Causal Relationships between Improvements in Software Development Processes and Final Software Product Quality

Rini van Solingen[1] and Egon Berghout[2]
[1]Department of Software Technology, Delft University of Technology, The Netherlands
[2]Centre for IT Economics Research, University of Groningen, The Netherlands
d.m.van.solingen@tudelft.nl
e.w.berghout@rug.nl

Abstract: A main assumption of software process improvement (SPI) is that improvements in a software development process result in higher quality software products. In other words, SPI assumes the existence of causal relations between process and product characteristics. To what extent have these causal relations, however, been explored? Which specific process improvements have which particular impact on which particular product quality attributes?

In this paper an overview is given of these "software process and product dependencies" (PPD). This overview comprises of a list of SPI-techniques and the associated product quality attributes that are addressed with these techniques. The extent of the causality is investigated and whether there is a possibility to identify more or less effective strategies for product quality improvement. The overview is based on a literature study and expert evaluation.

The research is summarised in a matrix of both software process elements and associated software product quality characteristics. This matrix contains both satisfactory and unsatisfactory results. On the one hand, a promising extensive base of publications on techniques and methods was identified. On the other, a disappointing deficiency of empirical validation regarding the actual impact of those techniques on product quality is also prominent. As it is, we remain with an inadequate and incomplete indication of the product characteristics that particular software process improvement techniques intend to ameliorate. This article, therefore, hopefully, also provides a basis for discussion on the need to make process-product dependencies more explicit.

Keywords: software development, software process improvement, learning, product-process dependencies, PPD.

1. Introduction

Software engineering is a relatively young discipline in which control over process, products and resources appears to be difficult (Humphrey 1989). Time and cost overruns frequently occur, leading to all sorts of product quality problems. These quality mishaps may cause dangerous situations in all kinds of areas. Well-known examples are airplane accidents, space-project failures, and automobile malfunctions all due to quality problems with software (Gibbs 1994; Glass 1998).

A number of 'good software engineering practices' have been defined, which should result in a higher probability of success (Humphrey 1989). These practices particularly refer to elements of the software development process that need to be fulfilled. The underlying assumption here is, that in order to make a high quality product, one requires to have a high quality development process. Consequently, the process depends on the requirements of the product. Improvement of software processes with the aim to improve the product, are grouped under the name 'product focused SPI' (PROFES 1998; 1999).

In the recent years software engineering has a tendency to move towards standardization of models, techniques and approaches. All models, techniques and approaches are packaged in state-of-the-art practice methodologies to support and enhance the software engineering industry. The general notion is that standardization and a methodological approach with structured steps and progress metrics will lead towards improvement of software-products. In this paper an overview is presented of the relationships between these best-practices and particular product quality characteristics. This overview should provide insight in which improvement to the development process is more suitable to improve which quality characteristic of the software product.

The overview is described according to the following steps:

1. Developing a process/product matrix. In this matrix the impact of actions in a development process and on a software product is illustrated. The axis of the matrix are defined as well as the scale on which impact can be measured (Section 3).
2. Surveying literature for process actions and identifying the impact of these actions on product quality (Section 4). An expert panel supports this identification.

Reference this paper as:
van Solingen, R. and Berghout, E. "Causal Relationships between Improvements in Software Development Processes and Final Software Product Quality." *The Electronic Journal Information Systems Evaluation* Volume 11 Issue 1, pp. 1-10, available online at www.ejise.com

3. Drawing conclusions from the matrix (Section 5).

The research was performed as part of the quality management initiative at Tokheim Industries and the EEC PROFES project.

2. Defining a process-product relation matrix

In order to support decision making regarding process-product relations, there needs to be a kind of representation that links these two dimensions (Hamann *et al.* 1998; Solingen 2000)[1]. Questions that should, for example, be answered are:

- What actions can be taken to improve software usability?
- When a particular working method is improved, what are the consequences for, for instance, maintainability?
- What alternative actions are there for unit testing that might have similar effects?

