Dealing with Deviations on Software Process Enactment : Comparison Framework

Manel Smatti
LSI-USTHB
Algiers, Algeria
msmatti@usthb.dz

Mohamed Ahmed Nacer
LSI-USTHB
Algiers, Algeria
anacer@mail.cerist.dz

Abstract – Software development is a collective and complex task carried out through the cooperation of human agents and automated tools, this interaction defines the software process. PSEE (Process-centered Software Engineering Environment) are environments designed to support the creation and exploitation of software process models that define the expected behaviors of process agents. However, human agents may deviate from the process model; therefore, the PSEE should be flexible enough to cope with these unexpected actions. This paper deals with the problem of deviation during software process enactment; it gives an overview of significant research works that have been proposed to support deviations in the context of software process execution.

Keywords – Software Process (SP), Software Process Enactment, Process-centered Software Engineering Environments (PSEE), Deviation

1. INTRODUCTION

Software development is defined as a collection of procedures accomplished through the cooperation and interaction of human agents and automated tools. Therefore, the quality of the final product depends always on the quality of the software process used to deliver it. However, the complexity of software products and the involvement of human in these processes made them more complex and difficult to manage. Furthermore, software development is a recurrent process; thus, pursuing a defined model in such case has become more than crucial. A software process model is an abstract representation of the software process; it is a description of the process expressed in a suitable Process Modeling Language (PML) [15] whose main objective is to provide required means to enact the process.

Process-Centered Software Engineering Environments (PSEEs) [11] are meant to support the creation and the exploitation of software processes; they are based on the explicit representation of the process and are centered on a PML interpreter that includes mechanisms to enact the process model.

Despite their great support for software development, PSEEs have not acquired an industrial success. This is mainly due to their rigidity and their lack of agility that is known to be inescapable in every software product.

Moreover, software products have become increasingly complex; their development processes are extending over several months or even several years, which lead them *to deviate* from their initial model. Deviations are known to be actions performed by process agents and which are not described or allowed in the process model. As a result of these actions, the quality of software products, delivery time and costs are affected. Finding solutions to cope with such problem has become more than important in order to guide software development.

Several research works have attempted to address this issue by classifying these deviations, proposing mechanisms to detect

them and finding out solutions to cope with this problem. In this context, we will give in this paper an insight about the relevant approaches that have been proposed in this field.

The paper is organized as follows. Section 2 gives an overview of software process enactment domain, dedicated environments, and some related problems. Section 3 deals with the deviation problem during software process enactment by introducing the deviation concept and giving an illustrative example. Section 4 highlights some relevant works proposed to support deviations during software process execution, these approaches are discussed with respect to some criteria we have defined. The paper is concluded in section 5.

2. SOFTWARE PROCESS ENACTMENT

Software Engineering Environments (SEEs) are meant to support software development. Most of them are based on a predefined software process model [11]. Process-centered Software Engineering Environments (PSEEs) give up the notion of a predefined process model, they support variety of processes. A PSEE is basically centered on a Process Modeling Language (PML) [15] and it provides tools to validate process models and enact them.

Using different PMLs, each of these environments is based on a different syntax. Though, they are all designed around the following three parts defined in [6] and taken back by recent approaches:

i. *Process model*: a static description of the process using a PML.
ii. *Actual process*: the process as it is performed in real world.
iii. Observed process: a reflected view of the actual process in the PSEE.

In [12], these different views of a software process are related through a *consistency relationship* that determines the ideal execution of a software process. Based on this consistency relationship, many problems, related to software process enactment, may be defined.

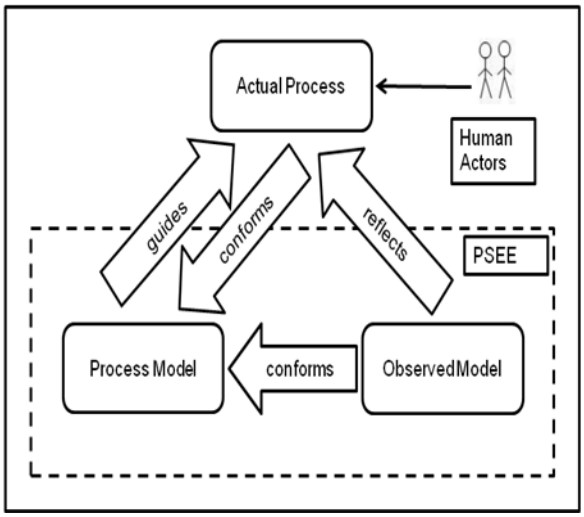

Figure 1: Consistency relationship in Software Process [12]

Despite the large number of prototypes that have been proposed in order to provide an environment that could be adopted by the business community, these attempts are remained and operated in the field of academia. This failure is due, particularly, to the rigidity of PSEEs which tend to provide more details to allow process execution.

