A Review of Detecting and Correcting Deviations on Software Processes

Manel Smatti[1], Mourad Oussalah[2] and Mohamed Ahmed Nacer[1]
[1]*LSI, USTHB, BP 32-Bab Ezzouar, Algiers, Algeria*
[2]*LINA, Université de Nantes, CNRS UMR 6241, 2 Rue de la Houssinière BP 92208-44322, Nantes, France*

Keywords: Software Process (SP), Deviation, Software Process Enactment, Detection, Correction.

Abstract: Deviations are known as unexpected situations that could arise during Software Process (SP) enactment. They are the difference between what is expected and what is carried out in real world. Experience has shown that the appearance of such situations is unescapable, especially within large software development projects. Moreover, their occurrence often leads to software development failure if they are not detected and corrected as soon as they appear. This paper presents a literature review of deviation problem on software processes. The most relevant approaches that have been dealing with this issue from the 90s until today are considered within this study. The main goal is to have a clear insight of what has been achieved and what worth to be considered by future works. To achieve this aim, we propose two comparison frameworks that highlight the addressed approaches from two different perspectives, how to detect deviations and how to correct them. As a result of this study, we propose a covering graph for each classification framework which puts in advance the strengths and the weaknesses of each approach.

1 INTRODUCTION

PSEEs (Process-centered Software Engineering Environment) (Matinnejad and Ramsin, 2012) are special environments dedicated to support (large) software development projects that are conducted through a set of steps that define their *processes*. The goal of proposing such environments is to assess software agents through the development steps in order to achieve the desired quality within the final products. Moreover, as they are often endowed of the process model description using a Process Modeling Language (PML) (García-Borgoñon et al., 2014), and in ideal situation a clear view of what is carried out in real world, PSEEs play an important role in process understanding, training, when hiring new engineers, and even organization strategies improvement, they are the core of any software development process (Fuggetta, 2000).

Because of their significant importance, PSEEs have gotten a great interest within the software engineering field. Many prototypes have been proposed to cater for all the needs of software development projects by offering means to model processes and enacting them, but the lack of flexibility, to cope with unforeseen situations, within these environments has led to their failure in being widely adopted within industry.

Deviations on Software Processes (SP) are known as unexpected situations that could arise during software development projects. They are either actions that violate the SP model constraints or those performed out of the control of the PSEE. In both cases, the PSEE becomes unable to support the software development and useless, though.

Many solutions have been proposed to address this issue that is related to the evolution aspect of software processes (Bandinelli et al., 1993) (Bandinelli et al., 1994). Most approaches deal with deviations at two different levels: (1) how to detect them, at a first time and (2) how to cope with them. Several methods have been used to address the first point like logic formulas evaluation (Cugola et al., 1995) (Kabbaj et al., 2008) (Almeida da Silva et al., 2011) and algebraic-based analysis (Yang et al., 2007). On the other hand, correcting the occurred deviations has not been much considered except of some proposals that aim at changing the process model so it could further support the software development as done in (Bandinelli et al., 1993) or the reconciliation approach adopted in (Almeida da Silva et al., 2011).

Through this paper, we aim at offering an original analysis of most approaches that have addressed the problem of deviations on software processes. Our main goal is to have a suitable classification that high-

Smatti M., Oussalah M. and Ahmed Nacer M..
A Review of Detecting and Correcting Deviations on Software Processes.
DOI: 10.5220/0005517500150025
In *Proceedings of the 10th International Conference on Software Engineering and Applications* (ICSOFT-EA-2015), pages 15-25
ISBN: 978-989-758-114-4

lights a set of important features we believe they should be considered when choosing an existing approach or proposing a new one. Such classification would be very useful to have a clear insight on what has been achieved so far and what has been left and worth to be considered by future works. Although we have looked over several studies that have considered the problem of deviations within different kinds of processes, we will focus through this paper on studying the most *representative* approaches that have *explicitly* considered this issue in the context of software processes.

The paper is structured as follows. Section 2 gives a general overview of deviations on software processes, the deviation concept is introduced as well as the motivations that have raised the interest devoted to this research area. In Section 3, we focus on how deviations are detected within the existing approaches. The comparison framework we have elaborated at this aim is introduced after defining each item belonging to the set of criteria we have selected for this purpose. Another comparison framework is given in Section 4. Unlike the first one, this second framework is concerned with solutions proposed to correct deviations. This framework is also built around a set of criteria we define. Section 5 discusses the results achieved within both frameworks. A covering graph is elaborated for each classification framework in order to get a clear insight about what criteria have been covered/ignored by each approach. Section 6 concludes the paper.