The information on the process side of the matrix indicates what (type of) process actions are available in the software engineering domain. The information on the product side indicates which product quality characteristics are influenced by one or more process actions. The cells express the extent of this influence. The resulting matrix of process actions and quality characteristics will then indicate what the impact of a certain process action will be on a certain quality characteristic or a set of quality characteristics, or indicate what process actions are influencing a certain product quality characteristic.

The product life cycle has been used to define the process side of the matrix. This is, because techniques can be applied in different phases of the development life cycle and may have different effects in different phases. For example 'reviews' have different effects depending on the life cycle phase: during requirements their impact is primarily on improvement of the 'functionality', while during 'coding' their impact is primarily on 'reliability' and 'maintainability'.

After studying sources of available product life cycle definitions and available processes, the following life cycle definition was used (based on: Bicego et al 1994; PROFES 1998; Pfleeger 1991):

- Product Requirements Specification
- Product Architecture Development
- Design & Implementation
- System Integration and Testing
- Product Maintenance
- Product Improvement

All process actions that were identified in the literature, were categorised in the above life cycle stages. The detailed sub-phases of this life cycle model are listed in Appendix B.

The product side of the matrix is organised along a subdivision of software quality characteristics. Many subdivisions have been published in the past (e.g. (Boehm 1978; McCall *et al.* 1977). In this research we decided to select the ISO 9126 classification for software product quality (ISO/IEC 1994). The software quality characteristics are illustrated in Figure 1. More detailed definitions of the quality characteristics and sub-characteristics are listed in Appendix A.

[1] A working example of a repository of product-process dependencies can be found in the PROFES PPD Repository (PROFES 1999).

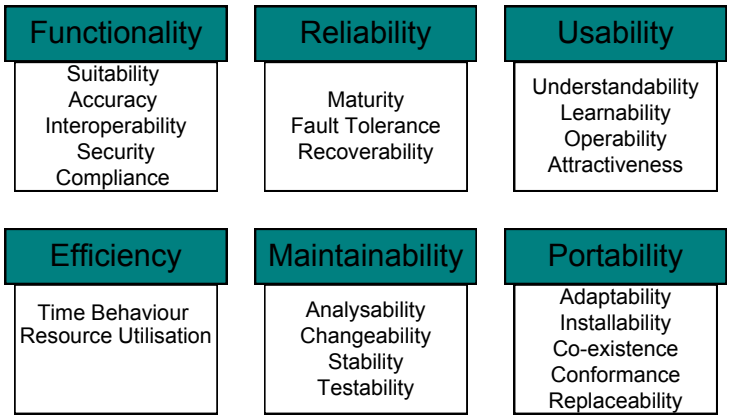

Figure 1: ISO 9126 software quality characteristics (ISO/IEC 1994).

A qualitative scale is used to express the extent of a process-product dependency, being:

+++ Highly positive impact, implying that the process action is expected to have a strong positive impact on a particular product quality characteristic.

++ Medium positive impact, implying that the process action is expected to have a significant positive impact on a particular product quality characteristic.

+ Low positive impact, implying that the process action is expected to have a modest impact on a particular product quality characteristic.

0 No Impact, implying that there is no impact expected (not to be confused by "unknown").

− Low negative impact, implying that the process action is expected to have a modest negative impact on a particular product quality characteristic.

−− Medium negative impact, implying that the process action is expected to have a significant negative impact on a particular a particular product quality characteristic.

−−− High negative impact, implying that the process action is expected to have a strong negative impact on a particular product quality characteristic.

For instance: n-version programming is expected to have a high positive impact on 'reliability', a high negative impact on 'maintainability' and a medium negative impact on 'efficiency'.

3. Searching for literature on process-product relationships

A literature study is used to identify the various process-product relationships. The search for literature was performed in several ways:

- Top-Down. In the top-down approach the software product life cycle model was used to find process actions for each phase within the life cycle model. As the breakdown of the model indicates, the life cycle model was broken down into activities for each phase. In this approach, process actions were identified, which could be applied in the according phase for the completion of some activity (as defined in the list of activities).
- Bottom-Up. In the bottom-up approach the software product quality model was used to find process actions that contributed to, or had had an impact on, a certain quality characteristic. In this approach, process actions were sought which could be applied throughout all phases of the life cycle model.
- Middle-both-ways. In the Middle-both-ways approach a certain software engineering technique or method, also known as process action, is selected to identify its impact on a certain set of quality characteristics and its place in the life cycle. In this approach, process actions can be selected from experience and the literature can give an indication of the impact of the process action and the placement of the process action. This approach is most effective when the impact of a certain process action needs to be identified without any prior references.

