineering measures and models.
   - AM3.4: PODS: A software engineering measurement case study.
   - AM4.3: Tool sampler.
3. A Computer Aided Instruction System (CAI).

4. A prototype tool sampler

5. A text book: *Fenton, N.: Software Metrics: A Rigorous Approach, Chapman & Hall, 1991,* (Fenton, 1991).

6. A Bibliography.

7. A Term Glossary.

More informations of the Project METKIT can be found in (METKIT, 1993).

4 Software Measures

We now give examples of software measures. In the past more than 500 software measures in order to analyze software complexity, effort for software maintenance, effort for writing programs and estimating project costs were proposed. We discuss some of these measures, like the COCOMO-Model, the Function-Point-Method (FPM), intra- and inter-modular software complexity measures.

4.1 The COCOMO-Model

In 1981 Boehm (Boehm, 1981) introduced the basic COCOMO-model (Constructive-Cost-Model) which is defined as:

$$EFFORT(P) = a\ LOC\ (P)^b,$$

where EFFORT (P) is the effort to write a program P, a is a constant, LOC is the Measure LOC (Lines of Code), b>1 for an increasing and b<1 for a decreasing function. The COCOMO-Model is a model for prediction, it is not only a measure. The predicted variable EFFORT (we call this an external variable) is EFFORT und the measure is LOC. The relation between the measure and the external variable is given by the formula above.

The COCOMO-Model recognizes three types of development environment and provides different variations of the basic model for each environment.

1. The organic environment is essentially that of a small scale, non-bureaucratic project and for this environment the model takes the form:

$$EFFORT(P) = 2.4\ LOC\ (P)^{1,05}.$$

2. The embedded environment is the opposite of the organic in that it relates to a very bureaucratic, tightly controlled and formal organization. For this environment the model takes the form:

$$EFFORT(P) = 3.6\ LOC\ (P)^{1,2}.$$

3. The so-called semi-detached environment is one that falls between the two extremes, and for this the model takes the form:

$$EFFORT(P) = 3.0\ LOC\ (P)^{1,12}.$$

The COCOMO-Model has been criticized several times, one argument is that the COCOMO-Model may be acceptable for predicting coding, neither are capable of dealing with complete software projects. For large software systems, coding is only the 4th greatest element of expense: in rank order, defect removal, paperwork, meetings, and coding are the top cost elements.

4.2 Function-Point Method

In 1979 Albrecht (Albrecht, 1979) suggested the Function-Point Method (FPM), which is a measure based upon a count of "function points" in order to make predictions. We will not explain the FPM in detail because this would be beyond the scope of this paper. We only give an idea how this methods works.

The total number of function points for a specification is given by:

$$UFC = 4*A + 5*B + 4*C + 10*D + 7*E$$

where

A = Number of external input types.

B = Number of external output types.

C = Number of inquires.

D = Number of external files (program interfaces).

E = Number of internal files (i.e. those generated) used and maintained by the program.

To compute the adjusted function points, FP, we need first to compute TCF. Having identified the various types of items, each is given a subjective complexity rating of either simple, average or complex.

The "Unadjusted Function Count" for a Project P can be calculated by

$UCF(P) = \sum_{i=1}^{15}$(Number of items variety i) * (weight)

$UCF(P) = \sum_{i=1}^{15}(NDi*Wi)$

Finally, the adjusted function count FP is derived from UCF by the multiplication of a so-called Technical-Complexity-Factor TCF:

$$FP = UFC * TCF.$$

TCF is calculated by fifteen factors F1-F15. For more informations see (Albrecht A.J, Gaffney, 1983), (Fenton, 1991), p.165 , Shepperd (Shepperd, 1993) or Goodman (Goodman, 1992). Finally, the TCF is calculated as:

$$TCF = 0.65 + 0.1\sum_{i=1}^{15} Fi.$$

The FPM is a widely used method in UK and US.

However, both the COCOMO-Model and the FPM have completely different assumption to predict effort. In (Bollmann-Sdorra, Zuse, 1993) and (Zuse, 1994a) it is shown that the COCOMO-Model assumes a ratio scale for the external variable effort while the Function-Point Method does not assume a ratio scale. Further investigations have to show what are the consequences of these different methods.

4.3 Inter- and Intra-modular Software Measures

We now discuss software measures which are based on source code, flowgraphs and structured charts. An example shall illustrate this.

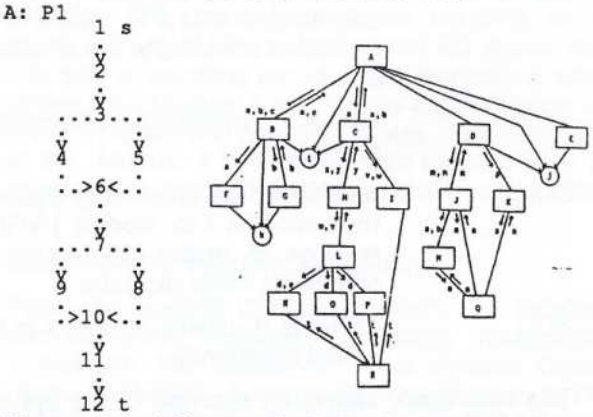


Figure 4.1: A flowgraph P1 and a structure chart D-BM of Bowles (Bowles, 1983).

Measures which are based on modules are called intra-modular measures. Following Page-Jones (Page-Jones, 1988) a module is here a collection of program statements with four basic attributes: input and output, function, mechanics, and internal data. Measures which are based on whole software systems are called inter-modular measures. Modules can be represented by flowgraphs. A flowgraph G=(N,E,s,t) is a directed graph, where N is the set of nodes, E the set of edges, s the start-node and t the exit-node. Every node is reachable from the node s and t is reachable from every node. That means, the flowgraph is connected. The structure chart D-BM represents the calling hierarchy of the modules, shows the coupling between the modules by parameters and the interactions of parameters and global variables among the modules.

We now consider some measures which analyze the calling hierarchy and coupling between the modules.

4.3.1 The Measures of Henry and Kafura

The Measures of Henry et al. (Henry, Kafura, 1981) are well known and were proposed at the beginning of the eighties. They are sensitive to the interactions between modules and the use of global variables. Hence, they can be used as software measures to analyze excessive coupling between modules via parameters, global variables and calls of modules. The Measures of Henry et al. are based on structure charts and use the fan-in and fan-out of modules.

We now give the definition of the Measures of Henry and Kafura, which are based on structure charts. Firstly, we give the definition of fan-in and fan-out, as defined by Henry et al. Fan-in and Fan-out are defined as:

- Fan-out: the number of local flows out a procedure/module plus the number of global data structures from which a procedure retrieves informations.
- Fan-in: the number of local flows into a procedure/module plus the number of global data structures which the procedure updates.

Local flows represent the flow of information to or from a routine through the use of parameters and return values from function calls. Combining these with access to global data structures give all possible flows into or out of a procedure.

The Measures D-FO and D-FI are introduced by the author. The Measure D-FI captures the fan-in and the Measure D-FO captures the fan-out as defined by Henry et al.

D-FI = fan-in of all modules,

D-FO = fan-out of all modules.

The Measure D-INFO has been introduced by Henry et al. The complexity of an entire structured design is calculated by the measure D-INFO:

$$D\text{-}INFO = (D\text{-}FI * D\text{-}FO)^2.$$

It can be shown that the multiplication with **2 is useless in order to compare the complexities of entire software systems. We can use the Measure

$$D\text{-}INFO' = (D\text{-}FI * D\text{-}FO).$$

The advantage of the Measure D-INFO' is that we get smaller numbers.

At the end of the eighties a modification of this measure (Henry, Wake, 1988) was proposed by creating a hybrid measure consisting of a intra-modular measure, like the Measures of Halstead (See Section 4.4) and McCabe and the Measure D-INFO. The hybrid Measure D-INFO-LOC is defined as:

$$D\text{-}INFO\text{-}LOC = LOC * (D\text{-}FI * D\text{-}FO)^2.$$

In general we can say that the Measures D-INFO and D-INFO' are very rough measures because D-FI and D-FO do not distinguish the types of fan-in and fan-out.

For this reason we propose the Measures of Bowles which allow a more detailed analyzation of the coupling between modules and the access to global variables.

4.3.2 Measures of Bowles

Bowles (Bowles, 1983) proposed in 1983 an interesting approach for calculating the complexity of single modules and the entire software system. Bowles considered the coupling of modules and the connections of modules by global variables.

Bowles (Bowles, 1983) considers the complexity at two structural levels.

1. The first level is the internal complexity of a module, that means in our terminology the intra-module complexity.
2. The second level is the complexity of the relationships between a given module and all other modules in the system. This is the inter-modular complexity of a software system.

In the next Section we consider the module level.

4.3.2.1 Complexity of a Single Module

The complexity CM of an individual module is calculated by the following factors.

- a = number of formal parameters.
- b = number of global variables shared with superordinate modules.
- c = number of parameters passed between this module and its subordinate modules.
- d = number of global variables shared with subordinates.

Finally, the complexity CM of a module is calculated with the formula

$$CM = 1 - 1 / (1 + a + 2b + c + 2d).$$

This measure takes into account the increased difficulty associated with remembering information associated with global data and the effort to understand subroutines. The Initial Scope complexity IS of a module is defined as

$$IS = 1a + 2b + 1c + 2d.$$

The Measure D-B-IS is the sum of the Initial Complexity of each module m:

$$D\text{-}B\text{-}IS = \sum_{m \in M} IS_m.$$

The Measure D-B-IS can be interpreted as a measure which analyzes the complexity of a module related to the environment of the system.

4.3.2.2 Complexity of the Entire Software System

We now consider the software complexity measures at the system level.

System Complexity Matrix SC

Bowles now introduces a system complexity matrix SC = f(P,G), where P is the set of parameters, and G is the set of global variables. For the matrix SC the following holds:

SC	is the System Complexity matrix and its size is M x M, where M is the set of modules in the structure chart.
SCi,j	shows the relationship of module i to module j.
SCi,j =	number of parameters p passed from module i to module j + 2 * number of global data items g shared by these modules.
SCi,j	for i=j is 0, unless module i is invoked recursively.

The next figure shows an example of the system complexity matrix SC for the structure chart D-BM.

STRUCTURE CHART.....: D-BM
BASIC MODULE COMPLEXITY MEASURES OF BOWLES 1983:

#	MODULENAME	LV	a	b	c	d	IS	CM
1	A	1.00	0.00	0.00	5.00	4.00	13.00	0.93
2	B	2.00	3.00	1.00	2.00	2.00	11.00	0.92
3	C	2.00	2.00	1.00	4.00	0.00	8.00	0.89
4	D	2.00	0.00	1.00	3.00	0.00	5.00	0.83
5	E	2.00	0.00	1.00	0.00	0.00	2.00	0.67
6	F	3.00	1.00	1.00	0.00	0.00	3.00	0.75
7	G	3.00	1.00	1.00	0.00	0.00	3.00	0.75
8	H	3.00	2.00	0.00	2.00	0.00	4.00	0.80
9	I	3.00	2.00	0.00	1.00	0.00	3.00	0.75
10	J	3.00	2.00	0.00	3.00	0.00	5.00	0.83
11	K	3.00	1.00	0.00	1.00	0.00	2.00	0.67
12	L	3.00	2.00	0.00	4.00	0.00	6.00	0.86
13	M	4.00	2.00	0.00	1.00	0.00	3.00	0.75
14	N	5.00	2.00	0.00	1.00	0.00	3.00	0.75
15	O	5.00	1.00	0.00	1.00	0.00	2.00	0.67
16	P	5.00	1.00	0.00	1.00	0.00	2.00	0.67
17	Q	5.00	1.00	0.00	0.00	0.00	1.00	0.50
18	R	6.00	1.00	0.00	0.00	0.00	1.00	0.50

Figure 4.2: The values of the discussed measures above related to the structure chart D-BM.

The explanation of the column headings is as follows: LV is the level of the module in the structure chart D-M, A, B, C, D, IS and CM are the dis-

cussed measures above for each module of the structure chart D-BM.

STRUCTURE CHART.....: D-BM
COMMUNICATIONS BETWEEN THE MODULES:

#	MODULE1	MODULE2	PARAM	GLOB	SC
1	A	B	3	1	5
2	A	C	2	1	4
3	A	D	0	1	2
4	A	E	0	1	2
5	B	C	0	1	2
6	B	F	1	1	3
7	B	G	1	1	3
8	C	H	2	0	2
9	C	I	2	0	2
10	D	E	0	1	2
11	D	J	2	0	2
12	D	K	1	0	1
13	F	G	0	1	2
14	H	L	2	0	2
15	I	R	1	0	1
16	J	M	2	0	2
17	J	Q	1	0	1
18	K	Q	1	0	1
19	L	N	2	0	2
20	L	O	1	0	1
21	L	P	1	0	1
22	M	Q	1	0	1
23	N	R	1	0	1
24	O	R	1	0	1
25	P	R	1	0	1

Figure 4.3: The system complexity matrix SC for the structure chart D-BM.

The table above shows the values of the matrix SC related to the structure chart D-BM. It also shows the number of parameters (PARAM) shared among MODULE1 and MODULE2, the number of global variables (GLOB) and the Measure SC, which shows the coupling of a MODULE1 to MODULE2. Column SC shows the intensity of coupling between the modules. It also shows that Module A has a very high coupling to other modules and global variables. For example, in # 1, Module A is coupled with module B by 3 parameters, 1 global variable and the value of SC is 5.

Measures for System Complexity

From the System Complexity matrix SC Bowles derives some software complexity measures. These are the Measures Entire System Complexity D-B-SCC, Average System Complexity D-B-ASC, Minimum System Complexity D-B-MI and Maximum System Complexity D-B-MA. M is the set of modules and m is an individual module. The measures are defined as:

$$\text{D-B-SSC} = \sum_{i=1}^{|M|} \sum_{j=1}^{|M|} SCi,j$$

$$\text{D-B-ASC} = \sum_{i=1}^{|M|} \sum_{j=1}^{|M|} SCi,j/|M|$$

D-B-MA = MAX(SC),

D-B-MI = MIN(SC).

