Machine Learning in Predicting the Appropriate Model of Software Process Models Deviation

Tarik Chaghrouchni[1], Issam Mohammed Kabbaj[2], Zohra Bakkoury[3]
[1]AMIPS Research Group
EcoleMohammadiad'Ingénieurs, MOHAMMED V University
Rabat, Morocco
Tarik.chaghrouchni@research.emi.ac.ma

[2]AMIPS Research Group
EcoleMohammadiad'Ingénieurs, MOHAMMED V University
Rabat, Morocco
kabbaj@emi.ac.ma

[3]AMIPS Research Group
EcoleMohammadiad'Ingénieurs, MOHAMMED V University
Rabat, Morocco
bakkoury@emi.ac.ma

JDIM
Journal of Digital Information Management

ABSTRACT: *Software Process Model deviation management allows to supervise process model execution and its adaptation when an inconsisteny is detected. The process model should be adjusted based on specified rules in order to ensure the continuity of the process model execution.*

Some works attempted to handle process model deviation by adjusting fragments and considering some rules to ensure that all constraints are met. Other studies considered only critical constraints in order to avoid cost/ schedule overrun: rules considering the criticality of the activities and their order. However, it is still generating uncertain executions with no mastery on the final result.

However, execution history can help to identify the best choice and improve the control of the model. Machine learning techniques will investigate this execution history and predict the behavior of a future execution.

In this paper, we present an approach using machine learning techniques to combine the Micro-Analyze method "priority and dependency of sub-activities " and data from previous executions to predict the appropriate software process model.

Subject Categories and Descriptors
D.2.3 [Software Coding Tools and Techniques]; F.1.1 Models of Computation

General Terms: Software Process, Machine Learning, Software Models

Keywords: PSEE, Process Model, Deviation, Dynamic Adaptation, Drools, Machine Learning

Received: 24 March 2018, Revised 30 May 2018, Accepted 14 June 2018

DOI: 10.6025/jdim/2018/16/6/308-323

1. Introduction

Software Process Models (SPM) are used to communicate

around the processes and analyze it. They represent the entry point to PSEEs (Process-centered Software Engineering Environments) which use them to coordinate process agents in their tasks. Software development process is a creative activity where consistency is rare, process model is always subject of changes due to each project phases and participants constaints, hence the PSEEs must accept a permanent evolution of process models, tolerate and manage inconsistencies and deviations.

The main role of the PSEEs is to describe the software process models and to give the possibility to monitor their execution: activity, product, Figure 1 presents the PSEE components and interactions.

An observed process deviation is an action performed with in the PSEE that is not reflected in the software process model, it breaks the consistency between the observed process model and the actual process model. The PSEE has then an incorrect view: It will detect and raise a deviation. Figure 2 presents a general overview of the Deviation Management Model under PSEE.

Since the 1990s, the software process community was confronted to managing inconsistencies and deviations of process models enactment and the need to continuously undergo changes and refinements to increase their ability to deal with requirements and expectations.