2 DEVIATIONS ON SOFTWARE PROCESSES

2.1 Deviation Concept

A deviation is an action that violates the process model constraints. It is the difference between what is expected and what is carried out in real world. In (Kabbaj et al., 2008), a consistency relationship, which describes an ideal SP enactment, has been defined based on the interactions between SP agents and the PSEE. Thus, deviations have been defined as actions that break this consistency relationship.

When a deviation occurs, the PSEE becomes unable to support SP agents through the development steps. The state resulting from such situation is called *inconsistency* (Kabbaj et al., 2008). Such situations are very likely to arise within any software development project. Moreover, experience has shown that the appearance of such situations is not an exception (Almeida Da Silva et al., 2013) because SP agents have often to deviate from the software process model. Thus, deviations are unescapable situations that need to be considered from the beginning of the software development process.

2.2 Motivation

The huge software applications that companies claim nowadays have increased the interest devoted to the software process area. This interest results in many contributions especially in SP modeling and SP enactment fields (Ruiz-Rube et al., 2012). The SP modeling field has reached an advanced level thanks to the wide panoply of PMLs that have been proposed (García-Borgoñon et al., 2014). Furthermore, the proposition of the standard SPEM (sta, 2008) by the OMG has facilitated the integration of such formalisms within industry. Whereas, most PSEEs have failed to gain such success because of multiple reasons, among them:

- PSEEs are built around SP models that are very rigid while SP agents are requiring more and more flexibility to be able to act using their skills and their previous experiences;
- New ways of development, like agile methods, are based on customers' implication within all of the software development steps. Thus, final requirements are often changed or modified during the software development life cycle, which often leads to deviate from the SP model in order to cope with these changes;
- Organizations often change their structures and their strategies within (large-scale) software development. New people are engaged while others left, new tools are integrated and much certifications are required. All these are important factors that need to be considered within software processes and their dedicated supports;
- Competition within software development companies to gain international markets is based on two important factors: time and cost of software development. Thus, having special environments that are able to support SP agents in a controlled fashion has become very challengeable.

3 DETECTING DEVIATIONS

The most important research works that have been led during the last twenty years to deal with deviations on software processes are considered through this paper. These approaches are classified and discussed through a set of criteria we have selected.

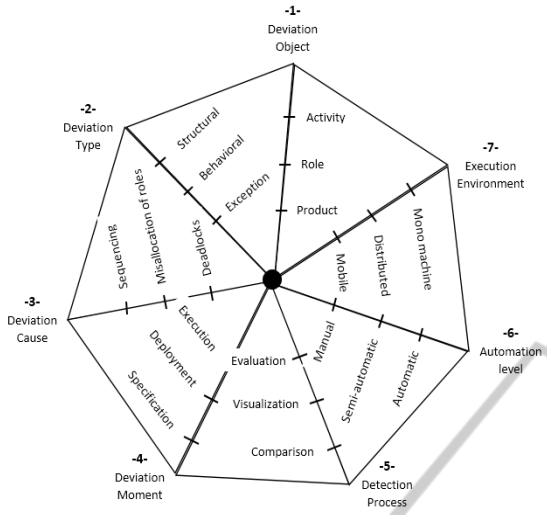

Figure 1: Detection framework criteria.

At a first time, we start by classifying the addressed approaches based on how they proceed to detect the encountered deviations. To achieve this goal, we project each solution on the set of criteria designed for this purpose (Figure 1). We define within the next subsection each criterion, its role and the values it could have.

3.1 Criteria

1. **Deviation Object:** we believe that it is very important to have information about the responsible element for the deviation occurrence. Based on the SPEM conceptual meta-model (sta, 2005), the deviation object could be the *activity*, SP agent through his *role* or the *artefacts* consumed and/or generated within a software process activity. Getting a clear insight about the identity of this element would make the correction much easier.
2. **Deviation Type:** based on their causes/consequences, several kinds of deviations could be distinguished. Even if there is no standard classification, most approaches have been built around a precise set of deviations they define. Through the classification framework we have elaborated (Section 3.2), we will highlight the most relevant deviation classes by enumerating the different types that have been proposed by the existing approaches.
3. **Deviation Cause:** enumerating the possible causes makes the analysis much easier and the solution much effective. Based on the existing approaches, deviations may occur because of different reasons. Depending on the nature/size of the process, the chosen process model and the SP agents skills, a deviation may occur because of:
   - A bad choice of the process model;
   - Misunderstanding of customers needs or misallocation of resources;
   - Wrong execution of activities or a bad sequencing;
   - ...etc.
4. **Deviation Moment:** most approaches define deviations as a problem related to SP enactment and/or SP evolution. In fact, most deviations are detected while the process is enacting but they are not all the reason of an execution problem. Deviations could be *static*, which means that they are related to the SP specification and that the problem has been encountered within the software process model before any deployment. Deviations may also occur during deployment due to a misunderstanding of the SP model constraints or even because of a bad assignment of roles.
5. **Detection Process:** detecting the occurred deviations within software processes falls into finding mechanisms to analyze and evaluate the software process along all its steps. Based on the chosen process model, the PML used to describe it and the tools integrated within the PSEE, several methods have been implemented for this purpose. For instance, visualization-based method and evaluation of first order logic formulas are two options that have been widely adopted by the existing approaches.
6. **Automation Level:** PSEEs are supposed to support SP agents to achieve the desired quality within the final products. The automation level offered by these environments play a key role in the success of any software development project. Thus, the ability of the PSEE to detect incorrect actions or transitions without human intervention is very important. In spite of the great number of prototypes that have been designed to support deviations on software processes, we have noticed that detecting such situations still require human intervention in most cases. As a result, the detection operation is often *semi-automatic* or *manual*.
7. **Execution Environment:** a software process could be enacted either within a *mono machine* environment or a *distributed* one. Moreover, some recent research works are focusing on finding out solutions to easily integrate nomadic users within software development projects. Thus, proposing a support for deviations, or for any other problem encountered within the software engineering field, requires a good estimate of the runtime environ-