Aggregated System Complexity Measure AS:

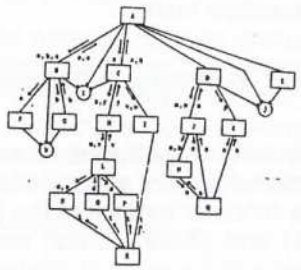
Additionally, Bowles introduces the so-called Aggregated System complexity AS(m) which is the complexity of each module m related to the other modules dependent on the use of global variables and parameters among the modules. The idea behind the Measure AS is the following:

- The complexity AS of a module m is the sum of the non-normalized (initial) complexities IS of the modules in the structure chart which communicate via parameters and global variables among each other.
- The AS complexity may be used as a guide to detect subsystems which are more complex than the average subsystem. Modules with high fan-out will be detected, as will collections of modules which are highly coupled, contrary to the edicts of structured design.
- The Measure AS can be easily computed and can be used in the design phase to identify subsystems which should be simplified before coding.

Considering the conditions above, the Measure Aggregated System complexity AS(m) is defined for each module m∈ M as:

$$AS(i) = \sum_{m \in M} k_m * IS(m),$$

where k_m=0 in the case that module i is not connected via global variables or parameters with modules m∈ M, else k_m=1. The next figure shows the complexity AS of each module for the structure chart D-BM.



STRUCTURE CHART.....: D-BM
AGGREGATED COMPLEXITIES:

#	MODULE	IMOD	IS	AS	AGGR—MODULES
1	A	5	13	39	A B C D E
2	B	4	11	25	B C F G
3	C	3	8	15	C H I
4	D	4	5	14	D E J K
5	E	1	2	2	E
6	F	2	3	6	F G
7	G	1	3	3	G
8	H	2	4	10	H L
9	I	2	3	4	I R
10	J	3	5	9	J M Q
11	K	2	2	3	K Q
12	L	4	6	13	L N O P
13	M	2	3	4	M Q
14	N	2	3	4	N R
15	O	2	2	3	O R
16	P	2	2	3	P R
17	Q	1	1	1	Q
18	R	1	1	1	R

SUM OF AGGREGATED COMPLEXITY: 77.00
AVERAGE AGGREGATED COMPLEXITY: 2.61
MINIMAL AGGREGATED COMPLEXITY: 1
MAXIMAL AGGREGATED COMPLEXITY: 39

MEASURES OF BOWLES

	MEASUREMENT VALUES
	D-BM
D-B-SA	24.000
D-B-SB	6.000
D-B-SC	29.000

```
D-B-SD                    6.000
D-B-IS                   77.000
D-B-SSC                  47.000
D-B-AV                    2.611
D-B-MI                    1.000
D-B-MA                   39.000
```

Figure 4.4: Aggregate System Complexity AS calculated for each module m of the structure chart D-BM.

MODULE is the considered module, IMOD is the number of modules which are related to MODULE via parameters or/and global variables. IS is the Initial System complexity of each module. AS is the aggregated system complexity of each module. AGGR--MODULES are the names of the modules which are connected with MODULE. For example, Module A is connected with the modules A, B, C, D and E via global variables and parameters. The Initial Complexity IS of module A is IS(A)=13 and the Aggregated System Complexity AS(A)=39.

The example above shows that Module A has the strongest coupling to other modules via parameters and global variables. However, Module H has a low module complexity of 4 but a high complexity AS(H) related to the other modules. The reason is a high coupling via parameters and global variables. The values of the Measure AS can be used to figure out complicated modules and modules which have complicated interactions among each other. These modules can be analyzed again by an inspection team.

4.4 Intra-modular Software Complexity Measures

We now present intra-modular software measures which are based on modules. For a study in detail of the presented intra-modular measures the book of Zuse (Zuse, 1991) and (Zuse, 1992a) are recommended.

The most famous measures are the Measures of Halstead.

4.4.1 Measures of Halstead

We propose four Measures of Halstead (Halstead, 1977). in order to measure the complexity of program based on the source code. The Measures of Halstead are defined as:

n1	Number of distinct operators,
n2	Number of distinct operands,
N1	Total number of operators,
N2	Total number of operands.

Measure Program Length N (HALST-N)

$N = N1 + N2$

Measure Program Vocabulary n (HALST-VC)

$n = n1 + n2$

Measure Volume V (HALST-V)

$V = N * LOG2\ n$

$V = (N1+N2)\ LOG2\ (n1+n2)$

Measure Difficulty D (HALST-D)

$D = (n1/2) * (N2/n2)$

Measure Effort E (HALST-E)

$$E = \frac{n1 * N2\ (N1+N2) * LOG\ 2\ (n1+n2)}{2 * n2}$$

and that is:

$E = D * V.$

Example 4.1
We present an example as presented by Halstead (Halstead, 1977):

```
Program                          N1   N2
-------------------------------------------
IF A=0 THEN
   LAST: DO;                      3    3
   GCD=B;RETURN;END;              4    2
IF B=0 THEN DO;                   3    2
   GCD=A;RETURN;END;              4    2
   HERE: G=A/B;                   3    3
   R=A-B*G;                       4    4
IF R=0 THEN GOTO LAST;            4    2
   A=B;B=R;GOTO HERE;             6    4

n1=9; n2=6; N1=31; N2=21;
N = 52
V = 203.16
D = 15.75
E = 3199.74
```

Figure 4.5: An example of the calculation of the Measures N, V, D and E as given by Halstead (Halstead, 1977).

The Measures of Halstead can be seen as complexity measures in general because they are based on source code.

4.4.2 Intra-modular Software Complexity Measures Based on Flowgraphs

We now give the definitions of intra-modular software complexity measures which are based on flowgraphs of programs. We denote D as the set of decision nodes, and M as the set of primes which are subflowgraphs with only one-entry and one-exit.

4.4.2.1 Measures LOC, PRIMES and the Measures of McCabe

Measure LOC

The Measure LOC is well known and is defined as:

$LOC = |N|$

N is here the set of nodes in a flowgraph. The very important question is: what is a line of code? This question is discussed by Park et al. (Park, 1992). A good overview of size measurement (LOC is a size measure) can be found by MacDonell (MacDonell, 1991). Following MacDonell five sub-techniques are widely employed to indicate understandability or complexity in lines of code:

1. Total lines (TLOC) which include all lines excluding blank lines.
2. Executable lines (ELOC) which quantifies all occurences of program verb clauses.
3. Non-commentary lines (NLOC) which count all lines except blank and comment lines.
4. Lines as separated by code delimiters
5. Statement count - this is usually has the same form as the ELOC or delimiter separated counting method.

Other definitions of a line of code can be found in Conte et al. (Conte, Dunsmore, Shen, 1986):
A line of code is any line of program text that is not a comment or blank line, regardless of the number of statements or fragments of statements on the line. They specifically includes alle lines containing program headers, declarations, and executable and non executable statements.

Levitin (Levitin, 1986) considers four types measuring the size of software:

- Source lines of code.
- The number of statements.
- Software Science of Halstead: Length and Volume (Halstead, 1977).
- The number of tokens in the program.

Another modification of the Measure LOC is the Measure SBSM-V of Bache et al. (Fenton, 1991), (Zuse, 1991).

Measures of McCabe:

The Measures MCC-V, MCC-V2, MCC-D, MCC-EV, MCC-EV0 of McCabe (McCabe, 1976) are based on the number of edges, the number of nodes and the number of primes (1-entry and 1-exit subflowgraphs (Zuse, 1991), p. 296) in a flowgraph.

McCabe (McCabe, 1976) used the definition of the cyclomatic number for strongly connected flowgraphs as a software complexity measure. McCabe denoted it as the cyclomatic complexity. The original Measure MCC-V of Mccabe is defined as:

MCC-V(G) = |E| - |N| + 2p, where

p is the number of connected components and G a flowgraph. For the calculation of the complexity of a module it holds p=1. Connected components are in the case of software complexity measurement the number of subprograms in a software system. That means, intra-modular complexity of a single module is:

MCC-V(G) = |E| - |N| + 2.

McCabe presented a modified definition of the Measure MCC-V and that is Measure MCC-V2. The Measure MCC-V2 of McCabe is a strictly monotonic transformation of the Measure MCC-V and it is additive to a sequential concatenation BSEQ of flowgraphs (Zuse, Bollmann, 1989), (Zuse, 1991), p.167. It is defined as:

MCC-V2(G) = |E| - |N| + 1,

MCC-V2(G)=MCC-V(G) - 1

For structured programs (Decision nodes with an out-degree of 2) the result of the Measures MCC-V is equal to MCC-D which is defined as:

MCC-D(G) = |D| + 1.

Following the discussion of the Measures of McCabe in (Zuse, 1991), p.151, the Measure MCC-D'=MCC-D - 1 can be used as a ratio scale. However, the Measure MCC-D is identical to the Measure DEC, which is defined as:

DEC = |D|.

In order to capture unstructured components of a flowgraph, McCabe defined the Measure "Essential Complexity" MCC-EV, which is sensitive to the unstructuredness of flowgraphs. Firstly, we introduce the Measure PRIMES because the Measure MCC-EV is based on this measure.

PRIMES = |M|,

where M is the set of primes in a flowgraph.

Primes are 1-entry and 1-exit subflowgraphs (Zuse, 1991), p.296.

The Measure MCC-EV is defined as:

MCC-EV(G) = |D| - |M| + 1.

The author introduces the Measure MCC-EV0 which is defined as:

MCC-EV0(G) = |D| - |M|.

The Measure MCC-EV0 is additive related to a sequential concatenation operation BSEQ.

4.4.2.2 Measures for Nesting

We now introduce measures which capture nested constructs in flowgraphs.

Measure SCOPE

The Measure SCOPE was introduced by Harrison et al. (Harrison, Magel, 1981). It is defined by the sum of the adjusted complexity of the nodes. It is sensitive to nested structures in unstructured and structured flowgraphs. The Measure SCOPE is defined as:

SCOPE = |N| + $\sum_{d=1}^{|D|}$ Cardinality of the Ranges(d).

Ranges are explained below.

Measure PEN

The Measure PEN is defined as:

PEN = SCOPE - |N|.

That means, only nodes which are predicated by decision nodes contribute to the complexity of the flowgraph.

Measure SCOPE-MN

The Measure SCOPE-MN captures the maximal nesting depth of a node.

Measure SCOPED:

The Measure SCOPED captures the sum of the nesting levels of the decision nodes in a flowgraph.

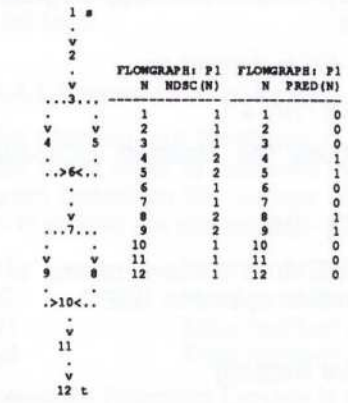
Measure AVG

The Measure AVG of Howatt and Baker capture the average nesting depth in the flowgraph: It is defined as

AVG = SCOPE/|N|.

We now give some examples of the Measure SCOPE.

```
A: P1
N: P1
R: Author
C: Example
SU: S
```



	FLOWGRAPH: P1 N	NDSC(N)	FLOWGRAPH: P1 N	PRED(N)
	1	1	1	0
	2	1	2	0
	3	1	3	0
	4	2	4	1
	5	2	5	1
	6	1	6	0
	7	1	7	0
	8	2	8	1
	9	2	9	1
	10	1	10	0
	11	1	11	0
	12	1	12	0

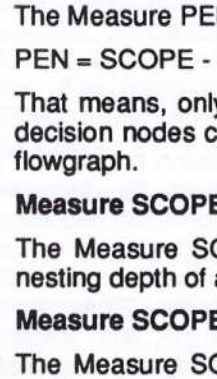
MEASUREMENT VALUES

	P1
LOC	12.000
SCOPE	16.000
SCOPE-MN	2.000
SCOPE-D	0.000
SCOPE-NL	0.000
SCOPED	2.000
PEN	4.000
NL	0.000

Figure 4.6: Nesting depth of the nodes of flowgraph P1. NDSC(N) is the nesting depth of node n for the Measure SCOPE and PRED(n) is the nesting depth of node n for the Measure PEN.

The next example shows the nesting depths of the nodes for an unstructured flowgraph P6.

```
A: P6
N: P6
R: Author
C: Example
SU: U
```



	FLOWGRAPH: P6 N	NDSC(N)	FLOWGRAPH: P6 N	PRED(N)
	1	1	1	0
	2	1	2	0
	3	1	3	0
	4	2	4	1
	5	2	5	1
	6	1	6	0
	7	1	7	0
	8	3	8	2
	9	2	9	1
	10	2	10	1
	11	2	11	1
	12	1	12	0

MEASUREMENT VALUES

	P6
LOC	12.000
SCOPE	19.000
SCOPE-MN	3.000
SCOPE-D	1.000
SCOPE-NL	1.000
SCOPED	4.000
PEN	7.000
NL	3.000

Figure 4.7: Nesting depth of the nodes of flowgraph P6.

The next example shows the nesting depths of the nodes for an unstructured flowgraph P6F.

```
A: P6F
N: P6F
R: Author
C: Example
SU: U
```


	FLOWGRAPH: P6F N	NDSC(N)	FLOWGRAPH: P6F N	PRED(N)
	1	1	1	0
	2	1	2	0
	3	1	3	0
	4	2	4	1
	5	2	5	1
	6	1	6	0
	7	1	7	0
	8	2	8	1
	9	2	9	1
	10	3	10	2
	11	1	11	0
	12	1	12	0

MEASUREMENT VALUES

	P6F
LOC	12.000
SCOPE	18.000
SCOPE-MN	3.000
SCOPE-D	0.000
SCOPE-NL	0.000

```
SCOPED                    4.000
PEN                       6.000
NL                        0.000
```

Figure 4.8: Nesting depth of the nodes of flowgraph P6F.

There are different complexities between the flowgraphs P6 and P6F. The reason is that P6 has a backward edge from the nodes 11 to 8 and flowgraph PF has a forward edge.

4.4.2.3 Measures for Loops

We now discuss software measures which analyzes loops in flowgraphs.

Measure MWD-HB

The Measure MWD-HB is an extension of the Measure MCC-V2 (Zuse, 1991), p.234, and is defined as:

MWD-HB = MCC-V2 + |L|,

where L is the set of loops in the flowgraph.

Measure SCOPE-NL

The Measure SCOPE-NL captures the number of loops with one-entry and one-exit in a flowgraph and is defined as:

SCOPE-NL = |L|.