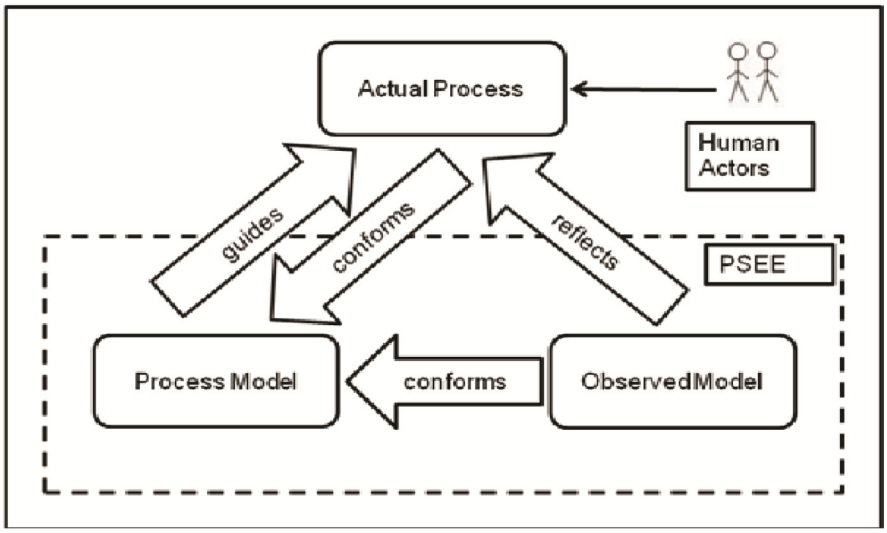


Figure 1. PSEE- Process-centred Software Engineering Environment

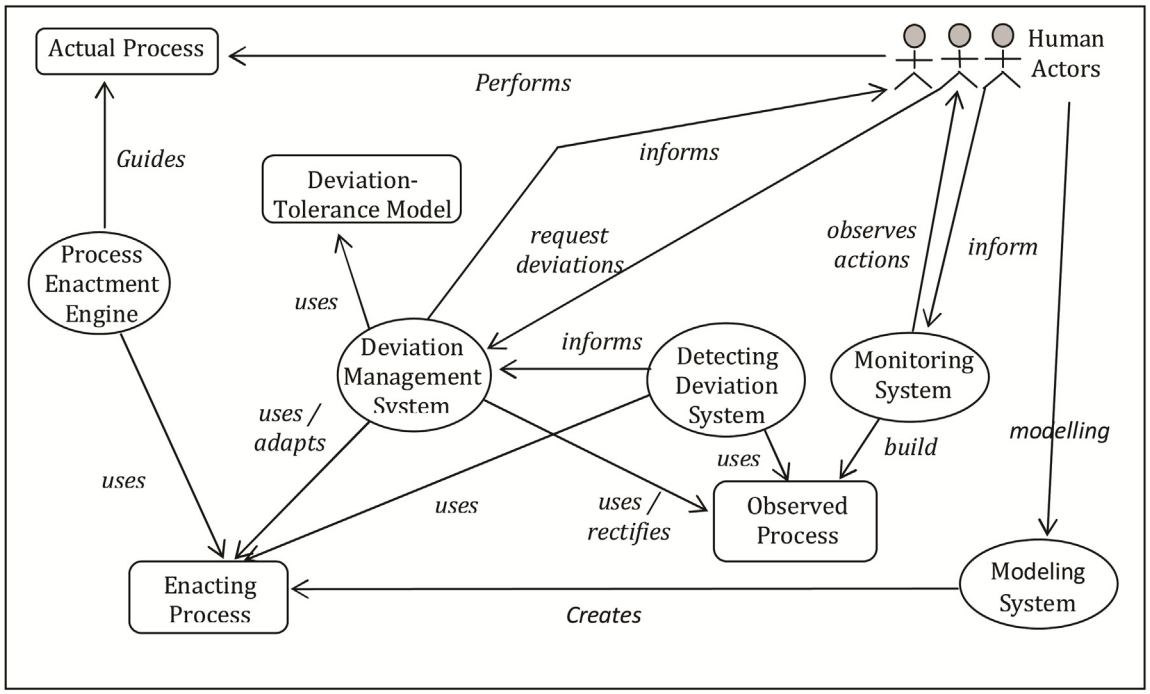


Figure 2. General overview of Deviation Management Model-PSEE

This problem will be partially addressed; There were a number of approaches that allow managing the inconsistencies by adapting the model and providing a large choice of methods to treat the deviation; This created a new challenge which is how to choose the most optimal one.

To address this new challenge and to improve the control of the process model execution, we will present a new approach using machine learning techniques to guide the choice of the optimal process model. Based on data from previous executions, machine learning system will handle the process model change by predicting the adequate process model, in other words, a system that will investigate the execution history and predict the behaviour that will take a future execution based on the various environment values.

This paper is organized as follows. Section 2 presents the related works conducted around software process deviation management. Section 3 deals with concepts of the optimized dynamic adaptation of process model with rules management. Section 4 outlines the machine learning prediction approach we propose, and illustrates it through a simplistic example. Section 5 present two scenarios to materialize the approach. Section 6 concludes the paper and gives some perspectives.

2. Related Works

In software process community, process model deviation problem tends to be enormously complex and hard to monitor, this impacts the methodology and practice of software development, and should be adequately reflected by the theories that capture and support program development process.