ment. It is obvious that the larger is the environment, the more difficult is the solution to implement.

3.2 Classification Framework

Deviations have been considered by researchers from the beginning of the 90s. Since that, several approaches have been proposed to address this problem within the different kinds of processes. The most relevant approaches are listed in Table 1. To select the appropriate papers, our work was partially inspired from the methodology followed in (Ruiz-Rube et al., 2012) when performing a systematic review. The papers related to our study have been selected based on a rigorous search using one of the three words: deviation, inconsistency, evolution conjointly with the software process expression. Searchers were performed on the most known digital libraries such: IEEE Digital Library, Springer and ACM Digital Library. As a second step, papers that have not treated the studied issue as a main subject, but have just given an insight about it, have been discarded.

Moreover, As we are interested to the Software Process area, and to not have a cluttered tables and graphs, we consider through this study only the most representative approaches that have been proposed within this field.

We start within this section by classifying the selected approaches based on how they proceed to detect deviations on software processes. Each solution is projected on the aforementioned set of criteria. The results obtained from this classification are presented in Table 2. Some of the values listed in this table have been *explicitly* extracted from their sources while others constitute the outcome of the analysis we have performed on each of the selected works.

Discussion. As we have mentioned before, the problem of deviations on SP enactment is not recent. SPADE (Bandinelli et al., 1993) and SENTINEL (Cugola et al., 1995) are the most known approaches that have dealt with this issue. Based on the SLANG and LATIN languages respectively, these two prototypes have been the basis of all the approaches that have been proposed later. Although they have reached an advanced level, especially after the proposition of their successors SPADE-1 (Bandinelli et al., 1996) and PROSYT (Cugola and Ghezzi, 1999) respectively, these prototypes have failed in being adopted within industry because of multiple reasons such as the complexity of the languages used to conceive them and the lack of flexibility within SPADE that does not support deviations until the process model is changed. Moreover, the wide panoply of PMLs that have been proposed in addition of the proposition of SPEM has led to consider new features to model software processes and enacting them.

On the other hand, recent approaches take advantage from what has been achieved within the software engineering field and its dedicated tools and languages. For instance, (Kabbaj et al., 2008) propose to support deviations within software processes described as a UML profile. Authors use XMI (XML Metadata Interchange) to automatically generate first order logic formulas from the UML description of the SP model. The analysis performed upon the obtained formulas enable authors to easily detect the occurred deviations.

4 CORRECTING DEVIATIONS

Detecting the unexpected situations that could arise during a given software development project is very important but the most important is to be able to handle them. Among all the approaches considered within this study, few have proposed a *correction* plan to fix the occurred deviations. Through this section, correction mechanisms that have been adopted by the covered approaches are highlighted. First, we start by enumerating some motivations that have incited to propose these correction plans. Then, we provide a brief definition of the possible solutions that have been widely adopted to deal with deviations on software processes. Finally, as done for the detection aspect, a classification framework is elaborated after defining each item belonging to the selected set of criteria.

4.1 Motivation

Offering a suitable development environment that meets all the SP agents requirements, including flexibility, still remains a big challenge within the Software Engineering field.

Deviations are very likely to occur within any software development project. At each moment, SP agents may decide not to follow the software process model because they think they are able to accomplish things better than as described within the SP model. Thus, they decide to act based on their skills or on their previous experiences. Such deviations, even if they are performed with a good faith, could be the main reason of any software project failure. Offering a good support for software processes results in:

- Considering the SP model as a plan that is supposed to guide SP agents to achieve the final goals

Table 1: Approaches dealing with deviations.