Measure NL

The Measure NL is similar to the Measure PEN. The difference to the Measure PEN is that are only nodes contributes to the complexity if they lie in a loop. It holds PEN=NL if are only loops in a flowgraph.

We present three example with the flowgraphs P6, P8 and P4.

```
A: P6
N: P6
R: Author
C: Example
SU: U
```

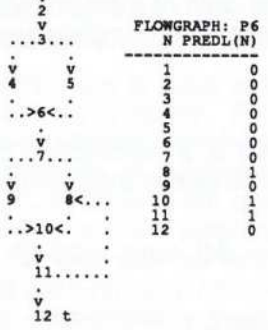


Figure 4.9: Flowgraph P6. PREDL shows of how many loop predicates a node n is predicated. For example, the nesting level related to a loop predicate of the nodes 8, 10 and 11 is 1, that means this node is predicated by one loop-predicate. It holds here: NL(P6)=3.

```
A: P8
N: P8
R: Author
C: Example
SU: U
```

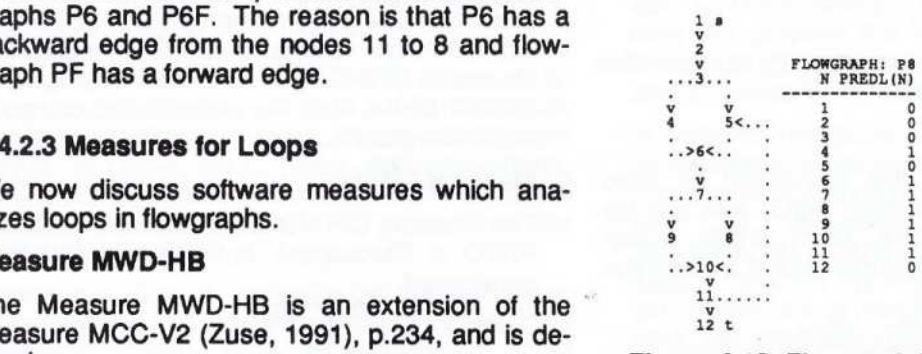


Figure 4.10: Flowgraph P8. It holds: NL(P8)=6.

```
A: P4
N: P4
R: Author
C: Example
SU:
```

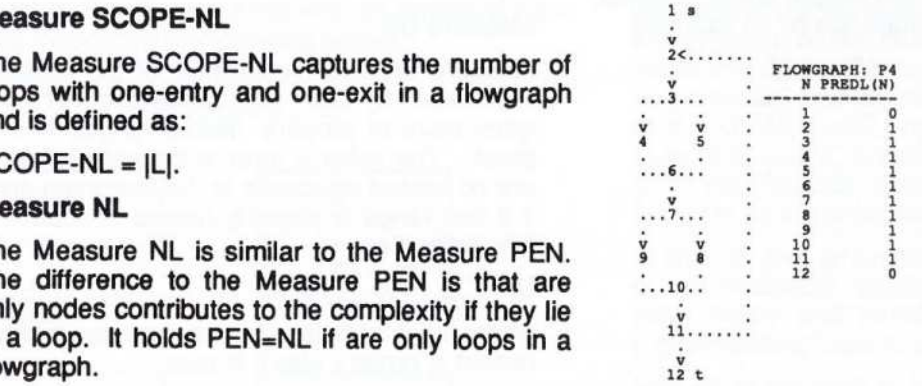


Figure 4.11: Flowgraph P4. It holds: NL(P8)=10.

Measure SCOPE-D:

The Measure SCOPE-D captures structured and unstructured loops. Unstructured loops are such with more than one entry to the loop body. It holds always: SCOPE-D ≥ SCOPE-NL. If holds SCOPE-D = SCOPE-NL then there are only structured loops in the flowgraph.

4.4.2.4 Measures for Unstructuredness

We now discuss measures which capture unstructuredness. As a basis for measures for unstructuredness the concept of overlapping ranges is considered. Ranges were introduced with the Measure SCOPE. The Measures PIWO, UN, RU, UOV, CPIWO and MCC-EV0 analyze nested and unstructured constructs in a flowgraph.

Measure UOV

The Measure UOV counts the number of overlapping ranges (see the examples below) and is defined as:

UOV = |OV|,

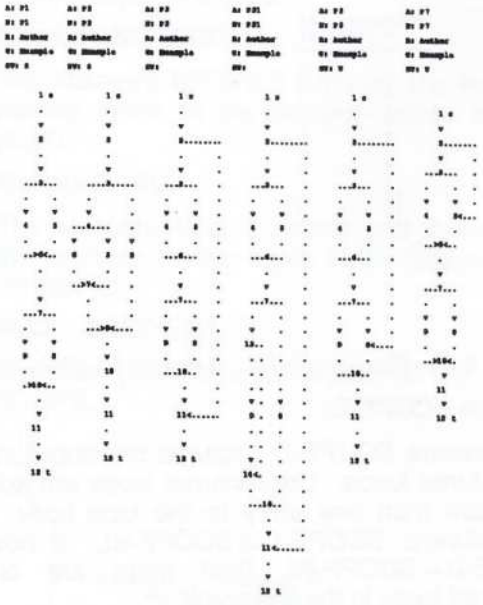
where OV is the set of overlapping ranges. See also (Zuse, 1991), p.458.

Measure PIWO

The Measure PIWO was introduced by Piwowarski in 1982 (Piwowarski, 1982) and are discussed in detail in (Zuse, 1991), p.458. It captures nested and unstructuredness in a flowgraph. It is defined as:

$$PIWO = MCC\text{-}D + \sum_{i=1}^{|D|}\sum_{j=1}^{|D|}P(i,j),$$

where MCC-D is a Measure of McCabe and P(i,j) are pairs of ranges. P(i,j) has the value one if one range is properly nested in another range, and P(i,j) has the value 2 if two ranges are overlapped, else it is zero. We illustrate this by an example.



MEASURES OF PIWOWARSKI

MEASUREMENT VALUES

	P1	P2	P3	P31	P5	P7
DEC	2.00	2.00	3.00	4.00	3.00	3.00
MCC-D	3.00	3.00	4.00	5.00	4.00	4.00
PRIMES	2.00	2.00	3.00	4.00	2.00	2.00
PIWO	3.00	4.00	6.00	9.00	7.00	6.00
UN	0.00	1.00	2.00	4.00	1.00	0.00
UOV	0.00	0.00	0.00	0.00	1.00	1.00
CPIWO	0.00	1.00	2.00	4.00	2.00	2.00

Figure 4.12: Illustration of the Measure of Piwowarski and a comparison to some other measures.

Flowgraphs P1, P2, P3, and P31 consist of properly nested structures. The Measure UOV, PIWO, CPIWO and UN show the differences of the structure of the flowgraphs. Flowgraphs P5 and P7 consist of overlapped ranges (Value of Measure UOV is 1). In flowgraph P5 the ranges of decision nodes 2 and 7 are overlapped. Overlapping ranges cause unstructuredness in a flowgraph. In flowgraph P7 ranges 2 and 3 are overlapped. The difference of the values of the Measures PRIMES and DEC of the flowgraphs P5 and P7 cause from the unstructured components in the flowgraphs.

Measure CPIWO

The Measure CPIWO is derived from the Measure PIWO of Piwowarski, but only the part P(i,j) is considered.

$$CPIWO = \sum_{i=1}^{|D|}\sum_{j=1}^{|D|}P(i,j).$$

The Measure CPIWO and UN should be considered together. The values of both measures are identical in the case that the flowgraph consists only of properly nested constructs.

Measure UN

Measure UN is a modification of the Measure of Piwowarski by Zuse (Zuse, 1991), p.491. It analyzes pairs of properly nested ranges in a flowgraph. The value is zero in the case when there are no nested structures in the flowgraph and it is 1 if one range is properly nested in another one. It is defined as:

$$UN = \sum_{i=1}^{|D|}\sum_{j=1}^{|D|}N(i,j),$$

where N(i,j) increases by one if range i is properly nested in range j, else it is zero.

Measure RU

The Measure RU is introduced by the author and analyzes unstructuredness in flowgraph. It is defined as:

RU = MCC-EV0/MCC-V.

The value is zero in the case of a structured flowgraph and 1 in the case of a complete unstructured flowgraph.

Measure MCC-EV0

The Measure MCC-EV0 is a modification of the Measure MCC-EV of McCabe by the author and it is defined as:

MCC-EV0(G) = |D| - |M|, and MCC-EV was originally defined as:

MCC-EV(G) = |D| - |M| + 1.

The Measure MCC-EV0 gives the value zero for a structured flowgraph.

5 Foundations of Measurement

In this Section we want to introduce the central idea of measurement and give a brief introduction in measurement theory which deals with the connection of empirical and numerical conditions by a homomorphism. We present measurement as it is seen by Roberts (Roberts, 1979), Krantz et al. (Krantz, Luce, Suppes, Tversky, 1971) and Luce et al. (Luce, Krantz, Suppes, Tversky, 1990) very briefly. In (Zuse, 1991), (Zuse, Bollmann-Sdorra, 1992), (Zuse, 1992a) and (Zuse, 1994) the application of measurement theory to software metrics is described in detail.

Firstly, we give a definition of measurement: Measurement is a mapping of empirical objects to numerical objects by a homomorphism. That means, measurement is based on a homomorphism what is also called a representation.

In 1988 Kriz (Kriz, 1988) showed that measurement is always connected with an empirical view. Kriz introduced the following picture.

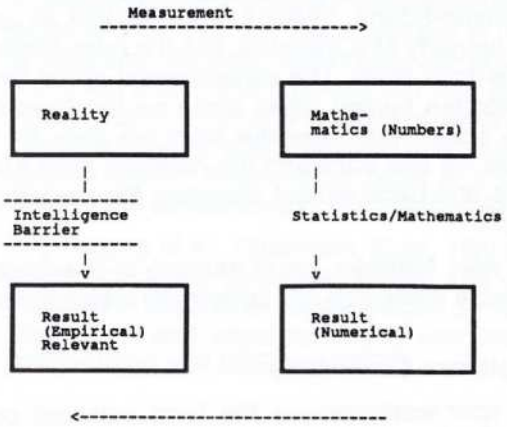


Figure 5.1: The measurement process as presented by Kriz. The empirical and formal relational systems are explained below.

Users want to have relevant empirical results of problems in reality. For example, users want to have relevant empirical statements about the complexity of programs. However, our human brain, in many of the cases, is not able to produce directly relevant empirical results. An exception is, for example, the length of wooden boards. In this case humans can make clear relevant empirical statements. Such statements could be: wooden board A is longer than wooden board B. We can make this statement without using a measure.

However, considering the complexity of programs, the human brain is very often not able to make such statements. The relevant empirical statements related to software complexity can change over the time and people have different ideas of complexity. In many cases the human brain is unable to make relevant empirical decisions. Kriz calls this problem the "intelligence barrier". That means, in many cases, the human brain is not able to reduce informations without certain help.

In order to overcome the problem of the intelligence barrier measurement is introduced. Measurement is a mapping of empirical objects (Reality) to numerical objects (Mathematics (Numbers) by a homomorphism. Mathematics is used to process the informations. Doing this we get mathematical results ("Result (Numerical)". Now, the important step is to give the mathematical results an empirical meaning or empirical interpretation in order to be able to make relevant statements of the objects in reality. The most important point of measurement is to give an interpretation of the numbers. In this case without an interpretation of the numbers it is not possible to make empirical statements. Measurement theory, as presented by Roberts (Roberts, 1979), Krantz et al. (Krantz, Luce, Suppes, Tversky, 1971) and Luce et al. (Luce, Krantz, Suppes, Tversky, 1990) gives the (relevant) empirical interpretation of the numbers by empirical conditions.

In order to give an empirical relevant interpretation of the numerical results we introduce measurement theory and translate numerical conditions (i.e. measures) back to empirical conditions.

First of all we want to introduce the notion of an empirical, a numerical relational system and a scale. Let

$\mathbf{A} = (A, \bullet\geq, o)$

be an empirical relational system, where A is a non-empty set of empirical objects (in our case programs, flowgraphs or structure charts), $\bullet\geq$ is an empirical relations on A (in our case: equal or more complex) and o a binary operation on A

According to Luce et al. (Luce, Krantz, Suppes, Tversky, 1990), p.270, we assume for an empirical relational system **A** *that there is a well-established empirical interpretation for the elements of* **A** *and for each relation Si of* **A**. We also assume the same for the binary operations.

Let further

$\mathbf{B} = (\Re, \geq, +)$

be a formal relational system, where $\Re$ are the real numbers, $\geq$ a relation on $\Re$, and + a binary operation on $\Re$.

A **measure** μ is a mapping $\mu: A \rightarrow B$ such that the following holds for all $a,b \in A$:

$$a \bullet\geq b \Leftrightarrow \mu(a) \geq \mu(b)$$
and
$$\mu(a \circ b) = \mu(a) + \mu(b)$$

Then the Triple (**A, B,** μ) is called a **scale.** According to this definition we see that measurement assumes a homomorphism. A homomorphism. preserves all relations between the empirical and numerical relational system.

We see that a scale does not consist only of numbers. It is more. A scale consists of a homomorphism between the relational systems **A** and **B**.

There is very often a confusion between scales and scale types. The definition of a scale was given above. A scale type is defined by admissible transformations. Common scale types are the following:

Name of the Scale	**Transformation g**
Nominal Scale	Any one to one g
Ordinal Scale	g: Strictly increasing function
Interval Scale	$g(x) = a x + b, a > 0$
Ratio Scale	$g(x) = a x, \quad a > 0$
Absolute Scale	$g(x) = x$

Figure 5.2: Scale types of real scales. It is a hierarchy of scale types. The lowest one is the nominal scale and the highest one is the absolute scale.

The table above can be found in many books and papers which deal with software measures, examples are (Card, Glass, 1990), p.116, (Conte, Dunsmore, Shen, 1986), p.129, (Fenton, 1991), p.30, (Mayrhauser, 1990), p.561.

However, the major question for the user is: how does he know what scale type is assumed using measures and what are the conditions for the use of a measure on a certain scale level. Or equivalently, how does a measure and reality look like which creates numbers which can be transformation by a certain admissible transformation of scales.

We now illustrate this with the measurement of the length of wooden boards.

