Applying Testing Techniques to Software Process Assessment: A Model-Based Perspective

L. García-Borgoñón, R. Blanco, J.A. García-García,and M.A. Barcelona

Abstract Software processes constitute a major asset for an organization. However, in many occasions there are differences between defi ned processes and executed processes. For this reason, organizations spend time and effort of their resources to fi nd these non-conformances. The use of software testing techniques could be a use-ful way to reduce these costs. This paper proposes a model-based approach and shows how software testing techniques can be applied to evaluate the execution conformity in a software processes context, and also to evaluate the model designed. A real execution of a NDT methodology process by means of the process model included in NDTQ-Framework (a solution based on this approach that is currently being used in software development organizations) illustrates the fi nal results. Finally, conclusions and future work are stated.

Keywords Model-Driven • Testing • SoftwareProcesses

27.1 Introduction

Software processes are recognized as fundamental assets in software-intensive organizations, since they support their capability to produce better products. Process defi nition, documentation, management and improvement have become in

L. García-Borgoñón (✉) • M.A. Barcelona
AragónInstituteof Technology, c/MaríadeLuna7 , 50018 Zaragoza, Spain
e-mail: laurag@ita.es; mabarcelona@ita.es

R. Blanco
University of Oviedo, Campusde Viesques, S/N33204 Gijón, Spain
e-mail: rblanco@uniovi.es

J.A. García-García
University of Seville, Av. Reina Mercedes, S/N41012 Sevilla, Spain
e-mail: julian.garcia@iwt2.org

organizations' key practices what is known as Business Process Management (BPM). Since business in a software-intensive organization is software development, software processes constitute the BPM focus within such organization [1].

A software process is a set of activities, methods and practices used in the production and evolution of software and the associated products [2, 3]. These processes and methodologies have always been described in appropriate terms to be used by a developer, but they are often described in manuals or books which project team follow as closely as possible [4].

However, differences can usually be noticed between organizations' defined processes and really executed processes in a specific project context. It occurs due to several causes, such as process described in unsuitable way, like could be natural language and misunderstandings which involves, or the existing gap between processes definition and execution. Thus, organizations draw on process and product assessment activities to solve this problem. Process assessment is a disciplined evaluation of an organizational unit processes against a process assessment model [5], which provides a set of indicators used for evaluating the effective process performance and management [6]. An assessment model can either represent the defined process or be based on one or more Process Reference Models [7].

These activities are manually executed, normally by quality offices, since processes orchestration are not widely used in software-intensive organizations. They verify and control, through a checklists set, that the process is followed properly, establishing a non-conformance record in cases of deviations between defined and executed processes. Non-conformances should be solved in a concrete time with a specific commitment. The cost of this kind of activity is usually called Quality Cost, and organizations assume it as necessary.

Besides, and particularly in periods of economic crisis like the one we are living, where optimization effectiveness and resource efficiency are essential, one of main objectives deals with reducing nonproductive costs.

In the last years, the Model Driven Engineering (MDE) [8] has been established as a common approach for software development [9], what has shaped the software industry to be model-centred. In addition, software testing is a very important phase in software development and maintenance, as it aims to find out faults in software products, thus helping developers to improve the quality of these products when the discovered faults are solved and reducing the cost produced by these faults.

As software processes are software too [10], this paper evaluates how a model-based approach working in liaison with testing software techniques can make easier activities related to performed processes evaluation, reducing their time and effort cost.

This paper is structured as follows: After this introduction, Sect. 27.2 shows the main work related to software processes evaluation, also known as Software Processes Assessment. Section 27.3 introduces some concepts related to testing techniques and describes how they are used in our approach. Then, Sect. 27.4 presents the proposed metamodel and Sect. 27.5 illustrates result in the NDTQ-Framework, a solution based on this approach that is currently being used in software-development organizations. Finally, Sect. 27.6 outlines conclusions and proposes future work in these lines of research.

27.2 Related Work

An organization's software process assessment is an activity especially related to software process improvement and, therefore, there are several proposals concerning it, particularly focused on integrating software process assessment and software process modeling. Later, more remarkable proposals on this topic will be pointed out.