	Year		
Proposed Approach	19990-2000	2000-2010	2010-Today
	(Bandinelli et al., 1993)	(Thompson et al., 2007)	(Yong and Zhou, 2010)
	(Cugola et al., 1995)	(Yang et al., 2007)	(Almeida da Silva et al., 2011)
	(Bolcer and Taylor, 1996)	(Egyed et al., 2008)	(Cugola et al., 2011)
	(Dami et al., 1998)	(Kabbaj et al., 2008)	(Ge et al., 2011)
		(Egyed et al., 2008)	(Bendraou et al., 2012)
			(Zhang et al., 2012)
			(Hull et al., 2013)
			(Rangiha and Karakostas, 2014)

Table 2: Classification framework for detecting deviations on SP.

	Deviation object	Deviation type	Deviation causes	Deviation moment	Detection process	Automation level	Execution environment
(Bandinelli et al., 1993)	-Activity	-Static -Dynamic	-Changing in requirements -Changing in the environment/organization	-Execution	Ad-hoc analysis	Manual	Mono machine
(Cugola et al., 1995)	-Role	-Environment level -Domain level	Violation of activities' constraints	Deployment	Evaluation of logic formulas	Semi-automatic	Mono machine
(Thompson et al., 2007)	-Activity -Role	**None**	A predefined set of seven causes	-Deployment -Execution	Evaluation of SQL queries	Semi-automatic	Mono machine
(Kabbaj et al., 2008)	-Activity -Role	-Environment level -Domain level	-Violation of Activities' constraints -Misallocation of roles	Execution	Evaluation of logic formulas	Automatic	Mono machine
(Almeida da Silva et al., 2011)	-Activity	**None**	-Sequencing	Execution	Evaluation of logic formulas	Automatic	Mono machine
(Bendraou et al., 2012)	-Activity -Artefact	-Organizational -Behavioral -Structural	-Activities' sequencing -Violation of methodological guidelines	-Deployment -Execution	Comparing the execution trace to a predefined set of rules	Automatic	Distributed

without imposing a specific manner;

- Offering a suitable tool that brings together all the SP modeling benefits and the required flexibility within a single environment;
- Obtaining a *complete* environment that could be easily integrated into industry.

4.2 Correction Plans

After detecting the occurred deviations, three possible solutions have been distinguished in (Cugola, 1998), these solutions have been adopted by all the approaches that have been proposed later:

1. **Nothing to Do:** in this case, deviations are detected but ignored. The software project is pursued without correcting deviations consequences. The PSEE has an erroneous view of what is carried out in real world and becomes useless, though.
2. **Changing the Model:** the process model is changed to meet the new requirements that have led to the triggered deviation. This solution is very costly and may generate other problems especially if there are several running instances of that model. In addition, a deviation may occur temporarily and does not require changing the entire SP model.
3. **Tolerating the Deviation:** in this case, the SP model is not changed but mechanisms are integrated within the PSEE to *tolerate* deviations under its control. Moreover, the PSEE supports SP agents to reconcile the *observed* process (the observed process is the partial view, of what is carried out in real world, owned by the PSEE (Cugola, 1998) (Kabbaj et al., 2008), the expression *Process definition enactment* has been used in (Cugola et al., 1996) and (Yang et al., 2007) to indicate the same view) with the process model (the planned process in (Zazworka et al., 2009)).

Unlike Section 3 where we have focused on the problem itself. Through this section, we will be dealing with deviations from a solution perspective. To be on the same wavelength, we start within the next subsection by defining each item belonging to the chosen set of criteria (Figure. 2). Approaches that have proposed a correction plan are then discussed according to these criteria.

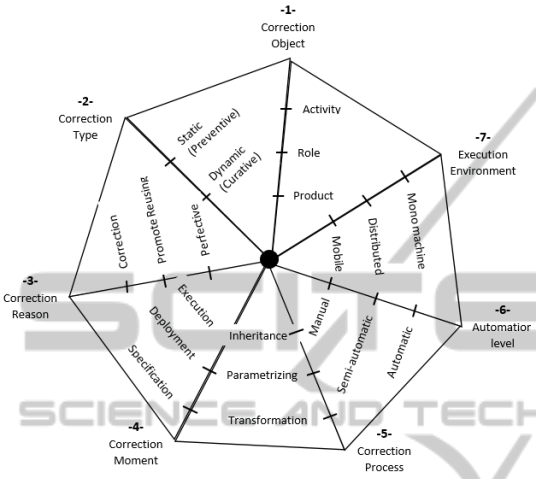

Figure 2: Correction framework criteria.