Many research works have focused on solving problems related to software process enactment. For instance, the problem of heterogeneous formalisms used to describe software processes, the problem of geographic distribution and the problem of deviations. The present paper focuses on the latter category; we introduce in the following sections the concept of deviation and most important contributions to solve this problem.

3. DEVIATION CONCEPT

3.1. Definition

A deviation is known as a performed action that is not described in the predefined process model or that violates some of the constraints expressed in the process [12]. For example, launching an activity of which preconditions are false, assigning an activity to a person other than the authorized ones or an invalid number of consumed or generated resources ...etc. A more detailed example taken from [1] is given below.

We have noticed that the definition given above can refer to the term inconsistency in some approaches. For instance, in [14] a deviation is defined as an inconsistency that may occur during process enactment. On the other hand, in [12] an inconsistency is known as the state of software process resulting from a process deviation and it is the difference between the actual value of a system variable and the expected value in [13].

3.2. Example of deviation

Lots of approaches have been proposed to support deviations during software process enactment. These approaches differ in their procedures when detecting deviations and in supports they offer to correct the resulting inconsistencies.

As an example, we take back the following simple software process proposed by Da Silva et al. [1] and we explain how authors proceed to detect deviations.

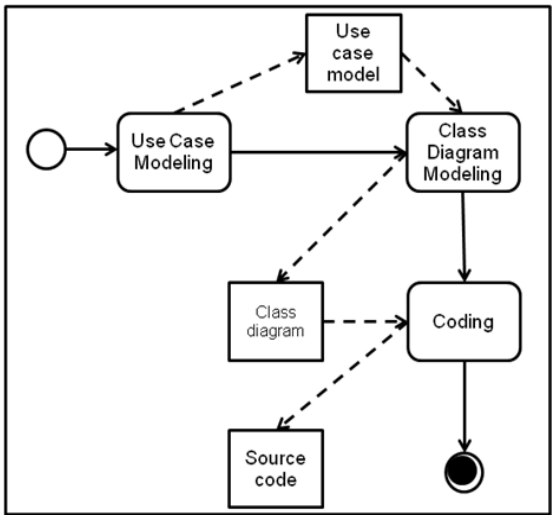

Figure 2: Software Process Example [1].

According to the authors, a deviation occurs when one of these three verifications performed by the PSEE fails. These verifications are performed for each activity as follows:

i. *When it is launched*: the PSEE verifies if the required artifacts for its beginning, according to the process model, are available.
ii. *During its execution*: the PSEE verifies that the execution steps are corresponding to those described in the process model.
iii. *At its end:* the PSEE verifies if the artifacts delivered by the activity are those expected according the process model.

In this contribution, authors have focused on artifacts consumed and delivered by the software process activities to detect deviations. However, experiences have shown that the problem of deviation is not related just to artifacts, but also to agents and resources involved. More details will be discussed bellow.

4. COMPARISON FRAMEWORK

The deviation problem during process enactment is not a recent one. In 90s, many research works have attempted to find out some solutions and propose prototypes to solve this problem.

For instance, The SPADE environment [3] [2], which is based on a process modeling language called SLANG (SPADE LANGuage), is a high-level Petri nets based formalism that uses an O2 object-oriented database for the storage of process data. Although it offers an extended support to process evolution through the reflectivity features of the SLANG, SPADE assumes that humans involved in the development process do not change the way they work unless they change the process model.

SENTINEL [9] is a PSEE prototype based on LATIN activity-based language, it adopts Client-Server architecture and records the relevant events occurred during enactment in a knowledge base. In SENTINEL, a history of the entire process execution is stored and analyzed off-line to discover deviations.

PROSYT [8], which is, especially, conceived to support any kind of distributed business processes, adopts an artifact-based language called PLAN. One of the key features of PROSYT is its ability to modify the level of enforcement adopted and the consistency handling policy at enactment-time. Table 1 summarizes some relevant approaches that have been proposed in the 90s to deal with deviations during software process enactment.

Table 1: Former Approaches dealing with deviations during SP Enactment.