OOSPICE is a project associated with the capability assessment space [11], although it also delivers methodology components in ISO 12207 [12], ISO 15504 and method engineering context. It copes with the concepts posed for assessing an organization's process enactment quality, through a metamodel with metaclasses such as Outcome as well as attributes relating to capability level on Process and Task. It is oriented towards capability appraisals against ISO 15504.

In [13] Hamann proposes an information model which integrates software process assessment and process modeling. It has basic elements based on process modeling such as Process, Product, Role and Tool, with the appropriate attributes and classes related to assessment information, such as Rating and Purpose. Makinen et al. [5, 14, 15] slightly modify Hamann's model, to make it more general and illustrative by classifying the elements into three categories: Assessment Model, Assessment Result and Modeling Result.

Henderson-Sellers and Gonzalez-Perez [4] propose a new standard metamodel to define and assess software processes with the same elements, but including two new concepts, powertypes and clabjects, as a way to solve problems derived from modelling both the methodology and project layers at the same time.

Despite evaluation concepts are similar, all these proposals are focused only in assessment based on a reference model, but they do not address the issue of assessing software process executed against software process modeled. That is the goal of our approach, which is presented in the following sections.

27.3 Applying Software Testing Techniques to Test Software Process

This section describes our approach with the aim of both, evaluating the executions conformity of the software process that has been modeled and measuring the level of acceptance of this model. Section 27.3.1 presents an overview of software testing and Sect. 27.3.2 describes the applications of software testing techniques to test a software process.

27.3.1 Software Testing Overview

Software testing is part of the Verification and Validation process (V&V) [16]. It determines whether the developed products fulfill the requirements and satisfy the user's needs. Software testing deals with verifying the behavior of a system that is

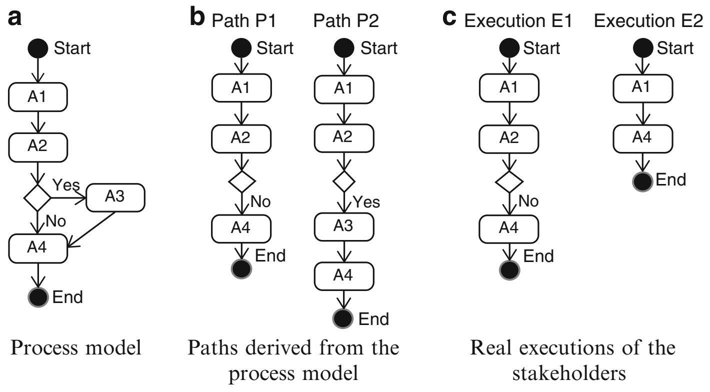


Fig. 27.1 Introductory example

executed under specific conditions, against the expected behavior. This evaluation is carried out according to some aspects of the system [17].

Reactive and proactive approaches have been developed to achieve this goal. Proactive approaches try to detect faults in the Software Under Test (SUT) before they produce failures in an operational environment. For example, the SUT can be executed with a test case and, after that, the observed behavior and the expected behavior can be compared to detect any deviation (a test case is a set of test input data, execution conditions and expected results [17]). Reactive approaches identify faults after they produce failures. For example, a monitoring-based technique can detect any deviation from the expected behavior of the SUT by observing its real time execution.

On the other hand, several adequacy criteria have been defined [18] to specify the situations of interest to be tested (to be covered), which constitute the test requirements, and determine whether sufficient testing has been done. For instance, the path-testing criterion requires executing all or selected paths of the SUT (each path is a test requirement). Measuring the test coverage achieved, that is the degree to which the tests execute (cover) the paths of the SUT, it is possible to determine whether the testing process can be stopped.

27.3.2 Problem Approach

As previously mentioned, software processes can be also considered software [10], therefore, it is possible to apply software testing techniques to test their conformity.

We will consider the introductory example depicted in Fig. 27.1, to illustrate the approach and objectives. We will use UML [19] for these examples because they are very intuitive, but other graphic representation language, such as BPMN [20], could be used. Part (a) shows a UML activity diagram that models a software process. The process model contains two correct activities paths (P1 and P2), which are shown in part (b). These paths represent the different scenarios of the software process modeled. Part (c) offers two UML activity diagrams that represent the real sequences of activities executed by the stakeholder (E1 and E2), which have been monitored. Analyzing the process model and the executions poses the following questions:

1. Do the executions E1 and E2 satisfy the process model?
2. If the path P2 is not followed (*covered*) by any execution, is it really necessary?