4.3 Criteria

1. **Correction Object:** A deviation is always caused within one (or more) of the SP elements: *activity*, *role* or *product*. As well, the correction step is applied within one, or more, of these items. It could be the same that has caused the deviation or another one. For instance, the deviation may be detected within an output product while the correction is applied within the activity that has produced it.
2. **Correction Type:** based upon the correction plans we have explained above. SP managers may decide even to change the model to cope with the triggered deviations or to change just the running instance within which the deviation has occurred. Thus, we are talking about *static* and *dynamic* changes, respectively. We consider the first kind of corrections as *preventive correction* while the second as *curative correction*.
3. **Correction Reason:** while interested to deviations on software processes, most researchers have focused their studies on how to detect these deviations and how to correct their consequences. We believe that holding this problem may have more benefits than just correction. For instance, the implementation of a given correction could have a *perfective* effect by improving the process functionalities. Moreover, such correction may improve the reusability of SP models by offering reusable configurations.
4. **Correction Process:** when deviations occur, SP agents/managers may choose multiple options while performing their correction plans. For instance, changing the component (activity, role or product) parameters could be very useful to reconcile the enacting process with the process model. The problem may also be solved by applying a set of transformations or even by changing the component by another one to add new concepts while keeping the old ones (inheritance concept).
5. **Correction Moment:** to correct a given deviation, changes could be applied to the process specification, while deployment or even during the process enactment. Thus, the deviation is not necessarily corrected at the *same* moment where it has been detected. For instance, the deviation may be detected at enactment time while changes are applied to the process specification (process model) to correct it or to avoid it in future time.
6. **Automation Level:** as for the detection process, correcting the occurred deviations may be done *manually* by SP agents. In best cases, the correction is *automatic* if it is entirely done by the PSEE without any human intervention. However, based on what has been achieved by former research works, this is almost impossible. Thus, a suitable correction always involves both the PSEE and SP agents (semi-automatic).
7. **Execution Environment:** through the different approaches that have been proposed in the literature, we have noticed that the solution, when proposed, is not always applicable at the same level within which the process is conducted. Therefore, the process may be led trough a distributed environment while the deviation support is implemented in a *mono machine* environment.

4.4 Classification Framework

The approaches that have proposed a correction policy are considered in this section. As mentioned before, we are interested at those approaches that have *explicitly* proposed a correction plan within the context of software processes. These approaches are classified according to the set of criteria listed above. The results obtained from this classification are summarized in Table 3.

Table 3: Classification framework for correcting deviations on SP.

	Correction object	Correction type	Correction reason	Correction moment	Correction process	Automation level	Execution environment
SPADE (Bandinelli et al., 1993) (Bandinelli et al., 1994)	All SP elements	Static	Perfective	Specification	Transformation	Manual	Mono machine
SENTINEL (Cugola et al., 1995)	All SP elements	-Static -Dynamic	-Perfective -Corrective	- Specification -Execution	Transformation	Semi-automatic	Mono machine
(Almeida da Silva et al., 2011)	All SP elements	Dynamic	Corrective	Execution	Restructuring	Semi-automatic	Mono machine

Table 4: Signification of Covering Graphs values.

Qualification	Detection		Correction	
	Criterion	Description	Criterion	Description
Low	Deviation object	detecting deviations within just one SP element.	Correction object	All the SP elements
Medium		Two SP elements are considered		Two SP elements are considered
High		All the basic SP elements are considered		One SP element is corrected
Low	Deviation type	Behavioral	Correction type	Static
Medium		Functional		Dynamic
High		Behavioral & Functional		Static & Dynamic
Low	Deviation causes	One kind of deviations	Correction reasons	Corrective or Perfective
Medium		A predefined set of possible causes		Corrective & Perfective
High		Causes related to each SP elements are distinguished		Promote reusing
Low	Deviation moment	Execution or deployment	Correction moment	Specification
Medium		Execution and deployment		Deployment
High		Execution, deployment and specification		Execution
Low	Detection process	Ad-hoc	Correction process	Transformation or restructuring
Medium		Using external tools		Inheritance or parametrizing
High		Integrated within the process specification		Design pattern
Low	Automation level	Manual	Automation level	Manual
Medium		Semi-automatic		Semi-automatic
High		Automatic		Automatic
Low	Execution environment	Mono machine	Execution environment	Mono machine
Medium		Distributed		Distributed
High		Mobile		Mobile

Discussion. As we can notice from this classification, in spite of the great number of contributions that have addressed the problem of deviations on software processes, just few of them have been interested at proposing correction mechanisms. The others have been much concerned with studying the problem itself than finding out solutions for it. The results are several classifications and prototypes that aim at detecting those deviations.