Year	Prototype	Authors	Description
1993	SPADE [3] [2]	Bandinelli et al.	Defined using a fully *reflective* language (SLANG) that is built over a high-level extension of Petri nets. SPADE includes mechanisms to integrate the definition of the process of changing as well as the development process (*meta-process*).
1995	SENTINEL [9]	Cugola et al.	Based on the LATIN language, a *knowledge base* is used to record relevant events occurred during enactment to perform a *pollution analysis*, using a *temporal-logic* based approach, to detect and tolerate some deviations.
1996	Endeavors [4]	Bolcer and Taylor	An open, extensible, *Internet-based* PSEE. It supports an *object-oriented* definition of SP and both distribution of people and artifacts. To support on the fly deviations handling, it allows *dynamic* modification of object fields, methods and behaviors at *runtime*.
1998	APEL [10]	Dami et al.	A framework that aims to support *heterogeneous PSEE* and to support process evolution. Thanks to the process server, each component (PSEE) can change the current process as well as the process model.
1999	PROSYT [8]	Cugola and Ghezzi	Built around the PLAN language, it adopts an *artifact-based* approach and it supports geographical distributed workgroups. It allows process managers to define a *deviation handling* and a *consistency checking* policies.

4.1. Criteria

In the remainder of this section, we will be interested to recent contributions proposed to solve the deviation problem during software process enactment. These approaches are based on former ones. Six solutions are discussed and compared with respect to a set of criteria we have defined, including:

i. *Proposed classification:* to better support deviations, some authors propose to classify them with respect to constraints they are violating or consistency relationships are breaking down, others based their solutions on what has been already proposed. So, in a classification, more than one type is identified.
ii. *Type of deviations covered*: based on the classification adopted, more than one type of deviation can be considered by the proposed approach.
iii. *Support type:* the goal of most solutions, considered in this paper, is not just to detect deviations that may arise during software process enactment, but also to provide a reconciliation mechanism between the process being enacted and the model initially adopted. This reconciliation can be automatic, semi-automatic or ad-hoc.

4.2. Discussions

4.2.1. Classification and considered deviations

Many deviation classifications have been proposed in the literature. In [6], deviations have been classified into: (1) *actual process deviation* which is an action that breaks the consistency relationship between the actual process and the process model, (2) *observed process deviation:* an action performed within the PSEE and that is not reflected in the process model, and (3) *environment deviation* that breaks the consistency relationship between the actual process and the observed process. This classification is taken back by the approach proposed in [12].

In [14], a deviation can be either *environment-level* or *domain-level*. Environment-level deviation refers to an inconsistency between the software performance and the process enactment whereas a domain-level deviation is the violation of a property defined in the performance model. According to the definitions given in this paper, software performance, process enactment and performance model refer to actual process, observed process and process model, respectively. Notice that this classification has been proposed in [7].

A more detailed classification has been proposed by Bendraou et al. in [5]. A deviation may be organizational, behavioral or structural. An organizational deviation occurs when an activity's deadline is not respected or because of a misallocation of roles...etc. Behavioral deviation may be micro or macro one. The micro behavioral one appears when violating methodological guidelines or business constraints while macro one arises when developers change activities' order. Finally, a structural deviation is gotten when inconsistencies are found in a delivered model.

4.2.2. How to detect deviations and when?

Deviations can be detected either on the fly i.e. during process enactment, or at the end of the execution by analyzing data gathered during the process enactment. To do that, PSEE performs a set of comparisons between the process model and the process as it is performed which allows to measure the distance between the actual process state and the expected one.

To treat structural and micro behavioral deviations in the context of multi-viewpoint development processes [5], each modeling action is represented using Praxis that has been extended to represent the viewpoint in which it has been performed. Praxis rules is the rule-based language that is used by the PSEE to detect deviations. To do that, the PSEE compares each rule with the praxis trace captured from the process execution. Praxis rules can be:

i. *Activity post-condition rules:* define structural constraints over a sequence of praxis actions.

ii. *Activity invariant rules:* define behavioral constraints over a sequence of praxis actions.

Deviation rules are associated with the process model as logical formulas in the approach proposed by Da Silva et al. [1]. To detect deviation, the PSEE performs three kinds of verification for each activity: 1) when it is launched, 2) during its execution and 3) when it finishes. The failure of one of these verifications means that the agent is deviating from the process model.

Logical formulas are also used by Kabbaj et al. [12]. Each activity is associated with a set of pre and post conditions and a set of invariants. The transformation of the process model, defined as a UML profile, to logical formulas is obtained automatically using XMI. An action is considered as a deviation if it *is not deductible* from the state that preceded its execution.