Studies of best-practices appeared to be most suitable. Although, the amount of research on software engineering is, of course, enormous, it was surprising to us, that in only few publications the effectiveness of new methods is validated (for instance, through case studies). It was even more surprising that so few publications indicate to which specific quality characteristic a technique contributes. Most publications stated that their approach 'largely increased product quality', without indicating what this actually meant and how significant this increase was. For the detailed outcomes of the literature survey we refer to (Soerjoesing

1999). Notable publications were (Wichmann 1997; Peng & Wallace 1993; Vliet 2000; Gilb 1988; Lyu 1996; Paul et al. 1993). In a publication by the Software engineering Institute (1997) several error detection techniques are presented, including their impact on quality characteristics. This guide also includes a preferred set of techniques to improve processes. Overall, more than 100 publications were reviewed to complete the matrix of Appendix C.

Due to the absence on information about the detailed process-product dependencies in the existing literature, we had to use an alternate approach to filling the PPD matrix: expert judgement. An expert panel interpreted the publications with respect to the matrix of Appendix C. This expert panel consisted of two business experts and one academic expert. On basis of their (embedded) software development background, they assessed the information provided with the various techniques and scaled the expected impact on a particular software product characteristic. This assessment was essential, because many of the papers provided insufficient information for an unambiguous placement in the matrix.

The experts all had an extensive background in embedded software development, however, primarily in one particular industry (Tokheim Industries). Consequently, this could cause a bias in the matrix making the results to some extent industry specific. We would like to challenge other organizations to validate the matrix in their organization and report on their experiences. It is not our intention to post the PPD matrix as the industry-wide externally valid model for product-process dependencies. The model is intended as a more detailed approach for particular companies to assess product quality by addressing specific process aspects. Usage of this PPD matrix in one specific organization for product quality estimation (based on the process model and project plan) is described in (Solingen 2000).

The PPD matrix of Appendix C contains 467 PPD's. From these PPD's, 424 PPD's have a positive effect and 43 a negative effect. All product characteristics are addressed by process actions. The smallest number is 3 (for instance, Functionality-Compliance). The largest number is 69 (Maintainability-Analysability). The following number of improvement actions refers to the following software product characteristics:

- Functionality: 103;
- Reliability: 85;
- Usability: 63;
- Efficiency: 51;
- Maintainability: 142;
- Portability: 23.

This implies that suggestions for improving particular software product characteristics can derived from the PPD matrix for every software product characteristic. Furthermore, possible additional effects of particular improvement actions can be derived from the matrix, These additional effects may, of course, be welcome or undesired.

As such, the PPD matrix turned out to be an excellent check list for improvement actions and an effective way to communicate the effects of software process improvements among the involved engineers.

4. Conclusions

In this paper an overview is provided of process improvement actions and their most likely effect on specific software product quality attributes. This overview is summarised in the matrix of Appendix C. This matrix should enable researchers and practitioners to identify more (and less) effective strategies for software process improvement on basis of particular product quality weaknesses. The relationships are based on an extensive literature review and assessed by an expert panel.

The matrix can be used as a checklist, however it should also be usable as a kind of knowledge base. In this case, the software engineers of an organization complete the various relations in the matrix on the basis of new experiences.

The proposed framework is also suitable for the validation of new techniques. Researchers that propose new techniques are encouraged to find practical evidence that their approach actually affects particular process and product relationships. We consider the identified disappointing empirical evidence of existing methods and techniques as a major improvement action for software engineering researchers.

Acknowledgements

The authors would like to thank Shyam Soerjoesing and all members of the PROFES consortium for their contribution to the work and concepts presented in this paper.