SPADE concept [1] treated process deviation problem by providing features for changing models “on-the-fly”. This approach may be effective to cope with major deviations from process models that are expected to occur again in the future, but is unsuitable for situations that require minor or temporary deviations.

Another approach based on SENTINEL [2] tolerates deviations while critical requirements of processes are not violated, but it makes no difference between deviations according to their origin, and has no consistency handling policies. However, practical situations of real-world software development require context-dependent tolerance of deviations.

A deviation management system [3], [3] was conceived to handle software process enactment evolution, it presented more flexibility taking in consideration specific and context-dependent aspects of the process model.

Then in 2011, Almeida [5] [1] treated the problem of process deviation and gave an excellent approach to handle early/ Late deviation detection, risk assessment, correction guidance. The approach was based on sequences of actions from process model to detect and handle the deviation: Executing the set of missing activities in a defined order: Changing the SPM activity flow and keeping the components unchanged.

A new approach [6] where the problem was tackled using dynamic handling of the deviation: Enacting the process by implementing a Fragment library that will replace the missing activities and verify the preconditions, post-conditions, input and output of those activities.

This solution was improved in 2015 [7] by conceiving a new system that identifies the mandatory activities and artefacts and guides smartly the change management: Ignoring the obsolete activities, delaying the optional ones and providing a new activity-based model that handles the deviation.

In 2016, a research [9] proposed the use of the process execution data: Knowledge acquired in previous executions, to support software processes continuous improvement. It defines measures to the software processes performance and the capture of process execution data using data provenance models.

3. Optimized Dynamic Adaptation of Process Model with Rules Management

The optimized dynamic adaptation of process models will adapt the enactment when deviation occurs by minimizing the execution duration and considering the risk. Humans must always remain the masters of decision, and be allowed to decide whether to adopt dynamic adaptation whenever they need to face unexpected situations.

Based on the optimized approach, two attributes are added to the class Work Definition:

1. Mandatory: This attribute will contain two values (True/ False)

If value = True, the elementary activity (Work Definition) must be executed during the enactment of the new fragment.
Else, the elementary activity can be skipped (will become optional).

2. Optional To Execute: This attribute will contain two values (True/False)

If value = True, the Work Definition cannot be ignored, it needs to be executed before the full adapted process model (new fragment) takes end.
Else, it can be ignored once for all;same for previous attribute.

The two added attributes define whether the elementary

fragment must be executed during the enactment of the new process model, and whether it can be ignored, the values are defined based on the knowledge acquired from the previous executions; it can also be defined by the process model Engineer.

A library of fragments is conceived to define the order of the elementary activities during the re-work as per the experience from the past executions considering some constraints: criticality, artefacts...: Each fragment in the library will be constituted of elementary fragments and their classification: their criticality, order in case of deviation and their optionality.

This approach will optimize the new fragment by considering only the mandatory elements as per the order defined in the library to decide the ones that should be executed without stopping the process enactment in order to remediate quickly to the deviation and to validate the constraints causing the violation.

In the case we can have more than one fragment suggested to replace the missing activity, the choice of the re-work fragment will be based on the execution duration of every fragment; this will be calculated based on the average relational duration of each activity to the project according to the previous executions taking in consideration the typology of each project (small, medium, big…)

• Fragment Library

Fragment library is an object that defines the order of the elementary activities during the re-work, and this order will be validated based on criticality and artefacts. Each fragment in the library will be constituted of elementary fragments that will be ranked based on a predefined order.

• Violated Constraints

• Precedence

• Precondition

• Input and In Output of executed Work Definition if they are created/updated by the missing WorkDefinition

• Output and In Output of missing Work Definition even they are not manipulated in the executed Work Definition.

• Mandatory Attribute

This attribute defines if the sub-activity needs to be re-executed if missed or not. We take in consideration if sub activities are consuming the input of the parent activity; we need then to allocate the value True.

The sub activity which is producing the output used by the following activity should be as well mandatory.

• Process Fragment Conception

The fragment to be considered should verify all the conditions that would be realized if the missing activity was executed as expected in the represented model. We consider the execution duration of the elementary fragment under each activity and the mandatory attribute.

The elementary fragments that are mandatory should be executed with the established order and saved in the Fragment library. The new fragment that will replace the missing activity will ensure that its precedence condition will be substituted by the precedence of the missing activity.