We have developed an assessment approach that combines reactive testing with path-testing criterion, to answer these questions. The goals of our approach are (1) to evaluate whether the executions of a software process satisfy the process model defined, and (2) to determine the paths of the process model that have not been covered by these executions and measures the degree of path coverage achieved. The following sections describe both goals.

27.3.2.1 Evaluating Executions Conformity

A testing process is carried out to evaluate executions conformity with the process model. First, each transition of a monitored execution is classified as *valid* or *invalid*. It is considered valid when it also appears in the process model (for instance, the transition from A1 to A2 in the execution E1 in Fig. 27.1), otherwise it is considered invalid (for instance, the transition from A1 to A4 in the execution E2 in Fig. 27.1).

After that, our approach analyzes the classification of all transitions of the execution to determine its conformity with the process model. An execution satisfies the process model only if all transitions have been classified as valid (for instance, execution E1 in Fig. 27.1). Otherwise, the execution does not satisfy the process model (for instance, execution E2 in Fig. 27.1).

On the one hand, an execution that satisfies the process model can be considered a positive test, as it tries to check the behavior of a specific scenario of the process model. Thus, it is called *Positive Execution* in our approach. On the other hand, an execution that does not satisfy the process model can be considered a negative test, as it tries to cover a scenario for which the process model has not been designed. Consequently, it is called *Negative Execution* in our approach.

27.3.2.2 Evaluating Path Coverage

Finding the paths of the process model that have not been followed by any execution can be useful to check the correctness of this model, and also to evaluate whether the software process is correctly executed. If a path is never covered, several reasons can be considered: (1) the path does not represent a scenario of the software process and therefore the model must be improved; (2) the path represents a scenario that is not very common but necessary, so the model is correct; and (3) the path represents a very common scenario that is not being considered by the stakeholder, so maybe the software process is not being correctly executed.

Our approach applies the path-testing criterion to the process model to address this issue. This way each path represented in the model is considered a test requirement, which is covered when a positive execution satisfies the path. Measuring the path coverage achieved allows observing the level of acceptance of the process model, due to the coverage is increased when a stakeholder executes a set of activities of the software process that is modeled by a path in the process model.

Considering the example in Fig. 27.1, the path testing criterion derives two test requirements: paths P1 and P2. Path P1 is covered by the positive execution E1 and path P2 is not covered by any positive execution then, the path coverage achieved is 50 %.

27.4 A Metamodel for Software Process Assessment

As mentioned above, many approaches have been developed in order to assess software processes with a reference model, but they have not been used to assess software processes executed against software processes modeled. This issue and the use of MDE to manage the conceptual complexity have been the basis of our proposal, that is, designing a metamodel for testing software process models.

This approach is presented in Fig. 27.2 and it is an extension of the software process metamodel presented in [21]. Besides, we present metaclasses from the extension metamodel.

The *Action* metaclass is the main class in the testing metamodel. We define an *Action* for each *Activity* aimed to test. It is possible not to test all activities in a process, so this relationship guarantees it. The attributes in this metaclass are: the start and end date of the action, the status of the action in a specific moment and the test result, where the result of the testing process performed on the action is recorded.

An *Action* has a set of preconditions and postconditions. The *Precondition* metaclass represents the previous action we need to check to be able to execute it. The *Postcondition* metaclass shows the following action and whether it is properly executed.

The *WorkProduct* metaclass represents a piece of the *Product* that is developed in each action, so that a *Product* could be considered as a collection of several *WorkProducts*. Finally, the *Stakeholder* represents someone or a tool that has actually executed the *Action*.

The main feature of our approach is that the metamodel extension allows us to test a process model which has been defined with the initial metamodel. This will enable us to establish testing points in the definition moment.