Empirical Relational System	**Formal Relational System**
Wooden Boards	$\Re$ (Real Numbers)
Relation: equal or longer than: $\bullet\geq$	Equal or greater than: $\geq$

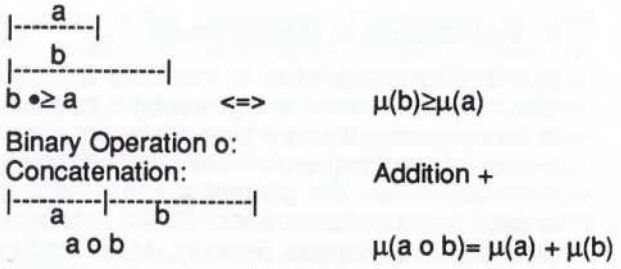


Figure 5.3: Example of measuring the length of wooden boards a and b.

If we measure the length of wooden boards with a ruler, then we have a homomorphism and an additive homomorphism. The homomorphism is

$$b \bullet\geq a \Leftrightarrow \mu(b) \geq \mu(a)$$

If the measurement of the length of wooden boards would not be a homomorphism, our world would crash.

Considering the example above, we have the empirical relation $\bullet\geq$ "equal or longer than" and a binary operation o, which combines two wooden boards to a new one. In (Zuse, 1991) and (Zuse, Bollmann-Sdorra, 1992) it is shown that an additive property of a measure, like the ruler, leads us to the ratio scale. The measurement of the length of wooden boards takes place on the level of a ratio scale. We know this from our daily life because we can transform the numbers from KM to miles and back without changing the meaning of length.

We now consider measurement in the area of software measures and complexity measurement.

6 Software Complexity

We now want discuss the term software complexity. Although in literature ideas of software complexity can be found like cohesion, coupling, we do not give a definition of software complexity.

We use another way to talk about software complexity. We discuss the idea of complexity which is hidden behind software complexity measures. We explain the term complexity with empirical conditions that we derive from measurement theory in form of axioms. These conditions can be seen as "Gedankenexperiments".

The derivation of the empirical conditions from software measures is shown in detail in Roberts (Roberts, 1979) and related to software complexity in Zuse et al. (Zuse, Bollmann-Sdorra, 1992), (Zuse, 1992a), (Zuse, 1992b).

It should be mentioned here that similar conditions can be also found by Weyuker (Weyuker, 1988)

and Fenton (Fenton, 1991). However, these authors do not use measurement theory, the give mathematical conditions.

Firstly, we want to introduce some notations. As an empirical relation we introduce $\bullet\geq$ which means "equal or more complex". That means, for example, having two programs P1, P2∈P, where P is the set of all programs,

$$P1 \bullet> P2$$

is interpreted that P1 is more complex than P2 and

$$P1 \approx P2$$

means that P1 and P2 are equally complex.

Many of the empirical conditions are based on concatenation operations of programs, flowgraphs or structure charts. In order to show these conditions we have to define a concatenation operation. Weyuker, for example, (Weyuker, 1988) p.1359, gives a definition of a concatenation operation of programs:
A program can be uniquely decomposed into a set of disjoint blocks of ordered statements having the property whenever the first statement in the block is executed, the other statements are executed in the given order. Furthermore, the first statement of the block is the only statement which can be executed directly after execution of a statement in another block. Intuitively, a block is a chunk of code which can be always executed as a unit.

In Bollmann et al. (Bollmann, Zuse, 1985), (Zuse, Bollmann, 1989) and Zuse (Zuse, 1991) (Zuse, 1992a) the sequential concatenation operation for flowgraphs and structure charts was introduced and denoted with BSEQ or DSEQ.

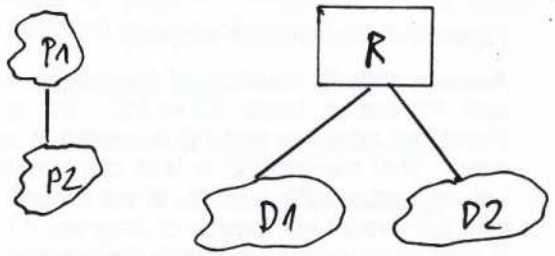


Figure 6.1: Sequential concatenation operation BSEQ=P1 o P2 of two flowgraphs P1 and P2, and the sequential concatenation operation DSEQ of a structure chart.

Two arbitrary programs P1 and P2 or flowgraphs are sequentially concatenated with BSEQ. The arbitrary programs or flowgraphs P1 and P2 are so-called primes because they have only one entry and one exit. Weyuker calls this chunks or blocks. D1 and D2 are sub-designs which are concatenated to a new structure chart by the root-node R. We denote DSEQ=D1 o D2 as the sequential concatenation operation of a structure chart.

The empirical conditions are derived, among others, from the extensive structure in measurement theory. The extensive structure gives an empirical interpretation of concatenation operations if holds for the measures:

$$\mu(P1 \circ P2) = \mu(P1) + \mu(P2)$$

The formula above describes the additive property of a measure. This is the case for the Measure LOC=|N|, where N is the set of nodes in a flowgraph, and the Measure of McCabe MCC-V2=|E| - |N| + 2 where E is the set of edges in the flowgraph: It holds:

$$LOC(P1 \circ P2) = LOC(P1) + LOC(P2)$$

and

$$MCC\text{-}V2(P1 \circ P2) = MCC\text{-}V2(P1) + MCC\text{-}V2(P2)$$

In the next Section we show several empirical conditions. These conditions give hypotheses about reality and give the user the possibility to talk about the term software complexity which is hidden behind the software complexity measures.

6.1 Independence Conditions

Firstly we consider the independence conditions. These conditions are related to the sequential concatenation operation BSEQ of flowgraphs. Let P be the set of all flowgraph, a,b,c,d∈P, $\bullet\geq$ an empirical relation like "equal or more complex" and o the concatenation operation BSEQ. The conditions C1-C4 are denoted as independence conditions and are described in detail in (Zuse, 1992a) (The figures only show the left part of the independence conditions).

Condition C1:

$a \approx b \Rightarrow a \circ c \approx b \circ c$, and $a \approx b \Rightarrow c \circ a \approx c \circ b$, for all $a, b, c \in P$.

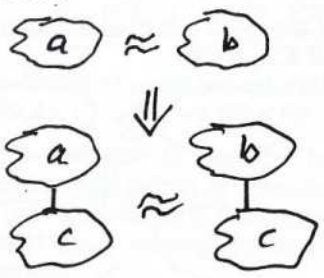


Figure 6.2: Independence condition C1.

Condition C1 is the weakest condition for independence between two components of a program.

Behind this condition is hidden a very important concept of software complexity and software measurement. Verbally formulated means independence: Is it possible to determine the complexity of a whole system from the complexities of the components of the system via a function F. More formally means that: Exists an F such that

$$\mu(P \circ P') = F(\mu(P), \mu(P'))$$

holds? The Measures LOC, MCC-V, MCC-V2 have this property, but the Measure NMCCABE=(|E|-|N|+1)/|N| does not have this property.

This formal statement is not yet discussed by authors in literature. However, it is discussed by verbally formulated statements, for example, by Fenton (Fenton, 1991), who writes: *The complexity of a sequential flowgraph should be uniquely determined by the complexities of the components.* Generally, we can say that the independence condition C1 for program complexity is not widely accepted.

Condition C2:

$a \approx b \Leftrightarrow a \circ c \approx b \circ c \Leftrightarrow c \circ a \approx c \circ b$, for all a, b, $c \in P$.

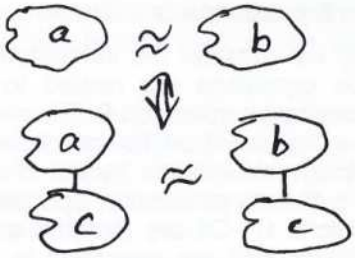


Figure 6.3: Empirical condition C2.

Condition C2 is stronger than condition C1. While condition C1 assumes an => condition C2 assumes <=>.

Condition C3:

$a \bullet\geq b \Rightarrow a \circ c \bullet\geq b \circ c$, and $a \bullet\geq b \Rightarrow c \circ a \bullet\geq c \circ b$, for all a, b, $c \in P$.

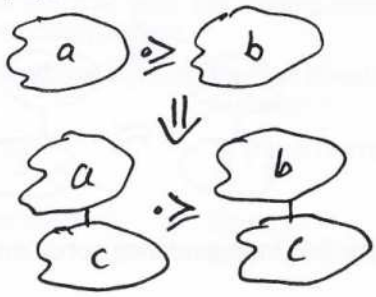


Figure 6.4: Empirical condition C3.

Condition C3 is also not generally accepted as an empirical condition for program complexity.

Condition C4:

$a \bullet\geq b \Leftrightarrow a \circ c \bullet\geq b \circ c \Leftrightarrow c \circ a \bullet\geq c \circ b$, for all a, b, $c \in P$.

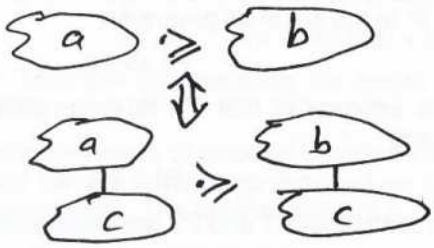


Figure 6.5: Empirical condition C4.

Condition C4 is stronger than condition C3. While condition C3 assumes an => condition C4 assumes <=>.

Conditions C1-C4 are denoted as independence conditions. That means they consider whether there are interactions between the components of a program. We illustrate the consequences of these conditions with the following example which is called the substitution property:

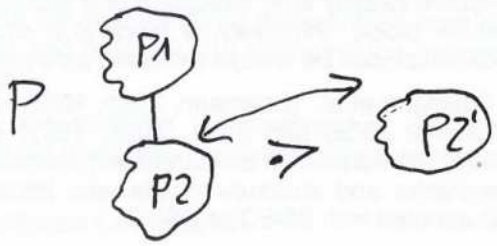


Figure 6.6: Substitution property.

Assume that P consists of the components P1 and P2 and it holds $P2 \bullet\geq P2'$. P2' is a well structured program and P2 is written in spaghetti code. That means, P2' is less complex than P2. Let us replace P2 by P2'. What should happen with the overall complexity of program P? Is now P less complex than before? If we agree then at least the empirical condition C4 has to be fulfilled.

We now give an example from Page-Jones (Page-Jones, 1988), p.307, where we show the importance of the independence conditions very clearly related to team work.

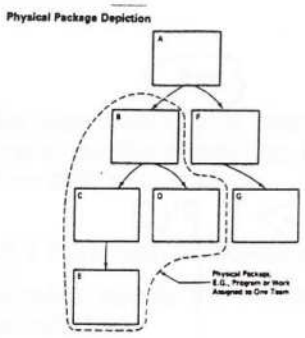


Figure 6.7: Physical Package Depiction.

Physical Package Depiction means the following: Assume the complexity or understandability of the structure chart above should be measured by the teams who have developed the physical packages A, B, C, D and E, and/or F and G. If the teams measure the complexity/understandability of the physical package consisting of the modules B, C, D and E, and F and G with a measure μ it is important to know what the complexity of the whole system should be. What shall happen with the complexity of the whole system if you decrease the complexity of the modules B, C, D and E related to a measure μ? We assume, people would agree that the complexity of the whole system should also decrease. However, only software measures which fulfil the independence condition C4 have such properties.

6.2 Conditions for Program Complexity Derived from the Extensive Structure

In measurement theory (Krantz, Luce, Suppes, Tversky, 1971), p.74, (Zuse, 1991), (Zuse, Bollmann-Sdorra, 1992), (Zuse, 1992a) we can find empirical conditions which are denoted as the extensive structure. They also can be interpreted as conditions for program complexity. These conditions (axioms) are the following:

1. A1': Weak order
2. A2': Axiom of weak associativity.
3. A3': Axiom of weak commutativity.
4. A4': Axiom of weak monotonicity.
5. A5': Archimedian axiom.

The weak order considers the ranking order of objects, in our case flowgraphs.

We now consider the empirical conditions A2'-A5' in detail. This conditions are based on concatenation operations and are hidden behind software measure which are additive related to the concatenation operation BSEQ. Let P1,..,P4 $\in$ **P** are flowgraphs and P1 o P2 the sequential concatenation operation BSEQ. The first condition is the axiom of associativity.

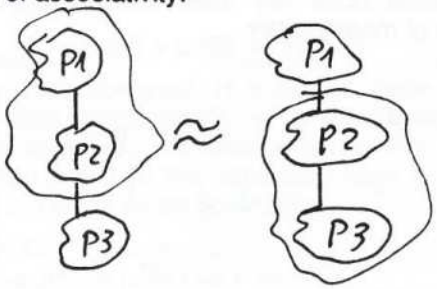


Figure 6.8: Axiom of weak associativity:
A2': P1 o (P2 o P3)≈(P1 o P2) o P3.

The axiom weak associativity is not discussed in literature as a condition for program complexity. This empirical condition is not questionable because both flowgraphs are identical, and identical flowgraphs should have the same complexity.

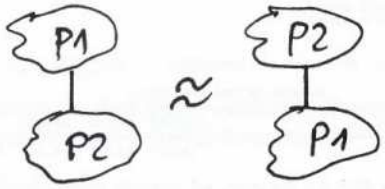


Figure 6.9: Axiom of weak commutativity:
A3': P1 o P2 ≈P2 o P1.

This empirical condition is questionable because many author require this condition and other authors reject this condition for program complexity (Zuse, 1991), p.534.

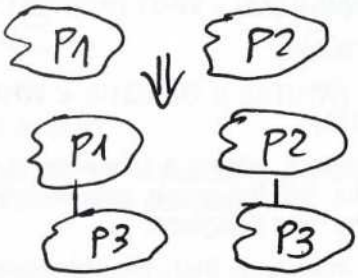
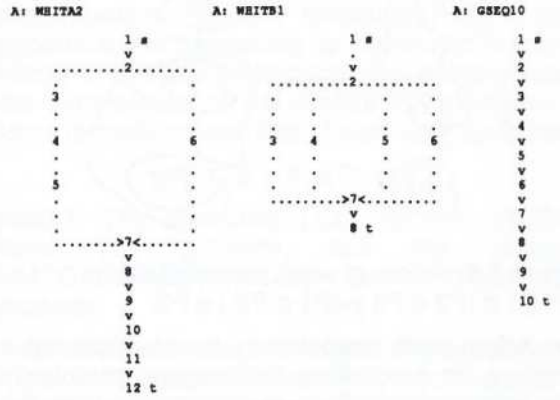


Figure 6.10: Axiom of weak monotonicity:
A4': P1 •≥ P2 => P1 o P3 •≥ P2 o P3.