Through this section, we have highlighted those approaches that aim at proposing some policies to correct deviations. Approaches like (Zazworka et al., 2009) have been discarded from this classification because they do not propose an *explicit* solution except pieces of advice authors give to help SP managers finding out some resolutions that are supposed to fix the occurred deviations once applied.

Almost all approaches that have proposed correction mechanisms have been validated on simple prototypes within *mono machine* environments. Some authors have focused on studying the problem of deviations within distributed environments as done in (Bendraou et al., 2012) but, as we have already mentioned, they have been much concerned with the problem than the solution.

5 RESULTS AND DISCUSSION

Within the previous sections, we have highlighted the most relevant approaches that have dealt with the problem of deviations on software processes. These approaches have been classified according to two different axes: (1) how they detect deviations and; (2) how they correct them.

As we may decide to choose one method or one tool among the existing ones, and since this choice is strongly based on the constraints (criteria) that are most relevant to us, offering an explicit covering

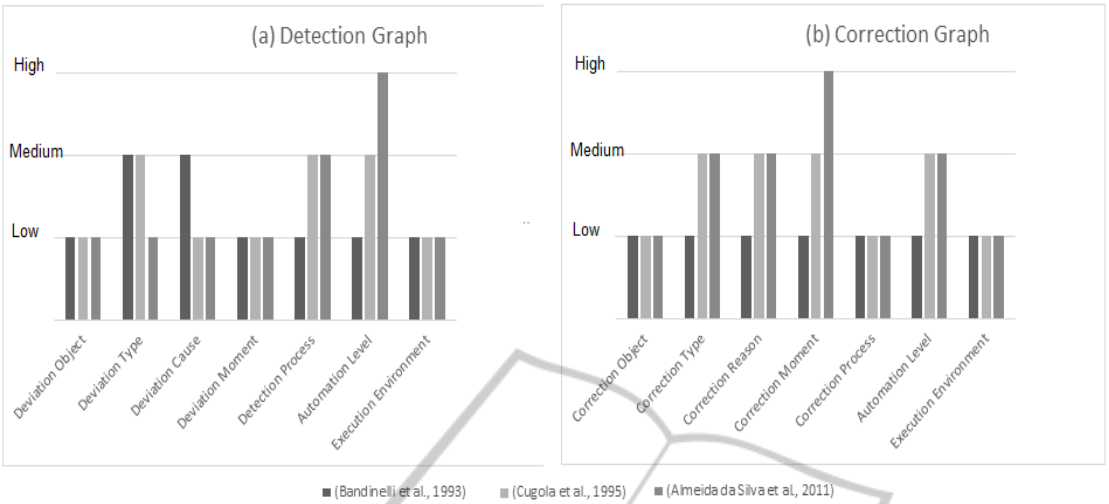

Figure 3: Coverings Graphs for detecting and correcting deviations on SP.

graph for both detection and correction aspects is very important to make the choice much effective.

Through this section, and based on the results obtained within sections 3 and 4, we draw a *covering graph* for each classification framework. For a sake of simplicity, each criterion, in both graphs, will have three possible values: *low*, *medium* and *high* [1]. For instance, for the Deviation Object criterion, if a given approach is able to detect the occurred deviations within just one SP element, the value *low* is assigned to this criterion. If two SP elements are considered within the detection mechanisms, this criterion would have *medium* as value. Otherwise, if all SP elements are covered, the value *high* is assigned to this criterion. On the other hand, when correcting a deviation, we are interested at involving as less components as possible into the correction procedure. This is because the more components are involved, the more expensive the correction is to apply, especially if there are several instances of that SP model. Thus, if a given approach proposes to change/modify just one SP element to correct the occurred deviation, the criterion Correction Object gets *High* as value, if two elements are involved, *Medium* is assigned to this criterion. Otherwise, *Low* is assigned.

We apply the same reasoning for the remaining items of both sets of criteria. Thus, we have noticed that in spite of the great number of classifications proposed in the literature (Deviation type), most of them could be categorized into either behavioral, related to SP agents, or functional, related to functionalities provided by the SP environment. According to what has been achieved within previous works, we assume that detecting behavioral deviations is easier than functional ones because they are related to SP agents behaviors that we are more familiar with. Being able to detect functional deviations comes at a second degree; such deviations could be the reason of a misunderstanding of customers needs or because of a bad execution of activities. In such cases, the PSEE must be endowed of mechanisms that facilitate the analysis of the enacting software process in order to warn SP agents and help them to fix these inconsistencies. Unfortunately, most of existing environments still require human intervention to detect such deviations. Thus, if we are able to treat these two classes of deviations, we would be able, we think, to offer a suitable environment that could be adopted by industrials. Once detected, the triggered deviations must be corrected. Changing the model (static correction) is one alternative that could solve the problem but we argue that this solution is not much efficient since some deviations are casual and do not require to change the entire SP model. Thus, being able to change the enacting SP instance (dynamic correction) without affecting the other instances is very challengeable. Nevertheless, as they have a very long lifecycle, some processes need to change their specification to fit the new requirements of their environment. Therefore, we consider approaches that offer the possibility of dynamic corrections, and static ones when necessary, as the best approaches.