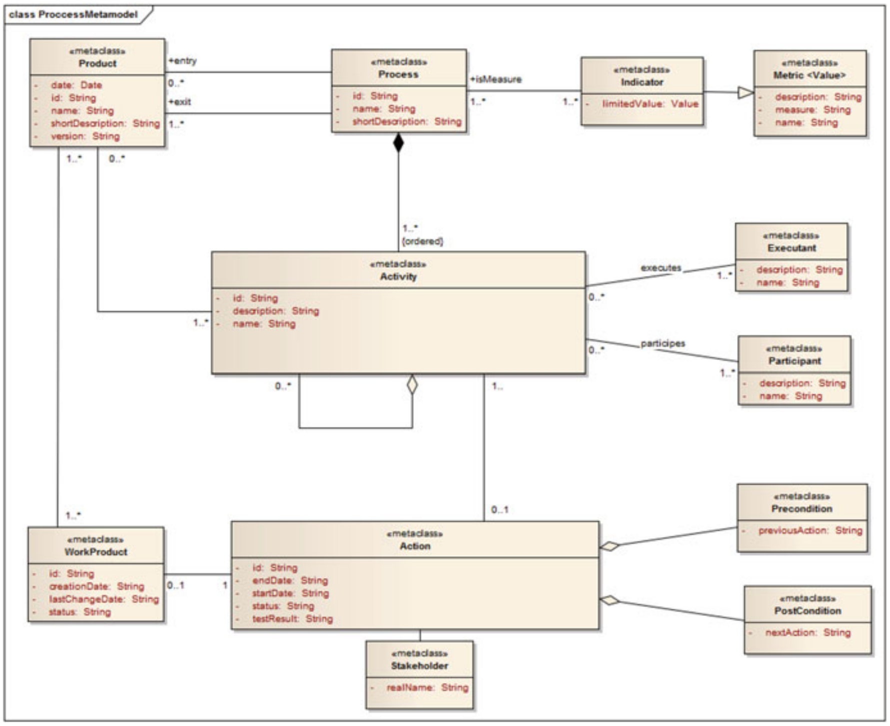


Fig. 27.2 Testing process models metamodel

27.5 NDTQ-Framework: A Practical Example

This section includes a practical example to illustrate our approach. We use processes currently supported by NDT [22]. It is a methodology that defines metamodels for every phase of software development life cycle by providing a framework that make easier the use of new methods, standards and paradigms and as a result, it helps us improve software development quality. NDT uses different software processes, each one supported by main models, standards and rules related to the field where it is defined. These processes are classified in six groups:

- Software Development Processes. They are defined in terms of NDT life cycle.
- Software Maintenance Processes. They are based on ITIL [23] and CMMI [2].
- Testing Processes. They are based on ISO/IEC 29119 [24] standard.[1]
- Software Quality Processes. They are based on ISO 9001:2008 [25] and CMMI.

[1] ISO 29119 has not been yet approved completely, but they are based on the group of processes already defined.

- Project Management Processes. They are based on PMBOK [26] and CMMI.
- Security Processes. They are based on ISO 27001 [27] standard.

It is necessary a real deployment supported by tools to allow a software-intensive organization using this methodology to benefit from all its potential. This enhances its use and accomplishment. NDTQ-Framework was created with this goal in order to support all processes defined by NDT.

NDTQ-Framework is a framework implemented on Enterprise Architect tool, developed by means of the UML profile presented in [21]. It is based on the metamodel our approach has extended, presented in Sect. 27.4, to achieve the goal we are looking for.

27.5.1 A Process Example: Requirements Engineering Process

We are going to use the requirements engineering process as an example, to show how the testing approach presented in Sect. 27.3 and the metamodel described in Sect. 27.4 work together. Figure 27.3 shows the map of activities of this process.