This empirical condition is identical to Condition C3. In literature this axiom is discussed controversly. We will illustrate the consequences if a measure do not have the property of the axiom of weak monotonicity.

In order to show the fatal consequences if the axiom of monotonicity does not hold in reality, we

give the following example (Zuse, 1991). We show that the Measure WHIT of Whitworth and Szulewski does not have the property of the axiom of monotonicity.



Axiom of monotonicity:

Binary operation BSEQ:

M1 = WHITA2 o GSEQ10
M2 = WHITB1 o GSEQ10

MEASURES OF WHITWORTH AND SZULEWSKI:

RESULTS OF THE MEASURE(S)

	WHITA2	WHITB1	GSEQ10	M1	M2
WHIT	16.68	13.29	10.00	32.53	36.51

Figure 6.11: Axiom of weak monotonicity for the binary operation BSEQ.

The example shows, that the Measure WHIT does not have the property of the axiom of monotonicity. A sequence GSEQ10 of 10 nodes is added by the binary operation BSEQ to the flowgraphs WHITA2 and WHITB1. The relation of the complexities is:

WHIT (WHITA2) > WHIT (WHITB1).

But, it holds:

WHIT (WHITA2 o GSEQ10) < WHIT (WHITB1 o GSEQ10).

That means, adding a sequence of nodes causes that the left flowgraph gets a lower complexity than the right flowgraph.

It is intersting that the Measures HALST-V, HALST-D and HALST-E of Halstead do not have the property of the axiom of monotonicity. This has consequences related to statistical operations.

In Section 7 we will see that such a behaviour of a software complexity measure has consequences related to the validation of software measures.

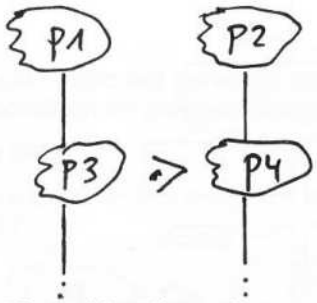


Figure 6.12: Archimedian axiom.:
A5': If P1 •> P2 then for any P3,P4 there exists a natural number n, such that nP1 o P3 •> nP2 o P4.

We explain the Archimedian axiom with two checking accounts which we denote with CH1 and CH2. Assume, account CH1 has 10 Dollars, and account CH2 has 100 Dollars. If we deposit n-times 2 Dollars to CH1 and deposit 1 Dollar to CH2, then there will be after n-times more money in account CH1 than in CH2.

The Archimedian axiom related to software measurement is not discussed in literature.

However, we can find many measures which do not have the property of the Archimedian axiom (Zuse, 1991).

6.3 Weyuker's View of Program Complexity

In this Section it is shown that there are more ideas of program complexity in literature. In (Zuse, 1992a) a complete overview of requirements of program complexity in literature is given.

We only discuss here the Weyuker properties because they are the most famous one. Weyuker (Weyuker, 1988) discusses desireable properties for software complexity measures. She discusses the properties of software measures in a mathematical way. This is an advantage because it is not possible to give them different interpretations.

Following our approach (Zuse, Bollmann-Sdorra, 1992), (Zuse, 1992a) we translate the Weyuker properties back into empirical properties by an implication =>, where o is again the sequential concatenation operation BSEQ. We only discuss the four most important requirements of Weyuker. The first property is called weak positivity and P, Q and R are programs.

6.3.1 Weak Positivity

Weyuker requires weak positivity. It is defined as

$$P \leq\bullet\ P \circ Q,$$

and

$$Q \leq\bullet\ P \circ Q.$$

The idea behind this condition is that Weyuker means adding something to a program makes it more complex.

6.3.2 Rejection of Condition C1

Weyuker rejects the independence condition C1 and requires

$$P \approx Q \Rightarrow \neg(R \circ P \approx R \circ Q).$$

Doing this Weyuker also rejects the conditions C2-C4 and the extensive structure as conditions for program complexity. As we will see in Section 7 has this consequences related to the validation of software measures.

6.3.3 Rejection of Weak Commutativity

Weyuker also rejects the axiom of weak commutativity which is also an axiom of the extensive structure. The axiom of weak commutativity is defined as

$$P \circ Q \approx Q \circ P,$$

for all P, Q$\in$ **P**.

Weyuker requires

$$P \circ Q \not\approx Q \circ P,$$

Rejecting the axiom of weak commutativity means also rejecting the extensive structure.

6.3.4 Requiring Wholeness

Weyuker requires wholeness which is defined as:

$$\mu(P1 \circ P2) > \mu(P1) + \mu(P2),$$

where μ is a software complexity measure.

Wholeness seems to intuitive for the user because it supports the idea of modularization. However, in (Zuse, 1991), Chapter 6, it is shown that it is not possible to combine wholeness, C1 and weak commutativity. The reason is that wholeness requires a ratio scale. In Bollmann et al. (Bollmann-Sdorra, Zuse, 1993) and Zuse (Zuse, 1994) it is shown that wholeness is a pseudo-requirement and it can be modified back to additive measures.

6.4 Required Conditions by Bache

Bache (Fenton, 1991), p.218, suggests axioms of program complexity. Bache formulates his axioms numerical, but using measurement theory it is possible to translate the numerical conditions back to empirical conditions by the implication =>. Again, o is the sequential concatenation operation BSEQ. Bache denotes F, H, G as arbitrary flowgraphs and P1 as an trivial flowgraph consisting of one edge and two nodes. For his considerations Bache assume that holds

$$\mu(H) > \mu(G) \Rightarrow H \bullet\!> G.$$

That means flowgraph H is always more complicated than flowgraph G. μ is a software complexity measure. We always show the original condition and then the translation back to an empirical condition by an implication =>.

Axiom 1:
$F \neq P1 \Rightarrow \mu(F) > \mu(P1) \Rightarrow F \bullet\!> P1$

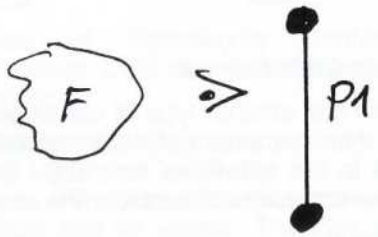


Figure 6.13: Axiom 1.

Axiom 1 means, that every arbitrary flowgraph F is more complex than a trivial flowgraph P1.

Axiom 2:
$\mu(F \circ G) > \max(\mu(F),\mu(G)) \Rightarrow F \circ G \bullet\!> \max(F,G)$

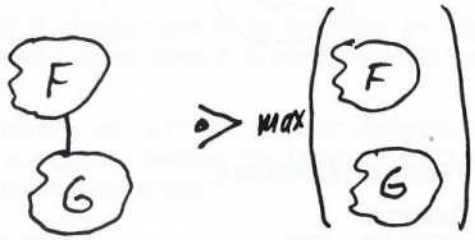


Figure 6.14: Axiom2.

Axiom 2 says that a sequence is more complex than the maximal single component.

Axiom 3:
$\mu(F \circ G) = \mu(G \circ F) \Rightarrow F \circ G \approx G \circ F.$

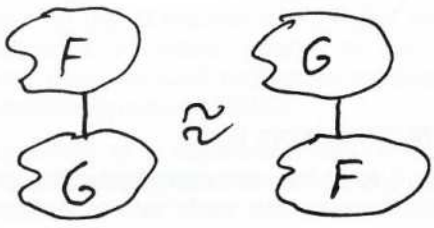


Figure 6.15: Axiom 3.

Axiom 3 is the weak axiom of commutativity. It is required by Bache.

Axiom 4:
$\mu(F \circ H) > \mu(F \circ G) \Rightarrow F \circ H \bullet\!> F \circ G$.

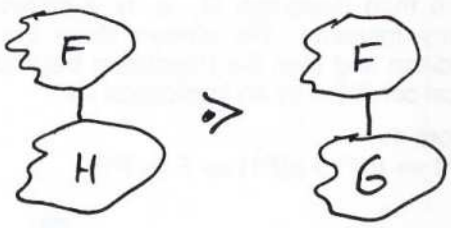


Figure 6.16: Axiom 4.

Axiom 4 is another type of the axiom of monotonicity than the axiom of weak monotonicity as defined in the extensive structure. Bache requires here > but does not consider the case of =.

Axiom 5:
$\mu(F(F1 \text{ on } a)) = \mu(F(F1 \text{ on } b)) \Rightarrow F(F1 \text{ on } a) \approx F(F1 \text{ on } b)$, where F(F1 on b) means F1 is nested in F on node b.

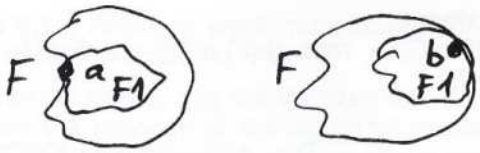


Figure 6.17: Axiom 5.

Axiom 6:
$\mu(F(H,F2,..,Fn)) > \mu(F(G,F2,..,Fn)) \Rightarrow (F(H,F2,..,Fn)) \bullet\!> (F(G,F2,..,Fn))$,
where F(G.....) means G is nested in F.

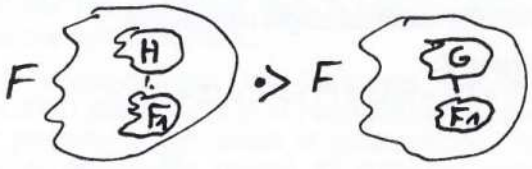


Figure 6.18: Axiom 6.

Axiom 6 says that the complexity of a program is independent of the node where a flowgraph is nested in another one.

Axiom 7:
$\mu(H(F)) > \mu(G(F)) \Rightarrow (H(F)) \bullet > (G(F))$,
where H(F) means that F is nested in H on an arbitrary node.

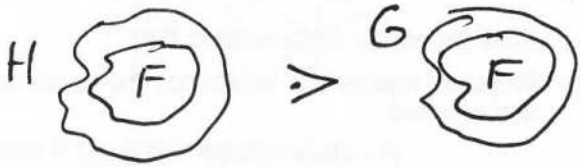


Figure 6.19: Axiom 7.

With axioms 6 and 7 Bache discusses nesting properties.

Axiom 8:
$\mu(F(G)) > \mu(F \circ G)$, that immediately implies $\mu(F(G)) > \max(\mu(F),\mu(G)) \Rightarrow (F(G)) \bullet\!> \max(F,G)$

Figure 6.20: Axiom 8.

Axiom 8 says that a nested flowgraph is more complex than the maximum of the single parts.

Axiom 9:
$\mu(H(G)) > \mu(G(H)) \Rightarrow H(G) \bullet\!> G(H)$.

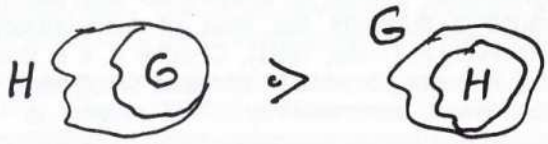


Figure 6.21: Axiom 9.

Axiom 9 is also a condition for nested structures.

The Axioms 1-9 show that program complexity can be described by empirical axioms. Bache (Fenton, 1991) presents the VINAP measures which fulfil the axioms above. It is important to notice that the VINAP measures also assume the extensive structure. This shows again that in the area of software metrics many empirical conditions about program complexity can be found.

6.5 Program Complexity behind the Measure of McCabe and LOC

We showed that discussing software measures measures implicitly empirical conditions for software complexity are assumed. Using measurement theory and assuming a homomorphism we can interpret the idea of program complexity behind a software complexity measure. We illustrate this with the Measure of McCabe and the Measure LOC.

6.5.1 Idea of Complexity behind the Measure of McCabe related to the Ranking Order

The idea of program complexity behind the Measure MCC-V2=|E|-|N|+1 of McCabe related to the ranking order is the following (Zuse, 1991), (Zuse, Bollmann-Sdorra, 1992). Let assume that P and P' are arbitrary flowgraphs.

e1: If P results from P' by inserting an edge, then P is more complex than P'.

e2: If P results from P' by inserting an edge and a node, then P and P' are equally complex.

e3: If P results from P' by transfering an edge from one location to another location, then P and P' are equally complex.

The conditions e1, e2, and e3 describe the ranking order of the Measure of McCabe related to the term complexity by operations on the flowgraph. This shows that the Measure of McCabe only measures a very simple aspect of program complexity.

6.5.2 Idea of Complexity behind the Measure of McCabe related to Additivity

We now show the idea of program complexity behind the Measure MCC-V2=|E|-|N|+1 of McCabe related to the concatenation operation BSEQ. It is easy to see that for the Measure MCC-V2 holds:

MCC-V2(P1 o P2) = MCC-V2(P1) + MCC-V2 (P2).

If a measure is additive related to a concatenation operation then the measure assumes the empirical conditions (axioms) of the extensive structure (Zuse, 1991), p.50:

1. A2': Axiom of weak associativity.
2. A3': Axiom of weak commutativity.
3. A4': Axiom of weak monotonicity.
4. A5': Archimedian axiom.

We now can easily see that the Measure of McCabe analyzes a very small aspect of software complexity. This aspect of complexity has been described by the conditions e1-e3 and the empirical conditions (axioms) of weak associativity, axiom of weak commutativity, axiom of weak monotonicity, and the Archimedian axiom.

We now show the idea of complexity behind the Measure LOC.

6.5.3 Idea of Complexity behind the Measure LOC related to the Ranking Order

We now consider the idea of complexity behind the Measure LOC. Let be **P** the set of all flowgraphs. P_0 is the flowgraph which contains exactly one node and no edges. The idea of complexity behind the Measure LOC is:

e0': $P \approx_0 P'$ iff $P=P'=P_0$.

e1': If P results from P' by inserting an edge, then P and P' are equally complex.

e2': If P results from P' by transfering an edge and a node, then P and P' are equally complex.

e3': If P results from P' by inserting an edge and a node, then P is more complex than P'.

The conditions e0', e1', e2', and e3' describe the idea of complexity behind the Measure LOC related to the ranking order.

6.5.4 Idea of Complexity behind the Measure LOC related to Additivity

For the Measure LOC holds:

LOC (P1 o P2) = LOC(P1) + LOC(P2).