References

Bicego, S., Koch, P., Kuvaja, A., Krzanik, A., Mila, L., Saukkonen, G. (1994) "Software Process Assessment and Improvement: The BOOTSTRAP Approach", Blackwell Publishers.

Boehm, B.W., Brown, J.R. Kaspar, H., Lipow, M., McLeod, G., Merritt, M. (1978) "Characteristics of Software Quality", North Holland.

Curtis, B., Chrissis, M.B., Paulk, M.C., Weber, C.V. (1993) "Key practices of the Capability Maturity Model", Software Engineering Institute, Version 1.1. SEI-CMU-93-TR-25

Gibbs, W.W. (1994) "Trends in Computing: Software's Chronic Crisis", Scientific American, Vol. September, pp.86-100.

Gilb, T. (1988) "Principles of software engineering management", Addison Wesley.

Glass, R.L. (1998) "Software Runaways", Pearson Education POD.

Hamann, D., Järvinen, J., Oivo, M., Pfahl, D., (1998) "Experience with explicit modelling of relationships between process and product quality", Proceedings of the 4th European Software Process Improvement Conference, Monte Carlo, Vol . December.

Humphrey, W.S. (1989) "Managing the Software Process", Addison-Wesley.

ISO/IEC, (1994) Information Technology -Software quality characteristics and metrics- "Quality characteristics and subcharacteristics", part 1.

Lyu, M.R. (1996) "Software Reliability Engineering", IEEE CS Press, McGraw-Hill.

McCall, J.A., Richards, P. K., Walters, G. F. (1977) "Factors in Software Quality", National Tech. Information Service, Springfield, AD/A-049-014/015/055, Vol. 1,2 and 3

Peng, W.W., Wallace, D.R. (1993) "Software Error Analysis", National Institute of Standards and Technology, Gaithersburg, NIST Special Publication, March, pp 500-209.

Pfleeger, S.F. (1991) "Software Engineering: the production of quality software", McMillan Publishing, New York.

PROFES PPD (1999) Repository [online]: http://www.vtt.com/, PROFES Consortium.

PROFES User Manual (1998) [online]: http:/www.vtt.com/, PROFES Consortium.

Soerjoesing, S P. (1999) "Software quality improvement through product-process dependency modelling", Tokheim Report, Bladel. An electronic version of this report can be received on request to the authors.

Software engineering Institute (1997) "C4 Software Technology Reference Guide-A Prototype", Carnegie Mellon University, Pittsburgh, CMU/SEI-97-HB-001, January.

Solingen, R van (2000) "Product focused software process improvement: SPI in the embedded software domain", BETA Research Institute, Eindhoven.

Vliet, J.C. (2000) "Software Engineering", Principles and Practice 2nd Edition, Wiley.

Wichmann, B.A. (1997) Measurement Good Practice Guide No.5, Information Systems Engineering National Physical Laboratory, National Physical Laboratory, Middlesex, November.