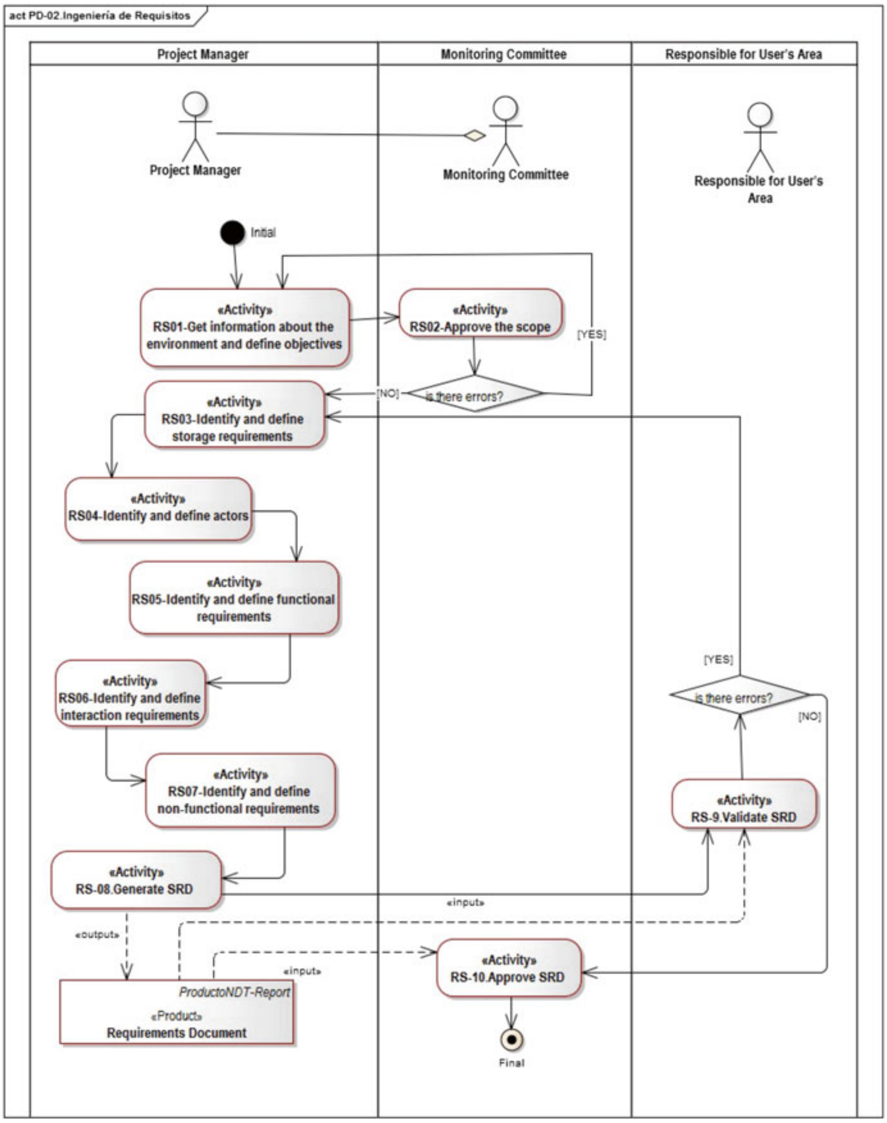
27.5.2 Applying Testing Techniques to Requirements Engineering Process Assessment

We have considered that all activities have an action associated, and preconditions and postconditions are previous and next actions respectively. In this case, we are not going to use the WorkProduct concept to explain the testing techniques used.

We are going to consider the execution shown in part (a) of Fig. 27.4, which has been obtained from a working report tool, to illustrate how to evaluate a requirement engineering process execution conformity with the process model represented in Fig. 27.3. This tool registers the real data demanded by the process model, such as the start and end dates for each action performed as well as the status. Actions *RS01*, *RS02*, *RS03* and *RS10* represent the execution of the corresponding activities of the process model. The action *Condition1* represents the execution of the conditional activity after *RS02*, and the status indicates the answer obtained.

The result of the testing process performed is shown in part (b) of Fig. 27.4. Along this process, each action is evaluated as valid or invalid. We analyze the data registered by the working report tool and the knowledge obtained from the model about which are its previous and next actions in order to determine the test result.

For instance, the action *RS02* is valid because it starts after its previous action *RS01* has finished. That means that the execution of actions *RS01* y *RS02* represents a valid transition, as it is present in the process model, whereas the action *RS10* is evaluated as invalid. This action starts after the other actions of the working report have finished, nevertheless none of them constitutes its previous action.