The idea of complexity behind the Measure LOC related to additivity is identical to the Measure of McCabe.

We see that the Measures of McCabe and LOC have different properties related to the ranking order but they identical properties related to the concatenation operation BSEQ.

The properties of a software measure related to concatenation operations are also very important in the context of validation of software measures.

7 Validation of Software Measures

The acceptance of software measures depends on whether a software measure can be validated and that the measure can be used as a predictor. The terms validation and prediction have to be considered together. Bieman et al. (Bieman, Fenton, Gustafson, 1992), p.49, define validation of a software measure as follows:
A software measure is only valid if it can be shown to be an accurate predictor of some software attribute.

Considering the definition of validation of Bieman, validation of a software measure related to an external variable V assumes that there exists a relationship (function) between the measure and the external variable V. This kind of validation is called external validation of a measure. An internal validation of a measures means, for example, that the idea of complexity behind the measure corresponds with the experience.

In order to validate a measure the property of the external variable has to be investigated and from this the property of the software complexity measure is derived.

7.1 Properties of an External Variable

We now discuss the properties of an external variable. We consider the external variable *costs (C)*. The question is whether the external variable costs can be used as a ratio scale. For this reason we have to consider the concatenation operation BSEQ=P1 o P2 between two programs with P1, P2∈**P**, where **P** is the set of programs. The simple question is whether for the external variable costs holds:

$$C(P1 \circ P2) = C(P1) + C(P2),$$

where C are costs and o is the sequential concatenation operation BSEQ.

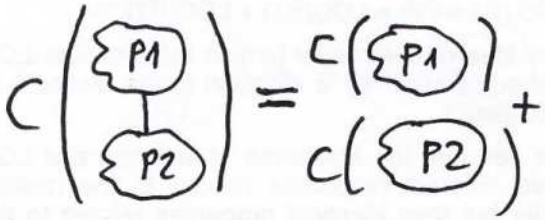


Figure 8.1: The question is whether the costs for maintenance of the two programs P1 and P2 which are combined to P1 o P2 is the sum of the costs of maintenance for P1 and P2.

This question includes the following property which is derived from the axiom of monotonicity.

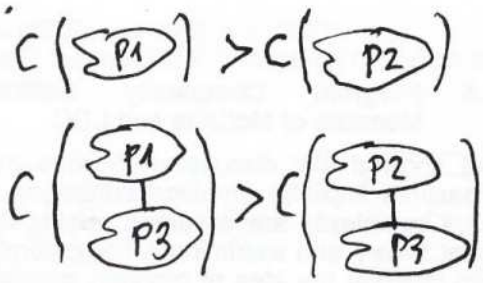


Figure 7.2: Costs of maintenance.

The question is whether for the costs of maintenance holds:

$$C(P1) \geq C(P2) \Rightarrow C(P1 \circ P3) \geq C(P2 \circ P3).$$

If users agree to the statement above, they consider the external variable as a ratio scale. The consequences are that only software measures, which are additive, like LOC and McCabe, are proper software measures for prediction of an external variable V.

The only possible function between the software measure and the external variable V, which has to be validated, is the following:

$$V(P) = a\, M(P)^b,$$

where P is a program, V(P) is the external variable, a is a constant, M is a software complexity measure which is additive.

This is a surprising result because this is exactly the COCOMO-model. That means, Boehm (Boehm, 1981) assumes that the external variable can be used as a ratio scale. The Measure LOC can be used as a ratio scale, too. More about the foundations of validation and prediction can be found in (Bollmann-Sdorra, Zuse, 1993) and (Zuse, 1994).

In this case the Weyuker properties are not appropriated properties for software measures.

8 Conclusion

It is no question that software measurement is an important method in order to get higher quality of software. Dieter Rombach said at the Eurometrics 1991 in Paris: *we should no longer ask if we should measure, the question today is how.* We agree to this statement.

Although in the past much research has been done in the area of software measurement, there are many open questions. Among others, today there is still no standardization of software measures. The proposed software measures in (IEEE Guide, 1989) are not widely accepted. Validation of software measures in order to predict an ex-

ternal variable is still a research topic for the future.

9 References

Albrecht, A.J.(1979):
Measuring Applications Development Productivity. Proc. of IBM Applic. Dev. Joint SHARE/GUIDE Symposium, Monterey, CA, 1979, pp.83-92.

Albrecht A.J.; Gaffney, S.H.(1983):
Software Function, Source Lines of Code and Development Effort Prediction: A Software Science Validation. IEEE Transactions of Software. Engineering Vol. 9, No. 6, 1983, pp. 639-648.

AMI(1992):
AMI Handbook - Applications of Metrics in Industry. Centre for Systems and Software Engineering, South Bank Polytechnic, Borough Road, London, SE1 OAA, UK, 1992.

Baker, A.L.; Bieman, J.M.; Gustafson, D.A.; Melton, A.; Whitty, R.A.(1987):
Modeling and Measuring the Software Development Process, Proc. of the Twentieth Annual International Conference on System Sciences, 1987, pp. 23-29.

Baker, A.L.; Bieman, J.M.; Fenton, N.; Gustafson, D.A.; Melton, A.; Whitty, R.A.(1990):
A Philosophy for Software Measurement. The Journal of Systems and Software, Vol. 12, No 3, 1990, pp.277-281.

Basili, V.; Selby, R.W.; Phillips, T.Y.(1983):
Metric Analysis and Data Validation across Fortran Projects. IEEE Transactions on Software Engineering, Vol. 9, No. 11, pp. 652-663, November 1983.

Basili, V.R.; Weiss, D.M.(1984):
A Methodology for Collecting valid Software Engineering Data. IEEE Transactions on Software Engineering, Vol. SE-10, No 3, Nov. 1984, pp.728-738.

Belady, L.A.(1979):
On Software Complexity. In: Workshop on Quantitative Software Models for Reliability, pp.90-94, 1979.

Bieman, James M.(1991):
Deriving Measures of Software Reuse in Object Oriented Systems. Technical Report #CS-91-112, July 1991. Colorado State University, Fort Collins, Colorado 80523, USA.

Bieman, James; Fenton, Norman; Gustafson, David(1992):
Moving From Philosophy to Practice in Software Measurement. In: Proceedings of the International BCS-FACS Workshop (Formal Aspects of Computer Software), May 3, 1991, South Bank Polytechnic, London, UK", by T.Denvir, R.Herman and R.Whitty (Eds.), ISBN 3-540-19788-5. Springer Publisher, 1992.

Boehm, B.W(1981):
Software Engineering Economics. Prentice Hall, 1981

Bollmann, P.; Cherniavsky, V.S.(1981):
Measurement-Theoretical Investigation of the MZ-Metric. In: R.N. Oddy; S.E. Robertson; C.J. van Rijsbergen; P.W. Williams (ed.), Information Retrieval Research, Butterworth, 1981.

Bollmann, Peter(1984):
Two Axioms for Evaluation Measures in Information Retrieval. Research and Development in Information Retrieval, ACM, British Computer Society Workshop Series, pp. 233-246, 1984.

Bollmann, Peter; Zuse, Horst(1985):
An Axiomatic Approach to Software Complexity Measures. Proceedings of the Third Symposium on Empirical Foundations of Information and Software Science III, Roskilde, Denmark, October 21-24, 1985. Reprinted in: Empirical Foundations of Information and Software Science III, Edited by Jens Rasmussen and Pranas Zunde, Plenum Press, New York and London, 1987, pp.13-20.

Bollmann-Sdorra, P.; Zuse, H.(1993):
Prediction Models and Software Complexity Measures from a Measurement Theoretic View. Accepted by the 3rd International Software Quality Conference. Lake Tahoe, Nevada, October 4-7, 1993.

Bowles, Adrian John(1983):
Effects of Design Complexity on Software Maintenance. Dissertation, Northwestern University, Evanston, Illinois, USA, June 1983.

Card, David, N.; Glass, Robert L.(1990):
Measuring Software Design Quality. Prentice Hall, Englewood Cliffs, New Jersey, 1990.

Chapin, N.(1979):
A Measure of Software Complexity. AFIPS National Computer Conference Spring 1979, pp.995-1002.

Chidamber, Shyam, R.; Kemerer, Chris, F.(1993):
A Metrics Suite for Object Oriented Design. M.I.T. Sloan School of Management, E53-315, 30 Wadsworth Street, Cambridge, MA 02139, CISR Working paper No. 249, February 1993, 31 pages.

Conte, S.D.; Dunsmore, H.E.; Shen, V.Y.(1986):
Software Engineering Metrics and Model. Benjamin/Cummings Publishing Company, Menlo Park, 1986.

Coupal, Daniel; Robillard, Pierre(1990):

Premilary Results of Factor Analysis of Source code Metrics. 2nd Annual Workshop Oregon Workshop on Software Metrics, Portland UNiversity, Oregon, March 19-20, 1990.

DeMarco, Tom(1982):
Controlling Software Projects Management, Measurement and Estimation. Englewood Cliffs, N.J.: Prentice Hall, 1982

DeMillo, Richard A.; Lipton, Richard J.(1981):
Software Project Forecasting. In: Software Metrics - An Analysis and Evaluation. The MIT Press, 1981, p.77-94.

Dumke, Reiner(1992):
Softwareentwicklung nach Maß - Schätzen - Messen - Bewerten. Vieweg Verlag, 1992.

Ejiogu, L.(1991):
Software Engineering with Formal Metrics. QED Technical Publishing Group, 1991.

Elliott, J.J (Editor); Fenton, N.E.; Linkman, S.; Markham(1988):
Structure-Based Software Measurement. Alvey Project SE/069, 1988, Department of Electrical Engineering, South Bank, Polytechnic, 103 Borough Road, London, SE1 OAA, UK.

Van Emden, M.H.(1971):
An Analysis of Complexity. Mathematical Centre Tracts, 1971.

Emerson, Thomas J.(1984):
Program Testing, Path Coverage, and the Cohesion Metric. IEEE COMPSAC, 1984, pp. 421-431

Fenton, Norman(1991):
Software Metrics: A Rigorous Approach. City University, London, Chapman & Hall, 1991.

Fenton, Norman(1991a):
The Mathematics of Complexity in Computing and Software Engineering. In: The Mathematical Revolution Inspired by Computing. J.H. Johnson & M.J. Looms (eds), 1991, The Institute of Mathematics and its Applications, Oxford University Press.

Fenton, Norman; Littlewood, B(1990):
Software Reliability and Metrics. Elsevier Applied Science, 1990.

Gilb, T.(1977):
Software Metrics. Winthrop Publishers, Cambridge, Massachusetts, 1977.

Goodman, Paul(1992):
Practical Implementation of Software Metrics. McGraw Hill Company, 1992.

Grady, Robert B.; Caswell, Deborah L(1987):
Software Metrics: Establishing a Company-Wide Program Prentice Hall 1987

Grady, Robert B.(1992):
Practical Software Metrics for Project Management and Process Improvement. Prentice Hall 1992.

Halstead, M.H.(1977):
Elements of Software Science. New York, Elsevier North-Holland, 1977.

Harrison, Warren; Magel, Kenneth(1981):
A Complexity Measure Based on Nesting Level. ACM SIGPLAN Notices, Vol. 16, No. 3, pp. 63-74, 1981.

Hecht, M.(1977):
Flow Analysis of Computer Programs. Elsevier, New York, 1977.

Henry, S.; Kafura, D.(1981):
Software Metrics Based on Information Flow,. IEEE Transactions on Software Engineering Vol. 7, No. 5, 1981.

Henry, Sallie.; Wake, Steve(1988):
Predicting Maintainability with Software Quality Metrics. TR88-46, 1988, Department of Computer Science, Virginia Polytechnic, Blacksburg, Virginia, USA.

Hutchens, D.H.; Basili, V.R.(1985):
System Structure Analysis: Clustering with Data Bindings. IEEE Transactions on Software Engineering 11(8), August 1985.

Hetzel, Bill(1993):
Making Software Measurement Work - Building an Effective Measurement Program. QED, 1993.

IEEE Guide(1989):
IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software. IEEE Computer Society. 345 East 47th Street, New York, NY 10017, USA.

IEEE(1989a):
IEEE Standard Dictionary of Measures to Produce Reliable Software, IEEE Computer Society. 345 East 47th Street, New York, NY 10017, USA.

Jones, C.(1991):
Applied Software Measurement: Assuring Productivity and Quality. McGraw Hill, New York, NY, 1991.

Kafura, Dennis; Canning, James(1988):
Using Group and Subsystem Level Analysis to Validate Software Metrics on Commercial Software Systems. TR 88-13, 1988, Polytechnic, Blacksburg, Virginia, USA.

Kearney, Joseph K.; Sedlmeyer, Robert L.; Thompson, William(1986):

Software Complexity Measurement. Communications of the ACM Vol. 29,No. 11, 1986.

Khoshgoftaar, T.M.; Munson, J.C.(1992):
An Aggregate Measure of Program Module Complexity. Annual Oregon Workshop on Software Metrics, March 22-24, 1992, Silver Falls, Oregon, USA.

Kitchenham, B.; Littlewood, B.(1989):
Measurement for Software Control and Assurance. Elsevier, 1989.

Krantz, David H.; Luce, R. Duncan; Suppes, Patrick; Tversky, Amos(1971):
Foundations of Measurement - Additive and Polynomial Representation. Academic Press, Vol. 1, 1971.

Kriz, Jürgen(1988):
Facts and Artefacts in Social Science: An Ephistemological and Methodological Analysis of E Research Techniques. McGraw Hill Research, 1988.

Lake, Al; Cook, Curtis(1992):
A Software Complexity Metric for C++. Proceedings of the Annual Oregon Workshop on Software Metrics, March 22-24, 1992, Silver Falls, Oregon, USA. Lakshmanan, K.B.; Jayaprakash, S.; Sinha, P.K.(1991):
Properties of Control-Flow Complexity Measures. IEEE Transactions on Software Engineering, Vol. 17, No.12, December, 1991, p.1289-1295.

Levitin, Anany V(1986):
How To Measure Software Size, And How To Do Not pp. 314-818, COMPSAC 86

Ligier, Yves(1989):
A Software Complexity Metric System Based on Intra- and Inter-modular Dependencies. IBM RC 14831 (#65457) November 5, 1989.

Longworth, H.D.; Ottenstein, L.M.; Smith, M.R.(1986):
The Relationship between Program Complexity and Slice Complexity During Debugging Tasks. IEEE COMPSAC, October 1986, pp.383-389.