Appendix A: ISO 9126 quality characteristics

Functionality	The capability of the software to provide functions which meet stated and implied needs when the software is used under specified conditions
Suitability	The capability of the software to provide an appropriate set of functions for specified tasks and user objectives
Accuracy	The capability of the software to provide the right or agreed results or effects
Interoperability	The capability of the software to interact with one or more specified systems
Compliance	Attributes of software that make the software adhere to application related standards or conventions or regulations in laws and similar prescriptions.
Security	The capability of the software to prevent unintended access and resist deliberate attacks intended to gain unauthorised access to confidential information, or to make unauthorised modifications to information or to the program so as to provide the attacker with some advantage or so as to deny service to legitimate users
Reliability	The capability of the software to maintain the level of performance of the system when used under specified conditions
Maturity	The capability of the software to avoid failure as a result of faults in the software
Fault-tolerance	The capability of the software to maintain a specified level of performance in cases of software faults or of infringement of its specified interface
Recoverability	The capability of the software to re-establish its level of performance and recover the data directly affected in the case of a failure
Usability	The capability of the software to be understood, learned, used and liked by the user, when used under specified conditions
Understandability	The capability of the software product to enable the user to understand whether the software is suitable, and how it can be used for particular tasks and conditions of use
Learnability	The capability of the software product to enable the user to learn its application
Operability	The capability of the software product to enable the user to operate and control it
Attractiveness	The capability of the software product to be liked by the user
Efficiency	The capability of the software to provide the required performance, relative to the amount of resources used, under stated conditions
Time behaviour	The capability of the software to provide appropriate response and processing times and throughput rates when performing its function, under stated conditions
Resource Utilisation	The capability of the software to use appropriate resources in an appropriate time when the software performs its function under stated conditions
Maintainability	The capability of the software to be modified
Analysability	The capability of the software product to be diagnosed for deficiencies or causes of failures in the software, or for the parts to be modified to be identified
Changeability	The capability of the software product to enable a specified modification to be implemented
Stability	The capability of the software to minimise unexpected effects from modifications of the software
Testability	The capability of the software product to enable modified software to be validated
Portability	The capability of software to be transferred from one environment to another
Adaptability	The capability of the software to be modified for different specified environments without applying actions or means other than those provided for this purpose for the software considered
Installability	The capability of the software to be installed in a specified environment.
Conformance	Attributes of software that make the software adhere to standards or conventions relating to portability.
Co-existence	The capability of the software to co-exist with other independent software in a common environment sharing common resources
Replaceability	The capability of the software to be used in place of other specified software in the environment of that software

Appendix B: List of life cycle phases for process dimension

1 Product requirement specification
1.1 Product requirement definition
1.1.1 Requirements gathering
1.1.2 Requirements definition
1.1.3 Requirements review
1.2 Feasibility study
1.2.1 Requirements analysis
1.2.2 Architecture specification
1.2.3 Project plan
1.2.4 Risk analysis
1.2.5 Technologies evaluation
1.2.6 Available designs evaluation
1.2.7 Product test plan Preparation
1.3 Application requirement specification
1.3.1 Product requirement analysis
1.3.2 Feasibility analysis
1.3.3 Application requirements specification
2 Product design
2.1 Functional design
2.1.1 Detailed product function design
2.1.2 Behaviour function analysis
2.1.3 Function structuring
2.1.4 Funct. product component mapping
2.1.5 User manual (functional) prototype
2.2 Architectural design
2.2.1 Detailed product architecture design
2.2.2 Design review
2.2.3 Technical product documentation
3 System design and implementation
3.1 Mechanical design and implementation
3.1.1 Mechanical design
3.1.2 Mechanical documentation
3.1.3 Mechanical design review
3.1.4 Mechanical component implementation
3.1.5 Testing of the mechanical components
3.2 Electronics design and implementation
3.2.1 Electronics design
3.2.2 Electronics documentation
3.2.3 Electronics design review
3.2.4 Electronic component implementation
3.2.5 Testing of the electronic components
3.3 Software design and implementation
3.3.1 Software design
3.3.2 Software documentation
3.3.3 Software design review
3.3.4 Software component implementation
3.3.5 Testing of the software components
4 System integration and testing
4.1 Product construction
4.1.1 Product components integration
4.1.2 Integration testing
4.1.3 Product documentation
4.1.4 Prototype building
4.2 Product validation
4.2.1 Unit testing
4.2.2 Prototype review
5 Production and Maintenance:
5.1 Pilot series production
5.2 Pilot series audit
5.3 Product installation
5.4 Product maintenance
5.5 Customer training
5.6 Customer support
6 Product Improvement:
6.1 Maintain product design
6.2 Defect reporting/tracking
6.3 Collecting Feedback
6.4 Monitor product quality
6.5 Product improvement area identification
6.6 Product follow up planning
6.7 Product improvement increment

Appendix C: Product and process dependencies matrix [7]