In [13], a list of deviation types is integrated into the process model. *Rule-sets* that define pre and post conditions are associated with each activity. Activity conditions are made up of N number of rule sets. For a condition to pass, at least one rule set defined in the condition must pass. Otherwise, a deviation is generated.

An *algebraic* approach based on the *polyadic* $\pi - calculus$ has been proposed in [14] to detect environment-level and domain-level deviations. To not be faced to the complexity of the $\pi - calculus$, a visualization support, TRISO/ML (TRidimensional Integrated Software development model/Modeling Languages) has been used, and a set of rules has been proposed for mapping software processes modeled in TRISO/ML onto $\pi - calculus$ processes. The $\pi - calculus$ model checking allows detecting inconsistencies; also, some properties are used to detect enduring inconsistencies as control flow, data dependency ...etc.

The solution proposed in [16] is based on *software visualization techniques*. It is a set of steps that are performed in parallel with SP enactment. The approach identifies a partial set of nonconformities that are initially used by the PSEE. The approach is *artifact-based*, i.e. to

detect deviations, it does not take into account the activity performed, but just the outputs. So, the approach can only be applied when results are available.

4.2.3. Dealing with deviations

The increasing complexity of software products has made the issue of their creation complex and difficult to manage. Therefore, the goal is no longer to define the problems that may arise during software development but rather to find solutions to address these issues in order to have high quality products.

Most of the solutions mentioned above aimed to propose mechanisms that facilitate the detection of deviations that may occur during software development, but also to find out some solutions to reconcile the process with its initial model, which defines the expected process.

When a deviation is detected, two policies are widely adopted [6]:

i. Change the initial model, so it can support the carried out process.
ii. Tolerate this deviation as much as it does not affect the expected process; i.e. its execution does not have great impact on the process.

Notice that approaches which report the detection of deviations until the end of the process, by analyzing data collected and stored during enactment, or those which do not give any support to correct deviations while enactment, even if they are able to detect them as they occur, do not offer great advantage because solutions they propose are either pieces of advice to prevent future deviations or to change the model. More detailed are given forward.

The approach proposed in [12] applies both solutions mentioned above. Each deviation type, among the thirties identified, is associated with a *tolerance value* interval and a qualification, *minor* or *major*. So, when detected, a deviation is analyzed, with respect to these criteria, and either *tolerated* or *not*. If not, changing the model is adopted.

A *function generator* is used in [1] to define a mapping between the observed process and deviation causes to generate correction plans that are proposed to the process agent. To do that, the PSEE continuously displays the set of detected deviations with their associated risks. So, at each moment, the process agent may ask the PSEE to guide him to fix detected deviations by generating a correction plan.

Table 2: Comparison framework of new approaches dealing with deviations during SP enactment.

Creteria	Approaches					
	Thompson et al. [13]	***Yang et al. [14]***	***Kabbaj et al. [12]***	***Zazworka et al. [16]***	***Almeida Da silva et al. [1]***	***Bendraou et al. [5]***
Deviation Classification	None	**Adopted:** -environment level -domain level	**Adopted:** -actual process -observed process environment	None	None	**Proposed:** -organizational -micro behavioral -structural -macro behavioral
Type of deviations treated	A predefined list of 7 deviations	Domain-level deviations	Observed process deviations	Predefined set of nonconformities	Not specified (3 verifications are scheduled)	Structural & micro behavioral deviations
How to detect deviations and when	Comparison approach /On the fly	model checking /At the end	Deduction system/ On the fly/	Visualization approach/At the end of each step	Deduction system/At the end	a set of 7 rules/On the fly
Type of support	Ad-hoc	Semi-automatic	Semi-automatic	Ad-hoc	automatic	Semi-automatic

No automatic guidelines have been given to correct a deviation when it is logged in [13]. The detection engine compares data collected by the monitoring engine with the properties defined in the process model. If any deviation is logged, an alert appears.

A model checking approach is applied to detect deviations in software processes modeled with the $\pi - calculus$ [14]. Processes are defined as a combination of activities and roles to which a set of rules is applied.

Zazworka et al. [16] use a conformance approach to detect deviations during software process enactment. It is a set of steps that accompany the enacting process. A partial set of nonconformities is defined to estimate the conformance between the enacting process and the expected one. The approach is not applied on activities themselves but to artefacts that result from them.

Table 2 summarizes what has been discussed above.

5. CONCLUSION

Developing high quality software products requires the cooperation of several factors. Although new technologies have brought a lot of facilities and improvement in these development processes, the human factor still plays a major role in their success.