Luce, R. Duncan; Krantz, David H.; Suppes, Patrick; Tversky, Amos(1990):
Foundations of Measurement - Representation, Axiomatization, and Invariance. Vol 3, Academic Press, 1990.

MacDonell, Stephan(1991):
Reliance on Correlation Data for Complexity Metric Use and Validation. ACM SIGPLAN Notices, Vol. 26, No. 8, August 1991.

Mayrhauser, Anneliese von(1990):
Software Engineering Methods and Management. Academic Press, Inc., 1990

McCabe, T.(1976):
A Complexity Measure. IEEE Transactions of Software Engineering. Vol. SE-1, No. 3, pp. 312-327, 1976.

McCabe, T; Butler, Charles W.(1989):
Design Complexity Measurement and Testing. Communications of the ACM, Vol. 32, No. 12, Dec 89, pp. 1415-1424.

METKIT(1993):
METKIT - Metrics Educational Toolkit. Information and Software Technology, Vol 35, No. 2, February 1993.

Mills, Everald, E.(1988):
Software Metrics. SEI Curriculum Module SEI-CM-12-1.1, December 1988, Software Engineering Institute, Pittsburg, PA, USA.

Möller, K.H.; Paulish, D.J.(1993):
Software Metrics. Chapman & Hall, 1993.

Morris, Kenneth, L.(1989):
Metrics for Object-Oriented Software Development Environments. Massachusetts Institute of Technlogy, Master of Science in Management, May 1989.

Munson, J.C.; Khoshgoftaar, T.M.(1989):
The Dimensionality of Program Complexity. Proceedings of the 11th Annual International Conference on Software Engineering, Pittsburg, pp. 245-254.

Myers, G.J.(1976):
Software Reliability - Principles and Practices. Wiley & Sons, 1976.

NASA, National Aeronautics and Space Administration(1981):
Software Engineering Laboratory (SEL), Data Base Organization and User's Guide, Software. Engineering Laboratory Series SEL-81-002, Sept, 1981.

NASA, National Aeronautics and Space Administration(1984):
Software Engineering Laboratory (SEL), Measures and Metrics for Software Development. Engineering Laboratory Series SEL-83-002, March 1984.

NASA, National Aeronautics and Space Administration(1986):
Software Engineering Laboratory (SEL), Measuring Software Design. Engineering Laboratory Series SEL-86-005, November 1986.

Ott, Linda M.; Thuss, Jeffrey J.(1991):
Sliced Based Metrics for Estimation Cohesion. Technical Report #CS-91-124, November 1991, Colorado State University, Fort Collins, Colorado 80523, USA.

Oviedo, Enrique I.(1980):
Control Flow, Data Flow and Programmers Complexity. Proc. of COMPSAC 80, Chicago IL, pp.146-152, 1980.

Page-Jones, Meilir(1988):
The Practical Guide to Structured Systems Second Edition, Yourdon Press, 1988

Park, Robert, e.(1992):
Software Size Measurement: A Framework for Counting Source Statements (Draft). Software Engineering Institute, Pittsburg, May 1992.

Perlis, Alan; Sayward, Frederick; Shaw, Mary(1981):
Software Metrics - An Analysis and Evaluation, The MIT Press, 1981.

Piwowarski, Paul(1982):
A Nesting Complexity Measure. SIGPLAN Notices, pp. 44-50, Vol. 17, No. 9, 1982.

Pressmann, Roger S.(1992):
Software Engineering: A Practitioner's Approach, Third Edition, McGraw Hill, 1992.

RADC, Rome Air Development Center(1984):
Automated Software Design Metrics. RADC-TR-S4-27, 1984, Air Force System Command, Griffies Air Force Base, NY 13441.

Roberts, Fred S.(1979):
Measurement Theory with Applications to Decisionmaking, Utility, and the Social Sciences. Encyclopedia of Mathematics and its Applications Addison Wesley Publishing Company, 1979.

Rocacher, D.(1988):
Metrics Definition for Smalltalk. ESPRIT Project 1257, Januar 1988.

Rombach, D.(1990):
Design Measurement - Some Lessons Learned. IEEE Software, March 1990, pp.17-24.

Rubey, R.J.; Hartwick, R.D.(1968):
Quantitative Measurement Program Quality. ACM, National Computer Conference pp. 671-677, 1968.

Samadzadeh-Hadidi, Mansur(1987):
Measurable Characteristics of the Software Development Process Based on a Model of Software Dissertation, University of Southwestern Louisiana, USA, May 1987

Selby, Richard, W.(1992):
Interconnectivity Analysis Techniques for Error Localization in Large Systems. Annual Oregon Workshop on Software Metrics (AOWSM), Portland State University, March 22-24, 1992.

Shepperd, Martin (Editor)(1993):
Software Engineering Metrics - Volume I: Measures and Validations. McGraw Hill Book Company, International Series in Software Engineering, 1993.

Shepperd, Martin; Ince, Darrel(1993a):
Derivation and Validation of Software Metrics. Clarendon Press - Oxford., 1993.

Shooman, Martin L.(1983):
Software Engineering. McGraw Hill, 1983.

Sommerville, Ian(1992):
Software Engineering. Fourth Edition, Addison Wesley, 1992.

Stevens, W.P.; Myers, G.J.; Constantine, L.L.(1974):
Structural Designs. IBM System Journal, 13(2), pp. 115-139, 1974.

Troy, Douglas; Zweben, Stuart(1981):
Measuring the Quality of Structured Design The Journal of System and Software. Vol. 2, 113-120, 1981, pp.113-120.

Weiser, M.D.(1982):
Programmers Use Slices When Debugging. Communications of the ACM Vol. 25, No. 7, July 1982, pp. 446-452.

Weyuker, Elaine J.(1988):
Evaluating Software Complexity Measures. IEEE Transactions of Software Engineering, Vol. 14, No. 9, Sept. 1988.

Wolverton, R.W.(1974):
The Cost of Developing Large-Scale Software. IEEE Transactions on Computer, Vol. C-23, No. 6, pp. 615-636, June 1974.

Yin, B.H.; Winchester, J.W.(1978):
The Establishment and Use of Measures to Evaluate the Quality of Software Designs. In: Proceedings of the ACM, Software Quality Assurance Workshop, pp. 45-52, IEEE Computer Society, 1978.

Zuse, Horst(1985):
Meßtheoretische Analyse von statischen Softwarekomplexitätsmaßen. Dissertation, (Ph. D. Thesis). TU-Berlin 1985, Department of Computer Science, Franklinstraße 28/29, FR 5-3, 1 Berlin 10, Germany,

Zuse, Horst; Bollmann, Peter(1987):
Using Measurement Theory to Describe the Properties and Scales of Static Software Complexity Metrics. IBM Thomas Watson Research Center, Yorktown Heights, RC 13504, 1987.

Zuse, Horst; Bollmann, Peter(1989):

Using Measurement Theory to Describe the Properties and Scales of Static Software Complexity Metrics. SIGPLAN Notices, Vol. 24, No. 8, pp. 23-33, August 1989.

Zuse, Horst(1991):
Software Complexity - Measures and Methods. DeGruyter Publisher, 1991, Berlin, New York, 605 pages, 498 figures.

Zuse, Horst; Bollmann-Sdorra, Peter(1992):
Measurement Theory and Software Measures. In: Proceedings of the International BCS-FACS Workshop (Formal Aspects of Computer Software), May 3, 1991, South Bank Polytechnic, London, UK, by T.Denvir, R.Herman and R.Whitty (Eds.), ISBN 3-540-19788-5, October 1992, Springer Publisher, Springer Verlag London Ltd, Springer House, 8 Alexandra Road, Wimbledon, London SW19 7JZ, UK.

Zuse, Horst(1992a):
Measuring Factors Contributing to Software Maintenance Complexity. Proc. 2nd International Conference on Software Quality, Triangle Research Park, NC, October 4-7, 1992, ASQC (American Society for Quality Control) 611 East Wisconsin Avenue, Milwaukee, Wisconsin 53202, USA, pp. 178-190.

Zuse, Horst(1992b):
Properties of Software Measures. Software Quality Journal, Vol 1, December 1992, pp. 225-260.

Zuse, Horst(1994):
Foundations of Validation, Prediction, and Software Measures. Accepted by the AOSMW94 (Annual Oregon Software Metric Workshop, Portland, April 20-22, 1994.

Zuse, Horst(1994a):
Software Complexity Metrics/Analysis. John Wiley Publisher: "Encyclopedia of Software Engineering" (Contains 300 articles, 1800 pages, three volumes), Will appear Spring 1994.

About the Author:

Horst Zuse received his B.S. degree in electrical engineering from the Technische Universität of Berlin, Germany, in 1970, the Diploma degree in electrical engineering from the Technische Universität in 1973, and the Ph.D. degree in computer science from the Technische Universität of Berlin in 1985. Since 1975 he is senior research scientist with the Technische Universität Berlin. His research interests are information retrieval systems, software engineering, software measures and the measurement of "complexity" and "quality" of software during the software life-cycle. From 1987 to 1988 he was for one year with IBM Thomas J. Watson Research in Yorktown Heights. His research work there was software measures, too. In 1991 he published the book: Software Complexity - Measures and Methods (De Gruyter Publisher). From 1989 till 1992 he was with the ESPRIT II Project 2384 METKIT (Metric-Educational- Toolkit) of the European Commission. Since 1990 he gives several times a year seminars about software measures for people of the industry within the scope of DECollege. He also gave many presentations and seminars about software measurement on conferences in US and Canada. Now, his research interests are validation of software measures and application of software measures in the whole software life-cycle.

Process Improvement: How Much Can the Organisation Endure?

Hans-Jürgen Kugler
International Software Consulting Network
and
K&M Technologies Limited
Ireland

Abstract

Process improvement and the supporting techniques are often viewed from a technological perspective. This paper emphasises the organisational and human aspects of process improvements. Successful process improvement has to be supported by results from disciplines other than information technology.

The question is asked as to whether technological and social process engineering techniques apply equally well to organisations of widely differing size and from different industrial sectors.

1. Introduction

Process improvement involves considerably more than the technical aspects of software development processes. According to Curtis' "Orange Peel Model" one has to look at the influence of the nested environments a software producing unit is operating in: the overall organisation and its objectives, the business sector, the market or regulatory environment, and so on.

Obviously process improvement has to deal with and reflect nested, interacting organisational structures of increasing size, in which human interrelationships generally play a more important rôle than technology.

Let us start by looking at the "quality challenge" — or 'problem', depending on your view — that software developers are faced with today, and examine whether it really is a problem to be solved by information technology.

1.1. The Problem

In recent years, the concept of quality has moved to the forefront of the challenges faced by system developers. There are many different aspects of quality that affect the design, development and delivery of systems in today's competitive world. These include:

- Satisfied Customer:
  - — receives what was asked for
  - — at the right time
  - — at the price that was quoted
- Functional Correctness:
  - — ship a system that works correctly from day one

Delivery of the system on time, and correct costing of the system demand that the developer estimate at an early stage, and with some degree of confidence, the development effort which will be required to produce the system.

Industry trends indicate that an increasing proportion of the functionality of today's systems is being realised in software. High-quality systems rely to an ever-growing extent on high-quality software.

In the current state of software practice, quality objectives as cited above are met only in rare cases. Software developments are all too often characterised by budget overrun,

schedule overrun, dissatisfaction with some features or performance, or total write-off in some extreme cases.

With a growing emphasis on quality, system developers will need to avoid these shortcomings in order to survive in the tomorrow's business environment.

It is worth noting that all these comments apply to an environment which is managed by humans, in which decisions are made by humans, and in which even the implementation of these decisions is driven more by humans than by technology.

1.2. Why *Process* is Important

Traditional approaches to quality have relied on intensive testing, aimed at detecting and eliminating defects prior to shipping. This somewhat restricted approach does not seek to prevent the re-occurrence of those same defects in future products. Detecting and eliminating the *source* of defects is necessary in order to achieve sustained high-quality in product development. This points out the need for a *repeatable* process in the first place, with constant monitoring to drive a program for continuous *improvement.*

The development process itself, therefore, is targeted as a key contributor to quality. A quality product can only be produced using a development process which itself adheres to high quality standards. Such a quality process minimises the introduction of defects into the product, and ensures that it will be built in predictable and manageable time, using predictable and manageable resources.

This perspective on quality is recognised by the ISO 9000 series of quality standards (applicable to hardware and software developments) and, in the US and Europe, by the SEI Capability Maturity Model and BOOTSTRAP (applicable to software developments). Assessment under either of these programmes focuses on the extent to which the development process is *managed.* Process monitoring with continuous improvement are also emphasised in these programmes.

A manageable, repeatable process can be monitored systematically and the sources of defects identified and eliminated. Through constant improvement, the process is refined to deliver consistently increasing levels of quality and reliability in the final product. It is important again to note that the monitoring is conducted by human teams—it is generally not inherent in a purely technological system.

1.3. Management of Change and Management for Change

Every business/industry has to adapt to change, or be faced with declining sales/profits.

Evolution in business is induced by:

- development of markets
- new products / services
- change of technology
- change of business environment, e.g. political, regulatory, legal, ...

A business/industry can be characterised by products, services, markets, etc., but the most pertinent—and generally most closely guarded—are its *processes*, the embodiment of the organisations "know-how".

Aim: To manage the development of organisational structures and its associated processes in a planned and purposeful way.

The average workload tends to increase, requiring higher productivity at the same or better quality. Productivity increase has been sought by "automating" activities by using the support of information technology. However, you need to know *what* to support, before being able to determine *how* to support it!

Without precise and unambiguous understanding of the requirements placed by the business—i.e. its processes—on the support system one may develop—or acquire—a support system, "doing things the right way", but it cannot be guaranteed that this will lead to "doing the right thing".

The support system needs to fit the process, and the process needs to fit the business.

Generally well co-ordinated teams are the secret of "good business"; here the team members act as players complementing each other to form one balanced (interacting) unit.

Process improvement obviously has a "social engineering" component. Technology can be used to support this, but it is not the primary driver for successful change. The efficiency (productivity and quality) of an organisation can be improved by studying and managing the co-ordination of elements of teamwork. This has been shown, for example, to reduce cycle time.