		Functionality					Reliability			Usability				Efficiency		Maintainability				Portability				
		Suitability	Accuracy	Interoperability	Compliance	Security	Maturity	Fault-tolerance	Recoverability	Understandability	Learnability	Operability	Attractiveness	Time-behaviour	Resource-Utilisation	Analysability	Changeability	Stability	Testability	Adaptability	Installability	Conformance	Co-existence	Replaceability
Product Requirements Specification	Argument-Based Design for Requirements	+++								+++						+++								
	Feature-Based Design for Requirements Tracing									+++						+++		+++						
	Feature-Oriented Domain Analysis	++								+++							+++							
	Quality Function Deployment	+++														++								
	Requirements Inspections	+++								+		+	+											
	Requirements Metrication Report	+++								+		+				++								
	Requirements Reviews	+++								++		++				++								
	Requirements Tracing and Parsing	+++	+++							+++		+++				+++								
	Software Requirements Review	+++								+		+				++								
	Software Requirements Specification	+++														++								
	Stress testing						++							+++	++									
	Systems Requirements Analysis	+++										+				++								
	Systems Requirements and Design Review	+++								+		+				++								
Product Architecture	Architectural Description Languages		+++				+++																	
	Control Flow Analysis/Diagrams	+++														+++								
	Data Flow Analysis/Diagrams	++														+++			++					
	Design Reviews	+++	+				++																	
	Design Verification Reporting	++	++				+									++								
	ERD Diagrams	++														+++								
	Formal Design Specification		+++				+++									+++								
	Formal Design Verification		+++				+++			+++						+++			+++					
	Function Point Analysis	+++	+++				+++																	
	Hardware Prototyping (Design Validation)	+++								+++		+++	++											
	Hybrid State Machines						++									+								
	Object-Oriented Design	++	+				++									+++								
	Simulation And Animation	+++								++		++	++	++	++	++								
	Systems Specification and Design	++														++			++					
	Systems Requirements and Design Review	+++								+		+				++								

		Functionality					Reliability			Usability				Efficiency		Maintainability				Portability				
		Suitability	Accuracy	Interoperability	Compliance	Security	Maturity	Fault-tolerance	Recoverability	Understandability	Learnability	Operability	Attractiveness	Time-behaviour	Resource-Utilisation	Analysability	Changeability	Stability	Testability	Adaptability	Installability	Conformance	Co-existence	Replaceability
Design & Implementation	FourGL-Programming						++							--	---		++							
	Abstraction				++									-	--	+++	+++			+++	++	+++	+++	+++
	Algorithm Analysis		+++				++	+++						++	-								+++	
	Algorithm Formalization															++								
	Algorithm Optimization																							
	Application Programming Interface			+++	++												++	++						
	Backward Recovery (Runtime Error Correction)		+++					+++	+++					-	-	+++								
	Certified Components	++	+++		+++	+++	+++			+		++	+		-	--	-	++	0	0		+	0	
	Checkpoint (Runtime Error Detection)							+++	+++															
	Cluster Implementation							+++	+++						-									
	Code Reuse	++	---				+++									++	++	+++						
	Coding Standards															+++	++		+					
	Compensation (Runtime Error Correction)		+++					+++	+++						-	--								
	Compilation						+																	
	Compression (Unit/Component Operation)													+	+++	---	---			--			0	
	Data Encryption			--		+++								---	--		--		--					
	Database Interaction Analysis		+++			+++																		
	Description of Design Methods and Tools																							
	Description of Programming Languages and																							
	Description of Software Specification Methods																							
	Exception handling (Runtime Error Correction)							+++	+++															
	Executable Code Reporting																							
	Expression Adjustability Analysis		+++													++	+++							
	Expression Computability Analysis		+++				+++	+++								-			-					
	Expression Effectiveness Analysist		-											+++	+++									
	Expression Side-Effects Analysis		+++				+									+			+++					
	Forward Recovery (Runtime Error Correction)		+++					+++	+++							+++	-		+					
	Graphical User Interface Builders											+++	++	-	-		+++	++						
	Input Checking Mechanisms		+++					+++			++	+++												
	Module Interconnection Languages		++	++			+									+++							++	
	N version programming (Runtime Error Handling)		++					+++																
	Software Source Code Inspections		+++				+++									+++			+					
	Source Code																							
	Source code commenting															+++	+++		++					
	Source Code Metrication Reporting						+									++								
	Source code reading																							
	Source Code Review Report						++									+++								
	Object-Oriented Programming Languages																++	++	++	+++				
	Program Description						+									++								
	Recovery Blocks (Runtime Error Detection)							+++	+++						-	++	-							
	Replication (Unit/Component Operation)							+++						-	--		---			--				
	Software Design Description	++	++													++								
	Software Design Review	++	++				+									++			++					
	Software Design Walkthroughs	++	+++				+++									+++			+					
	Principal Component Analysis		+++				+++									+++	++							
	Queueing Theory Analysis	+												+++	+++									
	Resource Sharing (Unit/Component Operation)							---	---						---		+++						+++	
	Scheduling Theory Analysis	+												++	++									
	Separation (Unit/Component Operation)		-					++									+++						+++	
	Rate monotonic Analysis						+++							+++	+++	++								
	Software Prototyping (Requirements Validation)	+++								+++	+++	++	++											
	Watchdogs							+++																
	Variable Assignment Convention Application		+++													+			+					
	Variable Commenting															+++								
	Variable Naming Conventions															+++			+					