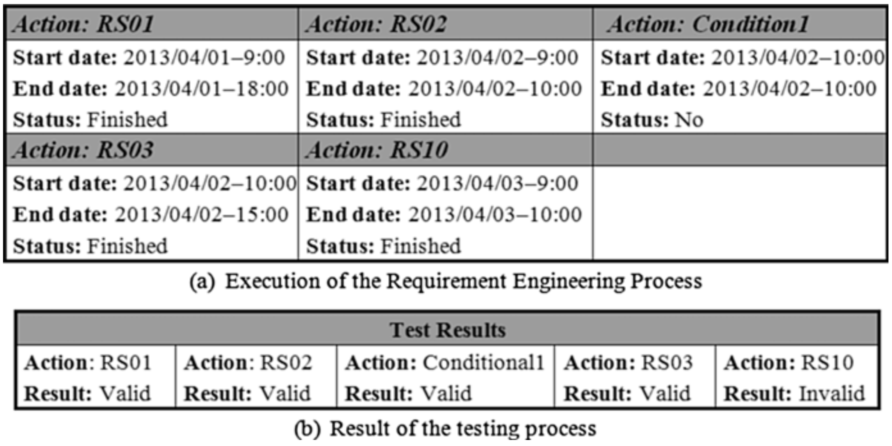


act PD-02.Ingeniería de Requisitos
Project Manager
Monitoring Committee
Responsible for User's Area
Project Manager
Monitoring Committee
Responsible for User's Area
Initial
«Activity» RS01-Get information about the environment and define objectives
«Activity» RS02-Approve the scope
[YES]
[NO]
is there errors?
«Activity» RS03-Identify and define storage requirements
«Activity» RS04-Identify and define actors
«Activity» RS05-Identify and define functional requirements
«Activity» RS06-Identify and define interaction requirements
«Activity» RS07-Identify and define non-functional requirements
«Activity» RS-08.Generate SRD
[YES]
is there errors?
[NO]
«Activity» RS-9.Validate SRD
«input»
«output»
«input»
ProductoNDT-Report
«Product» Requirements Document
«Activity» RS-10.Approve SRD
Final

Fig. 27.3 Map of activities of the requirements engineering process

The test result is invalid, because it is not possible to find a valid transition for action *RS10*.

As a conclusion, it should be mentioned that the execution does not satisfy the process model, since the test result of at least one action is invalid. Therefore, an inconsistency has been identified during the software process execution.



Action: RS01	*Action: RS02*	*Action: Condition1*
Start date: 2013/04/01–9:00 **End date:** 2013/04/01–18:00 **Status:** Finished	**Start date:** 2013/04/02–9:00 **End date:** 2013/04/02–10:00 **Status:** Finished	**Start date:** 2013/04/02–10:00 **End date:** 2013/04/02–10:00 **Status:** No
Action: RS03	*Action: RS10*	
Start date: 2013/04/02–10:00 **End date:** 2013/04/02–15:00 **Status:** Finished	**Start date:** 2013/04/03–9:00 **End date:** 2013/04/03–10:00 **Status:** Finished	

(a) Execution of the Requirement Engineering Process

Test Results				
Action: RS01 **Result:** Valid	**Action**: RS02 **Result:** Valid	**Action:** Conditional1 **Result:** Valid	**Action:** RS03 **Result:** Valid	**Action:** RS10 **Result:** Invalid

(b) Result of the testing process

Fig. 27.4 Example of an execution of a software process and the evaluation of its conformity

This example represents a negative test that is useful to identify inconsistencies in the process model. However, positive tests, which cover the different paths of the process model, must be considered to measure the path coverage that allows evaluating the level of acceptance of the process model.

27.6 Conclusions and Future Work

Nowadays, business processes constitute a very important asset for organizations in general and software-intensive organizations are not an exception. Defining, documenting and managing these processes require techniques and tools to support their application and maintenance. Nevertheless, these are not enough. Once the process is defined and deployed in an organization, it is mandatory to verify that it must be executed as it was defined. Usually, verification actions are mainly manual activities demanding an important effort and time cost. That is something that organizations assume as quality cost, although they need to reduce it as much as possible.

This paper presents a solution for automating this activities founded on a model-based approach and on the application of software testing techniques. This solution is offered by a metamodel and illustrated by a concrete solution named NDTQ-Framework.

This approach will improve in different ways as a future work. Firstly, we are working on obtaining more empirical data about its use in software-intensive organizations. This would allow establishing our approach as a continuous improvement mechanism according to the comments discussed in Sect. 27.3.2.2, which is widely recommended in many standards and good practices manuals. Getting data from these testing techniques may allow organizations to identify problems and make easier decision-making processes. Secondly, this approach can support an orchestration mechanism that conceives NDTQ-Framework as a whole solution for process definition and execution.

Finally, the approach can be used as assessment model in formal appraisals using a reference model like CMMI or ISO 15504, by helping obtain evidences as requested by these standards and models.