An approach to this is to examine the underlying process and identify those interactions or co-ordination points which involve "buffering", a potential cause for delays and complexity. Group interactions themselves also require analysis—once recorded as elements of the process. Both of these are candidates for process improvement or tuning.

2. Process Improvement and the Organisation

The process modelling part of process improvement aims at capturing the functional architecture of an organisation, i.e. who does what, when, and with whom.

Top-down analysis of the business should emphasise the link between:

- business objectives
- strategic goals
- operational goals
- processes
- tasks
- activities

Organisational charts help to identify actors and rôles, and this helps to relate the human element of the organisation and the technological support. Procedures, quality manuals, etc., provide the source of information for relationships and sequencing of activities—and help identify more actors and rôles.

An organisation is modelled as a collection of processes involving interacting rôle (performed by actors or agents), e.g. a review rôle/agent interacting with a component design rôle/agent.

The integration of process modelling and its results into the organisation as part of process improvement needs to be controlled:

Process Specification:

> The quality objectives for the process are defined. The current process followed by the organisation is analysed and mapped into the chosen process framework. Quality gaps and shortcomings are identified and, by reference to the framework, appropriate activities are added to the process to eliminate the gaps (e.g. management, review and measurement activities).

Process Deployment:

> The specified process is installed within the organisation. Suitable training must be provided to ensure effective take-up of the process throughout the organisation—training in the process itself, and in the methods, techniques and tools that are invoked as part of the process.

Process Improvement:

> The process is monitored and evaluated—based on information provided by the measurement activities. Problems areas are identified and the process is modified to overcome the problems. Note: Process Improvement is a continuous activity. The development process must allow for evolution and constant improvement to cope with changing quality demands and more demanding resource and time constraints.

3. "Social" Engineering

The above discussion shows that technological issues are actually a minor part of successful process improvement. The concerns of the organisation and how its members work to achieve the organisation's goals are the major area for improvement—yielding the greatest benefit if combined with the right technology, but also providing the potentially greatest obstacle.

Total Quality Management programmes are being implemented by ever increasing numbers of organisations. However, recent surveys quoted by [Buzan 1994] indicated that 80% of these quality programmes were considered failures by those who were involved in them. Figures on software process improvement attempts are likely to show similar tendencies—because they were technology driven, rather than by concern for the organisation and its individual members.

The sliding scale of effectiveness of process improvement deployment can be illustrated best by the degree of congruence between what is said, and what is actually done [Weinberg 1992]:

oblivious	We don't even know that we are performing a process.
variable	We do whatever we feel like at the moment.
routine	We follow our routines (except when we panic).
steering	We choose among our routines by the results they produce.
anticipating	We establish our routines by our past experience with them.
congruent	Everyone is involved in improving everything all the time.

3.1. Deming's Principles

Deming's fourteen quality principles have influenced TQM and certainly also the software process movement. To illustrate the importance of "social" process engineering the following four of Deming's principles are to be discussed further in the remainder of this paper:

- constantly improve every process in the system
- reduce and eliminate fear
- eliminate numerical quotas
- institute organisation-wide education and re-training programmes

3.2. Key Barriers

After the stages of analysing existing processes, modelling and specifying improved processes the 'real' problems are only about to begin. To effectively deploy and maintain the improved process requires support and commitment from all those involved. However, there is always a resistance to change, and the resistance grows with the amount of change (non-linear!). Systems and organisations develop by themselves, as stated so well in "Boulding's Backward Basis" [Weinberg 1986]:

"Things are the way they are because they got that way."

This emphasises the often less than logical change a system or organisation experiences over time. The manifestations of resistance are therefore often also more psychological than based on logical reasoning. Key barriers to overcome include:

- lack of management commitment; lack of continued management support
- improvement is just perceived as "flavour of the month" that will surely go away if ignored
- people perform best to what they are measured against, i.e. measurement changes behaviour
- general resistance to change
- short term management priorities—this is just another task that does not have immediate implications
- skill shortage, which may indicate lack of timely and appropriate training

- poor communication, resulting in lack of understanding

3.3. Technology Transfer Issues

> *"When the thinking changes, the organization changes, and vice versa."*
>
> [Weinberg 1992]

3.3.1. Right Understanding

A prerequisite for successful technology transfer is that the transferring and the receiving party get the right understanding

- *Intention*: it must be clear, based on company goals, why the steps proposed are being undertaken
- *Communication*: the plans and the intentions must be communicated to all involved—circulars and memos are messages, but do not imply communication
- *Perception*: it must be recognised that the perception of what is being proposed may differ depending on the rôle of the individual in the organisation, e.g. "is this measurement intended to be used to improve the process or to assess my performance"

3.3.2. Early Adopters and Opponents

The same—right—understanding will not be created throughout the organisation uniformly. There will be those who will react sooner than others. These are likely to be individuals or groups who have been suggesting changes in the past.

These potential early adopters can be used for spreading understanding much better through show piece applications. Such early adopters can become *process champions*, if they have the right level of technical or management standing in the organisation. Such process champions are needed at management and engineering level. They will drive themselves to whip up support for the changes being planned.

Besides such visionary individuals there are those who can become *process bashers*. They are likely to insist that the chosen course of action cannot work and is domed to failure. Together with 'fear of failure' (see below) these people can be singularly de-motivating. Empowerment of adopters (see also below) or even re-deploying such individuals are the most effective means of handling such situations.

Early recognition of 'procedural fiefdoms' is essential—buy-in needs to be created here or top level management intervention may be required.

3.3.3. "Buy-In"

Buy-in is necessary to widen the base of the technology transferred. The rationale for the introduction of the improvement steps needs to be understandable—ideally quantifiable based on objective data:

- What can I gain—now and later?
- What is the risk—in particular: now?

Generally changes that promise gains of 20% or more have a significant chance of being adopted, provided the risk element is taken care of. The risk element can only be controlled or eliminated by empowerment to take risk, i.e. management backing for change deployment.

3.4. The Process Champions

Process champions are required for various activities:

Process Definition Group	Following the analysis phase champions of the process improvement are needed to produce a realistic overall process model.
Process Support Group	The teams must be involved in the definition of the improvement of the processes they are taking part in—this in turn must be supported effectively and with enthusiasm.
Process Owners in projects	In the development projects process champions must support and track the execution of the defined process for the project. This is also a valuable source for feedback.

Process champions play 'tutor' or 'mentor' rôles in their respective organisations. They help the growing process improvement movements. However, small to medium sized organisations are at a distinct advantage here, because it may be difficult to spare such a valuable resource, or even contract such a resource in the first place.

3.5. Deming's Principles Re-Visited

3.5.1. Continuous Improvement and Measurement

One of Deming's principles is to constantly improve all aspects of the processes in the organisation. However, goals become concrete and reachable when they can be quantified—continuous improvement is too vague to qualify. A plan with measurable milestones is required, and this may conflict with Deming's principle of elimination of numerical quotas.

This does not mean that the principles are wrong, but that it is important in which way they are applied within the organisation, and in which way they are presented to the individuals making up this organisation.

Another aspect of the continuous improvement principle is that it seems to suggest that there will be steady improvement, no decline at any time. Improvement means learning, and learning—at individual and organisational level—means experimenting, making mistakes, and learning from these mistakes. This suggests that there will be not "smooth" improvement curve that one can ride on, but that it will look rather "jagged" with ups and downs—however, with an overall improvement trend.

3.5.2. Fear of Failure

Admission and acceptance of failure is necessary, but this can only work if individuals are empowered to take risks and fail, without fear for their career (within reason). This touches on one of the other principles of Deming: elimination of fear. The above discussion makes it clear that in a continuous improvement situation the likelihood of increasing fear is actually greater than in a static environment. This must be countered pro-actively by improved training and education about the way progress is made, and complemented by supporting management behaviour.

Metrics programmes that include "observation of the individual" may contribute to the creation of fear, unless there is a clear understanding that the process and not the individual is being measured. Trust and non-obtrusive interference is required in order to be able to collect objective data.

3.5.3. Training and Education

All of the previous points imply a significant education and training programme for the organisation. It is relatively easy to generate the necessary team spirit to implement the required changes given a good training and education programme. However, this cannot be a once-off effort, if the improvement is to be maintained.

The translation from theory to practice requires constant re-enforcement, and studies of human memory indicate that a trainee will 'forget' about 80% of the detail of what was being learned within 24 hours, and about 99% within two weeks—unless countermeasures are taken.

This means that appropriate learning and review and application cycles must be established as part of a *continued* training programme.

3.6. Position of the Organisation

The previous discussion in this section has outlined significant implications of process improvement for the organisation and vice versa. But can process improvement with all its implications be recommended for all organisations, irrespective of their size, product or service and market sector?

Can one quantify minimum required investment in terms of percentage of available resources? Do the risks increase with decreasing organisation size? Do these paradigms work in all industrial and service sectors? And can one quantify potential return on investment?

There are no real answers, yet—only a few examples and case studies.

4. Conclusion

There are more questions than answers. And for most improvement paradigms, as soon as they involve organisational changes, there is a lot of 'woolly' argumentation, but little quantitative evidence (at least in the public domain).

However, it is certain that the—positive or negative—impact of organisational issues on the success of process improvement is greater than that of any supporting technology. Management ignorance of facts—"we are doing all of this already, aren't we"—or management impatience—"we want level 4 within six months" are probably among the foremost contributors to the failure of improvement programmes.

> *"Poor management can increase software costs more rapidly than any other factor."*
>
> [Boehm 1981]

But—where are the guarantors for success?

Selected References

Framework for Success — A Guide to Quality in Software Development and Support. Dublin: National Centre for Software Engineering, 1992b.

Akima, Noboru and Fusatake Ooi. "Industrializing Software Development: A Japanese Approach." IEEE Software 1989 (March 1989): 13 ff.

Bollinger, Terry and Clement McGowan. "A Critical Look at Software Capability Evaluations." IEEE Software 1991 (July 1991): 25 ff.

Boehm, Barry W. Software Engineering Economics. Prentice-Hall, Englewood Cliffs, NJ, 1981

Buzan, Tony. "A New approach to Quality—Mental Literacy." CSM, March 1994

Cusumano, Michael A. Japan's Software Factories: A Challenge to US Management. Oxford, 1991.

Debou, Christophe. "AMI: A New Paradigm for Software Process Improvement." In ISCN'94, Practical Improvement of Software Processes and Products, Dublin, May 1994

Humphrey, Watts and Bill Curtis. "Comment on "A Critical Look"." IEEE Software 1991 (July 1991): 42 ff.

Humphrey, Watts, Terry Snyder, and Ronald Willis. "Software Process Improvement at Hughes Aircraft." IEEE Software 1991 (July 1991): 11 ff.

Humphrey, Watts S. Managing the Software Process. SEI Series in Software Engineering, Readig, MA: Addison-Wesley, 1989.

Huczynski, Andrzej and David Buchanan. Organizational Behaviour. Prentice-Hall, New York, 2nd ed., 1991.

Messnarz, Richard. "BOOTSTRAP and ISO 9000: A Quantitative Approach to Objective Quality Management." In ISCN'94, Practical Improvement of Software Processes and Products, Dublin, May 1994

Raghavan, Sridhar A. and Donald R. Chand. "Diffusing Software-Engineering Methods." IEEE Software (July 1898 1989): 81-90.

Weinberg, Gerald M. The Secrets of Consulting. Dorset House Publishing, New York, 1986

Weinberg, Gerald M. Quality Software Management—Systems Thinking. Dorset House Publishing, New York, 1992

ISCoN94 Worksop , Thursday May 5

ISCN
International Software Consulting Network

Workshop Issue	Questions to be Discussed
Business Factors	1. What measurable benefits have been gained from process improvement programmes ? If these measures are evaluated for an organisation A, are the results really usable for any other organisation B? Does the improvement highly depend on different kinds of organisations ? 2. Are the goals of a measurement programme aligned to our business goals and stated in quantifiable terms ? 3. Market trends - e.g. the Indian scene: MOTOROLA in India has achieved maturity level 5. The industrial scene has completely transformed in the last 5 years. Many other Indian companies (about 10) have very active software quality thrusts. Will the Indian market be able to come to the European and US market with products at lower prices and equal quality ? Will this lead to the same situation as for hardware where most of the HW is produced in e.g. countries such as TAIWAN ?
Cultural Factors	1. Are quality improvement programmes cultural neutral? Or are there techniques that will work only in some countries or cultures but not in others ? 2. Is "institutionalisation" or a culture of continous process improvement (in the sense of SEIs CMM, or BOOTSTRAPs quality profile model) possible? Is it desirable?
Psychological Factors	1. Does the staff understand why the measurement programme is established and what activities are expected from them ? Will they accept that or is it a very difficult process to convince the staff ?
Management Issues	1. Is there a best way to start a systematic effort at software process improvement in an organisation ? (Contrary: Are there poor ways to start that doom such an effort to failure?) 2. What is the extent of evidence to support process improvement via configuration management ? What specific business results can be achieved? Which companies and markets are taking steps to make improvements via configuration management ?
Technical Issues	1. Would it not be useful to have an international task force working on software process improvement practices for practitioners written by practitioners ? 2. How can the European results be compared to the US ones? Is there any significant difference? Which problems do we have when comparing European with US data?
Technology Transfer	1. Is the European programme ESSI for technology transfer really attractive if companies have to invest more than 2 man months in proposals (with a hit rate of only 10%) and have to wait for the money such a long time ? Does a customer of the prime contractor really wait until ESSI provides the resources ? 2. What kind of training should the staff be given ? 3. What are the trends in methodology, technology, and in the market place? 4. Process Improvement for small organisations: Will all the presented programmes (ESI, ISCN) , methodologies and technologies (ami, BOOTSTRAP, CMM..etc.) also work for small and medium sized organisations? 5. Synergy effect: ISCN94 tries to discuss a synergy between different methodologies (ami, BOOTSTRAP, METKIT, SCOPE). How can other programmes and methodologies (e.g. TICKET, SPICE) contribute to such an approach ? Are there any US partners who want to co-operate with the European projects? Will this be accepted by the CEC and will this be fruitful?

With the registration on site you are asked to submit questions, issues, problem statements, or experiences for inclusion in the workshop programme. All inputs before lunch on May 4 are taken into account. Let us have an interesting workshop and fruitful discussions. Kaizen (Start to think about problems and solutions!; Always think about possible improvements in your work procedures) shall help you in this process.

H.J. Kugler
R. Messnarz