Supporting human actors to achieve the required quality products has led to propose dedicated environments called Process-centered Software Engineering Environments (PSEEs). Having a description of the process model to enact, the PSEE is supposed to guide human agents to achieve the required quality of a software product.

The trend of PSEEs towards modeling software processes details has made them inflexible and rigid. People often need to deviate from the process model to cope with unexpected events that may occur during software process enactment. Thus, providing mechanisms to deal with these deviations has become crucial.

In this paper, we have given an overview of deviation problem during software process enactment through an illustrative example [1]. Some relevant research works have been briefly presented and discussed. Three major criteria have been considered when discussing these approaches: (1) deviation types treated by the approach; (2) how to proceed to detect these deviations and (3) what mechanisms have been adopted to deal with detected deviations.

Almost all solutions have been validated through prototypes with small executed examples. However, software processes are, usually, very complex and extended over several years. So, validating these solutions on real software projects may help integrating these environments into industrial fields.

REFERENCES

[1] M.A. Almeida da Silva, R. Bendraou, J. Robin, and X. Blanc. Flexible deviation handling during software process enactment. In Enterprise Distributed Object Computing Conference Workshops (EDOCW), 2011 15th IEEE International, pages 34–41, Aug 2011.

[2] S. Bandinelli, E. Di Nitto, and A. Fuggetta. Supporting cooperation in the spade-1 environment. IEEE Transactions on Software Engineering, 22(12):841–865, 1996.

[3] S. Bandinelli, C. Ghezzi, A. Fuggetta, and L Lavazza. SPADE: An environment for software process analysis, design, and enactment. In Software Process Modeling and Technology, pages 223–248. Wiley, 1994.

[4] G. A. Bolcer and R. N. Taylor, “Endeavors: A Process System Integration Infrastructure”. In Proceedings of the Fourth International Conference on Software Process (ICSP4), Brighton, UK, December 1996.

[5] R. Bendraou, M. A. Almeida da Silva, M. P. Gervais, and X Blanc. Support for deviation detections in the context of multi-viewpoint-based development processes. In CAiSE Forum, pages 23–31, 2012.

[6] G. Cugola. Tolerating deviations in process support systems via flexible enactment of process models. IEEE Transactions on Software Engineering, 24(11):982–1001, 1998.

[7] G. Cugola, E. Di Nitto, A. Fuggetta, and C. Ghezzi. A framework for formalizing inconsistencies and deviations in human-centered systems. ACM Trans. Softw. Eng. Methodol., 5(3):191–230, July 1996.

[8] G. Cugola and C. Ghezzi. Design and implementation of prosyt: a distributed process support system. In Enabling

Technologies: Infrastructure for Collaborative Enterprises, 1999. (WET ICE '99) Proceedings. IEEE 8th International Workshops on, pages 32–39, 1999.

[9] G. Cugola, E.Di Nitto, C. Ghezzi, and M. Mantione. How to deal with deviations during process model enactment. 17th International Conference on Software Engineering, pages 265–265, 1995.

[10] S. Dami, J. Estubler, and M. Amiour. Apel: A graphical yet executable formalism for process modeling. In E. Nitto and A. Fuggetta, editors, Process Technology, pages 61–96. Springer US, 1998.

[11] V. Gruhn. Process-centered software engineering environments, a brief history and future challenges. Annals of Software Engineering, 14(1-4):363–382, December 2002.

[12] M. Kabbaj, R. Lbath, and B. Coulette. A deviation-tolerant approach to software process evolution. In Ninth international workshop on Principles of software evolution: in conjunction with the 6th ESEC/FSE joint meeting, IWPSE '07, pages 75–78. ACM, July 2007.

[13] S. Thompson, T. Torabi, and P. Joshi. A framework to detect deviations during process enactment. In Computer and Information Science, 2007. ICIS 2007. 6th IEEE/ACIS International Conference on, pages 1066–1073, July 2007.

[14] Q. Yang, M. Li, Q. Wang, G. Yang, J. Zhai, J. Li, L. Hou, and Y. Yang. An algebraic approach for managing inconsistencies in software processes. In Software Process Dynamics and Agility, pages 121–133, 2007.

[15] K. Z. Zamli and P. A. Lee. Taxonomy of process modeling languages. In Computer Systems and Applications, ACS/IEEE International Conference on. 2001, pages 435 – 437, June 25-29 2001.

[16] N. Zazworka, V.R. Basili, and F. Shull. Tool supported detection and judgment of nonconformance in process execution. In Empirical Software Engineering and Measurement, 2009. ESEM 2009. 3rd International Symposium on, pages 312–323, Oct 2009.