• Logical Formalization of Process Models

We will consider that

- *Fragment i = Work Definition i = Wi*
- *Fragment i.j= SubWork i.j = SW i.j*

We consider that deviation occurs since *WorkDefinition i* was not executed.

To formalize this representation, we will consider the structure "*M*" which is the union of the set of formulas that describe structure and behaviour of an enacting process model, According to the conditions that the new process fragment will verify, *M* will verify:

$$goal\ (Work\ Definition\ i) \rightarrow goal\ (New\ Work\ Definition)$$

$$deviation\ (launch, Work\ Definition\ (i{+}1)) \wedge (precedence\ (Work\ Definition\ i) \vee validated\ (New\ Work\ Definition\ i)) \rightarrow (precedence\ (Work\ Definition\ (i{+}1))$$

$$output\ (output\ (Work\ Definition\ i), New\ Work\ Definition)$$

$$inOutput\ (inOutput\ (Work\ Definition\ i), New\ Work\ Definition)$$

$$input\ (Output\ (Work\ Definition\ (i{+}1)), New\ Work\ Definition)$$

$$input\ (inOutput\ (Work\ Definition\ (i{+}1)), New\ Work\ Definition)$$

$$deviation\ (launch, Work\ Definition\ (i{+}1)) \wedge (validated\ (New\ Work\ Definition\ i) \rightarrow inOutput\ ((output\ (Work\ Definition\ (i{+}1)), Work\ Definition(i{+}1))$$

An artefact which is Input/Output of a Work Definition which is a composition of elementary sub-works is as well an artefact Input/Output of a sub-work.

$$Wi.\ \text{sub Works -> not Empty ()}\ \text{implies}$$

$$Wi.\ \text{output -> forAll (p | Wi.subWorks -> select (}SW \mid SW.\ \text{output -> exists(p)) -> size() = 1)}$$

$$Wi = SWj\ v \ldots v\ SW(j+m)$$

WNi is the Work Definition that will replace *Wi.*

WNOi represents the new approach fragment that will be considered in the process model taking in consideration the mandatory values, optionality and the order established in the Fragment library. The most optimized fragment will replace better the missing activity with better quality and less risk.

Processelements
WorkProduct("Wi") for i from 1 to n
WorkDefinition("SWi"); for i from 1 to m
Relationship betweenprocesselements
input("Wi", "Artfacti");
output("Wi", "Artfacti");
input("WNi", "Artfacti");
output("WNi", "Artfacti");
Preconditions, post-conditions and precedences
validated("Wi") -> precondition("W(i+1)") ;
validated("Artfact(i)") -> precondition("W(i+1) ") ;
precedence("Wi+1") ;
finished("Wi") -> precedence("W(i+1)") ;

WNi = SWi1 v SWi2 v ...v SWin

implies

WNOi = Union (*SWi where SWi. mandatory = True*)

WNO will be the optimized fragment

implies

WNO. duration = min (*WNOi. duration*)

The *SWi* will contain the value True in optional To Execute, it will be stored as per defined order and be executed once the activity subject of deviation is completed.

For example, activatable, enactable and enacting are the states of a Work Definition which are represented by activatable (*Wi*), enactable (*Wi*) and enacting (*Wi*).

The Work Definition *Wi* will produce artefact which will have the state defined (artefact, *Wi*).

3.1 Graphical Implementation of the Optimized Approach

To implement a graphical modeling workbench to visualize the EMF models, manage activities state change, intercept events and identify the rule to apply: Put in action the rules management, we used the EMF (Eclipse Modeling Framework). Figure 3 show the class diagram implemented.

To enable the specification of graphical modeling workbench with validation rules and actions using declarative descriptions, we used Sirius which is an Eclipse project which allows creating easily graphical