Deviations occur because of multiple reasons (Deviation causes); treating one kind of deviations does not seem to be very useful, especially if we consider the huge size of some software processes. In addition, software processes tend to evolve during time; consequently, limit the solution to a predefined list of

[1]This is a choice of our own, authors may prefer to assign other values for both sets of criteria.

possible deviations does not give any guarantee. The most adequate solution is to be able to define correction patterns for each SP element independently from the other ones. The reason of finding out mechanisms to support these deviations (Correction reason) is to correct the mistakes occurred from not respecting SP models constraints (corrective). Ignoring such mistakes could induce to software project failure. Some SP agents may prefer to ignore these inconsistencies within a given SP instance and apply some changes/modifications in order to prevent them in a next time (Perfective). An ideal solution when supporting deviations is to offer mechanisms to correct them and avoid their further occurrence within not just the software process within which they have been detected but also within a wide spectrum of other kinds of software processes. Thus, we assume that proposing a correction pattern for each SP element is the best solution to promote SP reusing.

Each software process passes by three important phases: Specification, deployment and execution. Detecting deviations during enactment (Detection moment) means that we were unable to perform an earlier analysis that detects the existing anomalies (during specification or while deployment). We believe that the sooner these deviations are detected, the closer to success the software project is. On the other side, correcting deviations should not affect the SP model (SP specification) in best cases. In fact, we are interested at reducing the correction level as much as possible while integrating the different correction patterns into the SP specification.

Deviations could be detected (Detection process) due to an analysis performed by SP agents (ad-hoc) or thanks to special analysis tools (external tools). In best cases, the analysis mechanisms are offered within the SP specification and implemented while instantiating the SP model. On another side, when correcting deviations (Correction process), we can choose to change the SP model even by restructuring the existing elements or by transforming them. In this case, we are required to change all the enacting instances so they could fit the new SP model. Another solution could be to keep the same SP model while changing some parameters or adding some sub-elements in order to facilitate difficult tasks. The best solution remains to integrate correction patterns that could be changed when necessary without affecting the related SP element.

The signification of all the adopted values for each criterion is summarized in Table 4.

To not have cluttered graphs, we will project three approaches on each covering graph: *detection* and *correction*. The results are shown on Figure 3. We have intentionally projected the same three approaches on both graphs. Thus, the reader may have a clear insight on how the same approach proceeds to treat both detection and correction aspects.

As we may decide to choose one method or one tool among the existing ones, and since this choice is strongly based on the constraints (criteria) that are most relevant to us, offering an explicit covering graph for both detection and correction aspects is very important to make the choice much effective.

6 CONCLUSIONS

Through this paper, we have highlighted the problem of deviations on Software Processes. The most relevant approaches that have dealt with this issue have been considered within this study.

The paper starts by introducing the deviation concept. A deviation is an unexpected situation that could arise within a software development process. The appearance of such situation, if not corrected, often leads to the violation of SP model constraints and to the software development failure, consequently. The paper gives also an insight about the relevant reasons that have incited to devote such interest to this research area.

As it is almost impossible to cover, in one study, all what has been achieved within a given research field. We have covered, through this paper, the most important approaches that have *explicitly* considered the deviation issue as a separate problem. Moreover, approaches that have been concerned with this issue within business processes have been discarded.

We have classified the selected approaches using two comparative frameworks. These frameworks deal with: (1) detecting deviations and (2) correcting them. Our choice to make such a distinction while developing this study is justified by the following reasons:

- Holding thi problem on software processes falls into two important directions: finding ways to detect the occurred deviations and offering mechanisms to correct them;
- Separating the classification from these two perspectives makes the analysis much effective. The reader would be easily awards of what has been achieved and what has been left within both directions.

For both classification frameworks, we have conceived a set of criteria. Then, we have detailed every set by defining each item belonging to it. After that, we have projected the selected approaches on each

set. To facilitate the comprehension of each classification, we have listed the obtained results within tables.

As a result, we can notice that besides the great number of approaches proposed to deal with deviations within software processes, just few works have explicitly offered mechanisms to correct them. Moreover, we have realized that most of approaches are *tool-dependent*. For instance, in (Bendraou et al., 2012), authors say *Having different modeling languages would not change anything to the proposed solution because of the use of Praxis*. On the other side, within the Software Engineering field, we are not interested at having the same execution tool. The goal has always been to offer *generic* solutions and let the developers free about the choice of the execution features.