Acknowledgements This work has been supported by the projects TEMPROS (TIN2010-20057-C03-02) and Test4DBS (TIN2010-20057-C03-01) of Ministerio de Educación y Ciencia, Spain, and the NDTQ-Framework project (TIC-5789) of the Junta de Anda-lucía, Spain.

References

1. Bendraou R, Gervais MP (2007) A framework for classifying and comparing process technology domains. In: International conference on software engineering advances, ICSEA 2007. Cap Esterel, French Riviera, France. IEEE, pp 5–5
2. Chrissis MB, Konrad M, Shrum S (2003) CMMi. Addison-Wesley, Boston
3. Humphrey WS (1989) Managing the software process (Hardcover). Addison-Wesley Professional, Reading
4. Henderson-Sellers B, Gonzalez-Perez C (2005) A comparison of four process metamodels and the creation of a new generic standard. Inf Softw Technol 47(1):49–65
5. Makinen T, Varkoi T, Soini J (2007) Integration of software process assessment and modeling. In: Portland International Center for Management of Engineering and Technology, Portland, PICMET 2007. IEEE, pp 2476–2481
6. Coletta A (1997) Process assessment using spice: the assessment activities. SPICE: The theory and practice of software process improvement and capability determination, Computer Society Press. IEEE, pp 99–122
7. ISO/IEC (2005) ISO 15504-1 information technology - process assessment - part 1 concepts and vocabulary
8. Ardagna D, Ghezzi C, Mirandola R (2008) Rethinking the use of models in software architecture. In: Quality of software architectures. Models and architectures. Springer, Berlin, pp 1–27
9. Schmidt DC (2006) Model-driven engineering. Computer (IEEE Computer Society) 39(2):25
10. Osterweil L (1987) Software processes are software too. In: Proceedings of the 9th international conference on software engineering. IEEE Computer Society Press, Los Alamitos, pp 2–13
11. Henderson-Sellers B, Stallinger F, Lefever B (2002) Bridging the gap from process modelling to process assessment: the oospice process specification for component-based software engineering. In: Euromicro conference, Dortmund, Germany, 2002. Proceedings. 28th. IEEE, pp 324–331
12. ISO/IEC (1995) ISO 12207 Information technology - software lifecycle processes
13. Hamann D (2006) Towards an integrated approach for software process improvement: combining software process assessment and software process modeling. Fraunhofer-IRB-Verlag
14. Lepasaar M, Makinen T (2002) Integrating software process assessment models using a process meta model. In: IEEE international engineering management conference, 2002. IEMC'02, Cambridge, UK, vol 1. IEEE, pp 224–229
15. Makinen T, Varkoi T (2008) Assessment driven process modeling for software process improvement. In: Portland International conference on management of engineering & technology, Portland, PICMET 2008. IEEE, pp 1570–1575
16. IEEE Standards Software Engineering (2004) Vol. 2. Process Standards. IEEE Std. 1012 2004. Standard for Software Verification and Validation Plans

17. ISO/IEC (2006) ISO/IEC 24765 software and systems eng. vocabulary
18. Zhu H, Hall PA, May JH (1997) Software unit test coverage and adequacy. ACM Comput Surv 29(4):366–427
19. OMG (2012). UML (unified modeling language). Last accessed 01-2013. http://www.omg.org/spec/UML/
20. OMG. BPMN, Business process modeling notation, version 2.0. Last accessed 01-2013. http://www.omg.org/spec/BPMN/2.0/
21. García-Borgoñon L, García-García JA, Ortega MA, Escalona MJ (2012) Software process management: a model-based approach. In: Proceedings of the 21st international conference on information systems development (ISD)
22. Escalona MJ, Aragon G (2008) Ndt. A model-driven approach for web requirements. IEEE Trans Softw Eng 34(3):377–390. http://doi.ieeecomputersociety.org/10.1109/TSE.2008.27
23. ITIL (Information Technology Infrastructure Library). Last Accessed 01-2013. http://www.itil-officialsite.com
24. ISO/IEC (2013) ISO/IEC 29119 software engineering – software testing standard. International Organization for Standardization
25. ISO/IEC (2008) ISO 9001:2008 quality management systems - requirements. International Organization for Standardization
26. Project Management Institute (2008) A guide to the project management body of knowledge (PMBOK guide)
27. ISO/IEC (2005) ISO 27001 information technology - security techniques - information security management systems - requirements