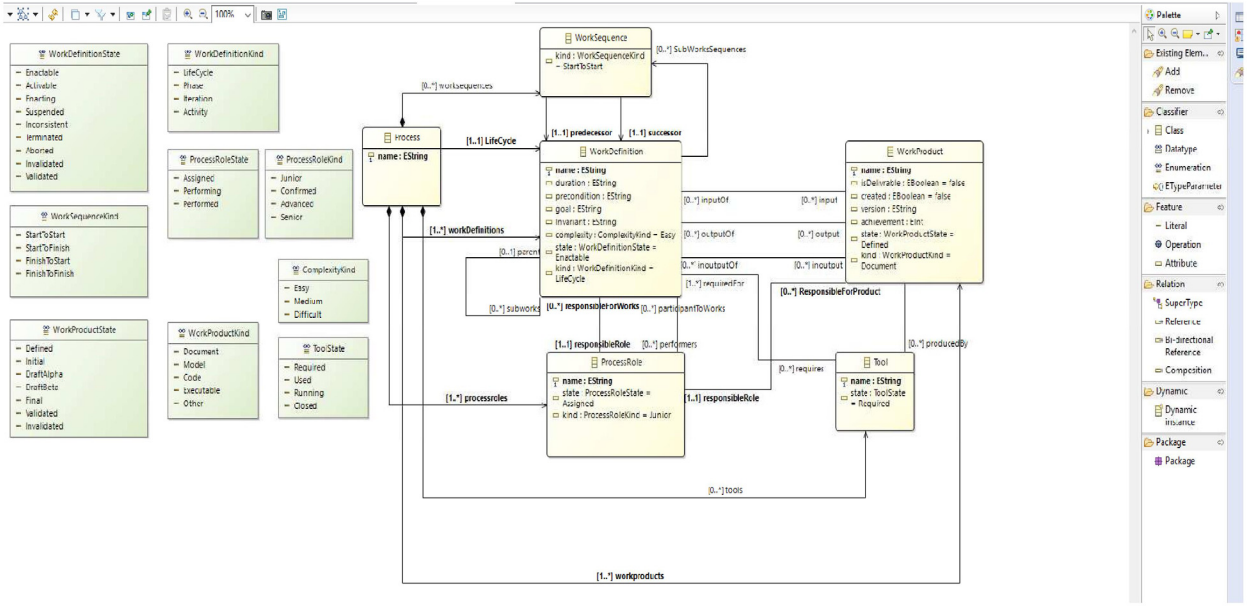


Figure 3. Class diagram of optimized approach

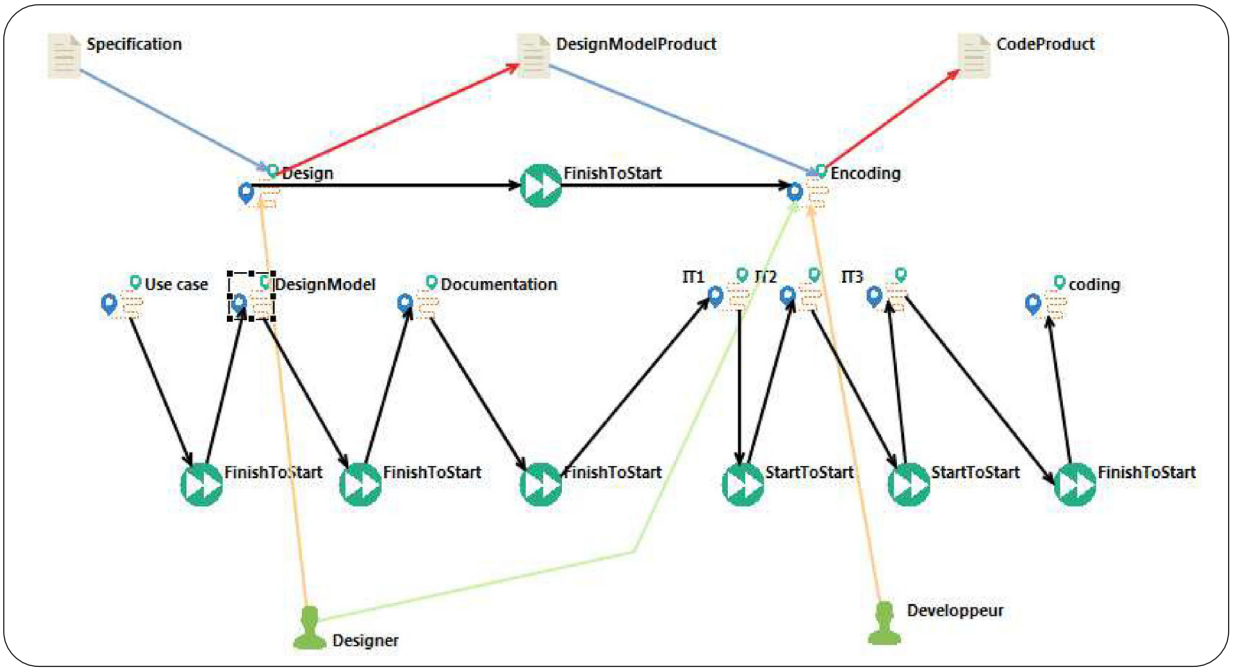


Figure 4. Simplistic Representation of the approach using Eclipse-Sirius

modeling workbench by leveraging the Eclipse Modeling technologies. Figure 4 shows the representation of our model under Sirius.

3.2 Rules Management

To implement the rules management system: A tool allowing managing the rules and constraints, to set rules and apply them to data; we went for the most widely used rules management systems: Drools. It extends and implements the matching algorithm “Rete Pattern”. In simple terms, Drools is a collection of tools that allow to separate and reason on business logic and data found in business processes. The two important keywords we need to notice are logic and data.

This system allows to handle the rules, manages the input and precedence conditions as well as the activities states and performers roles. To illustrate the use of this file we define the following three rules:

• **Input Rule:** If the state of a work Definitions is enacting and the input is not valid, then the state becomes Activable.

• **Precedence Rule:** It’s between two work definition; if the relation of Work Sequence between them is “Finish

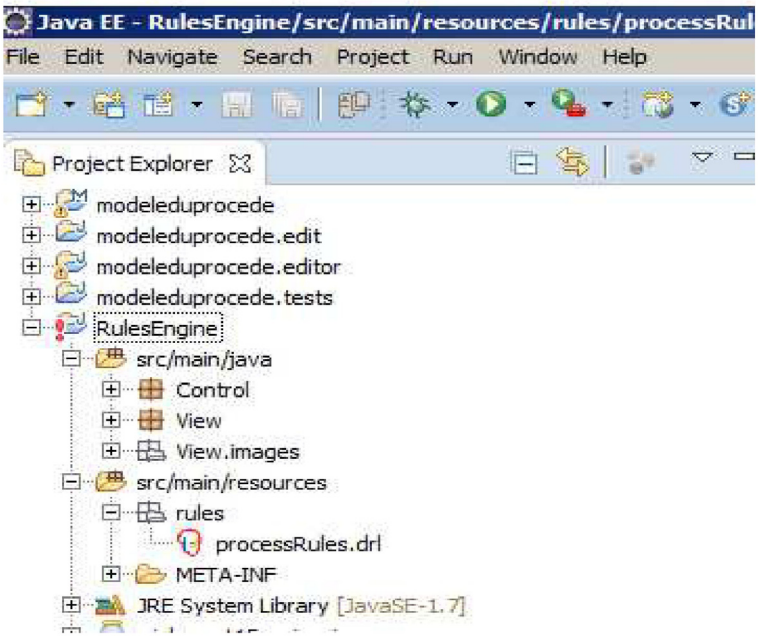


Figure 5. View under Eclipse of RulesEngine project

To Start" and the state of the second is enacting and its predecessor has a state different than Terminated then a deviation is detected. The rule engine will suspend this enacting one and enact the predecessor if it hasn't a subwork. But if it gets a subwork we check if the mandatory is true then we enact the subworks, else we see if it's optional to execute, if it's true then we leave it else we execute it later.

• **Launcher Rule:** If the state of a work definition is enacting and the launcher is one of the performers then the deviation is tolerated. Otherwise the deviation will be rejected and the state will be Activable.

The Process Model enactment will be controlled using Drools. As Sirius project produces XML file which will be the link between the Design project and Drools project. The best way is to use MVC architecture: create a new drools project and name it Rules Engine, by default there are 2 directories; src/main/java and src/main/resources. The first one contains all java classes and the second contains all .drl files (rules), Figure 5 shows the view under Eclipse of the Rules Engine project with its components:

The rules engine which is a component of rules management system will execute a set of defined rules, monitor and maintains the variety and complexity of decision logic that is used by operational systems.

4. Machine Learning in Prediction of the Adequate Process Model

The process engineer after a number of executions will be uncertain of parameters choice making the execution of the adapted process model successful.