		Functionality					Reliability			Usability				Efficiency		Maintainability				Portability				
		Suitability	Accuracy	Interoperability	Compliance	Security	Maturity	Fault-tolerance	Recoverability	Understandability	Learnability	Operability	Attractiveness	Time-behaviour	Resource-Utilisation	Analysability	Changeability	Stability	Testability	Adaptability	Installability	Conformance	Co-existence	Replaceability
System Integration and Testing	Back-to-back Testing						+++																	
	Bad Data Testing (Unit Testing)		+																					
	Black Box Testing (Unit Testing)	++	++				++					++												
	Boundary Value Testing (Unit Testing)		++				+++	+++																
	Code Coverage Metrication	+					+++																	
	Cyclomatic Complexity Metrication						+									+++			+++					
	Discriminant analysis (Software Quality																							
	Efficiency and Performance Analysis													+++	+++									
	Field Testing	+++	+++				+++	0	+	++		++		++	++						++			
	Full Code Coverage Testing (Unit Testing)		+++				+++							+					++					
	Good Data Testing (Unit Testing)		+++				+							+					+					
	Integration testing		++				+++	+																
	Interface Testing And Analysis		++	++			++																++	
	Halstead Complexity Metrication						++									++	++		++					
	Logistic Regression (Software Quality Predictor)		+++				+++																	
	Min-Max Testing [Boundary Value Test]		+++				+++																	
	Object-Oriented Metrics for complexity reduction	0	++			0	+++			+++				0	0	++		0	+++					0
	Penetration analysis					+++	+					+												
	Regression Testing		+++				+++											+++						
	Source Code Test Planning and Reporting						++									+			+++					
	System Test Planning and Reporting	++	+				++		+	+		++		++	+	+			+					
	Testing	+++	+++				+++		+++	+		++		++	++	+			+					
	Unit Test Planning and Reporting	++	++				++							+	+	++			+					
	White Box Testing (Unit Testing)		+++																+					
Product Maintenance	Maintainability Index Technique															++	++	++	++					
	On-line Software Uploading												+		-						++	0	0	+++
Product Improvement	Video-assisted Product Evaluation (Usability)	++	+				+			+++	+++	+++	+	+++										
	Usability inspection	+								+++	+++	+++	++											
	User Documentation									+++	++	+++				+								
	User Documentation Review	++								++		++				+								
Baseline	Cleanroom Software engineering						+++			+++						+++								
	Component-Based Software Development						+++									--	++	+	--					
	Configuration Management	++														+	++	+	++					
	Document templates						+									+++	++		++					
	Organization Domain Modeling											++				+++								
	Personal Software Process for Module-Level		+++				+++									+++	++	++	++					
	Problem Reporting	++	++				+++									++		++						
	Quality Assurance Planning and Reporting																							
	Simplex Architecture						+++	+++							-	--	-	++	+					
	Software Development Planning and Tracking															++								
	Log files		++				++	++	++						+	+			+					
	Quality Profiling (Reliability)																							
	System Manuals									+						++								