To consider this last issue, we are currently working on proposing a *platform-independent* solution to detect deviations and correcting them at a second time. Our main objective is to offer a *conceptual-level* approach that facilitate the detection of deviations within a large spectrum of software processes. Moreover, the challenge for us is to offer a *reusable* approach since it is one of the main objectives within the Software Engineering field.

REFERENCES

(2005). *Software Process Engineering Metamodel (SPEM) 1.1*. Object Management Group.

(2008). *Software Process Engineering Metamodel (SPEM) 2.0*. Object Management Group.

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

Almeida Da Silva, M. A., Blanc, X., Bendraou, R., and Gervais, M. P. (2013). Experiments on the impact of deviations to process execution. *Ingénierie des systèmes d'information*, 18(3):95–119.

Bandinelli, S., Di Nitto, E., and Fuggetta, A. (1994). Policies and mechanisms to support process evolution in psees. In *Software Process, 1994. 'Applying the Software Process', Proceedings., Third International Conference on the*, pages 9–20.

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

Bandinelli, S., Fuggetta, A., and Ghezzi, C. (1993). Software process model evolution in the spade environment. *Software Engineering, IEEE Transactions on*, 19(12):1128–1144.

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

Bolcer, G. A. and Taylor, R. N. (1996). Endeavors: A process system integration infrastructure. In *Software Process, 1996. Proceedings., Fourth International Conference on the*, pages 76–89. IEEE.

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

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

Cugola, G. and Ghezzi, C. (1999). 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.

Cugola, G., Ghezzi, C., and Pinto, L. (2011). Process programming in the service age: Old problems and new challenges. In Tarr, P. L. and Wolf, A. L., editors, *Engineering of Software*, pages 163–177. Springer Berlin Heidelberg.

Cugola, G., Nitto, E., Ghezzi, C., and Mantione, M. (1995). How to deal with deviations during process model enactment. In *Software Engineering, 1995. ICSE 1995. 17th International Conference on*, pages 265–265.

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

Egyed, A., Letier, E., and Finkelstein, A. (2008). Generating and evaluating choices for fixing inconsistencies in uml design models. In *Automated Software Engineering, 2008. ASE 2008. 23rd IEEE/ACM International Conference on*, pages 99–108.

Fuggetta, A. (2000). Software process: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, pages 25–34. ACM.

García-Borgoñon, L., Barcelona, M., García-García, J., Alba, M., and Escalona, M. (2014). Software process modeling languages: A systematic literature review. *Information and Software Technology*, 56(2):103–116.

Ge, X., Paige, R. F., and McDermid, J. A. (2011). Failures of a business process in enterprise systems. In *ENTERprise Information Systems*, volume 219 of *Communications in Computer and Information Science*, pages 139–146. Springer Berlin Heidelberg.

Hull, R., Su, J., and Vaculin, R. (2013). Data management perspectives on business process management: tutorial overview. In *Proceedings of the 2013 international conference on Management of data*, pages 943–948. ACM.

Kabbaj, M., Lbath, R., and Coulette, B. (2008). A deviation management system for handling software process enactment evolution. In Wang, Q., Pfahl, D., and Raffo,

D. M., editors, *Lecture Notes in Computer Science*, volume 5007, pages 186–197. Springer.

Matinnejad, R. and Ramsin, R. (2012). An analytical review of process-centered software engineering environments. In *Engineering of Computer Based Systems (ECBS), 2012 IEEE 19th International Conference and Workshops on*, pages 64–73.

Rangiha, M. and Karakostas, B. (2014). Process recommendation and role assignment in social business process management. In *Science and Information Conference (SAI), 2014*, pages 810–818.

Ruiz-Rube, I., Dodero, J. M., Palomo-Duarte, M., Ruiz, M., and Gawn, D. (2012). Uses and applications of spem process models. a systematic mapping study. *Journal of Software Maintenance and Evolution: Research and Practice*, 1(32):999–1025.

Thompson, S., Torabi, T., and Joshi, P. (2007). 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.

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

Yong, Y. and Zhou, B. (2010). Software process deviation threshold analysis by system dynamics. In *Information Management and Engineering (ICIME), 2010 The 2nd IEEE International Conference on*, pages 121–125.

Zazworka, N., Basili, V., and Shull, F. (2009). 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.

Zhang, H., Kitchenham, B., and Jeffery, R. (2012). Toward trustworthy software process models: an exploratory study on transformable process modeling. *Journal of Software: Evolution and Process*, 24(7):741–763.