Taking in consideration that each execution is unique, the fragment of process model that will be adapted will take in consideration the project characteristics, such as project size, project team...

The same fragment will succeed the process model execution in some cases and fail in others based on the mentioned characteristics.

The process engineer should not choose the fragment based only on the recent history, he should take in consideration all parameters that can guide the choice, since he can have at his disposal the history of various executions and especially the different adapted process models with different configurations. This history will help to choose the fragment of the process based on all parameters and ensure the best execution. In other terms to predict the appropriate fragment that will deal with deviation.

One of the techniques that address this issueis machine learning. It uses advanced statistical models to analyze past instances and to provide the predictive engine in many application spaces. These models are trained on the sample data provided, which should include a variety of classes and relevant data believed to affect the classification: Prediction of executions causing the most relevant deviations as well as deviations that are not acceptable.

In other words, it will predict the best possible process model executions as well as the ones having a significant delay or lack of important features by taking into account some parameters being identified before the start of the SPM execution: composition of the team, project size...

4.1 Machine Learning

In machine learning, asking the right question and knowing the correct answer is the most important. The most important part of the process is to know what question to ask. After that, we should ask whether we have enough and correct data to answer that question.

When a process model is being launched, we need to define how we can predict its execution. The case is that a large piece of predictive data is based on which team will execute each activity, the size of the project, constraints on the budget of the project...

Firstly, we identify, choose and get the data that we want to work with: Define what data or data model that we get from the process is predictive. There comes the requirement for pre-processing the data, so that the process could understand these data, and the good thing is that the machine learning products usually provide some data pre-processing modules to process the raw or unstructured data.

Secondly, we get a candidate copy of data which could be processes through the machine learning techniques to get the actual golden copy.

After the data is pre-processed, we get good structured and shaped data. This data will be the input of machine learning; process has to be iterative until the aimed data is available. In machine learning the major chunk of time is spent in this process. That is, working on data to make it structured, clean, ready and available to apply the algorithm. The result of the algorithm is a model which is considered as the first most appropriate model that we get, but still it needs to be massaged: since this is an iterative process, we do not actually know what the best candidate model is, until we again and again produce several candidate models through the iterative process. We do it until we get the model that is good enough to be considered as appropriate.

Hence, theseven steps of machine learning are:

• **Gathering Data:** This step is very important because the quality and quantity of data gathered will directly determine how good the predictive model can be.

• **Data Preparation**: We'll first put all our data together,

and then randomize the ordering. the order of data should not affect what we learn.

• **Choosing a Model:** The next step in our workflow is choosing a model. There are many models that researchers and data scientists have created over the years. Some are very well suited for image data, others for sequences (like text, or music), some for numerical data, others for text-based data.

• **Training:** Using data to incrementally improve our model's ability to predict the result.

• **Evaluation:** This is where that dataset that we set aside earlier comes into play. Evaluation allows us to test our model against data that has never been used for training. This metric allows us to see how the model might perform against data that it has not yet seen.

• **Parameter Tuning:** It is possible to improve the training once evaluation is done by tuning the parameters. There were a few parameters we implicitly assumed when we did our training, and at this step, is a good time to go back and test those assumptions and try other values.

• **Prediction:** Machine learning is using data to answer questions. So Prediction, or inference, is the step where the value of machine learning is realized.

4.2 Machine Learning: Scenarios and Case Study

The architecture of the approach to implement will be composed of:

• **Module of Events Management:** Activity state change and treatment launching, it will provide the event to the rules management system once a deviation is detected.

• **Rules Management System:** This module will intercept the event and identify the rule to apply based on the input.

• **Execution Engine:** It intercepts the rule to be applicable which is identified by the rule management system.

• **Predictive System:** Which gathers the data based on previous executions and predicts the appropriate process model.

The Monitoring system will control the observed model, once a deviation is detected (Detecting Deviation System), it will be treated by the Deviation Management System based on the tolerance model, the Rules Management system will identify the rule to apply, predictive system will use Machine learning technique to identify the appropriate process model and communicate it to the execution engine: Process Enacting Engine.

Figure 6 represents the architecture of the system.

We will present two scenarios of machine learning; the flow of data (dataset) consists of variables to describe the observed SPM which should be saved by the Process Engineer into CSV files.

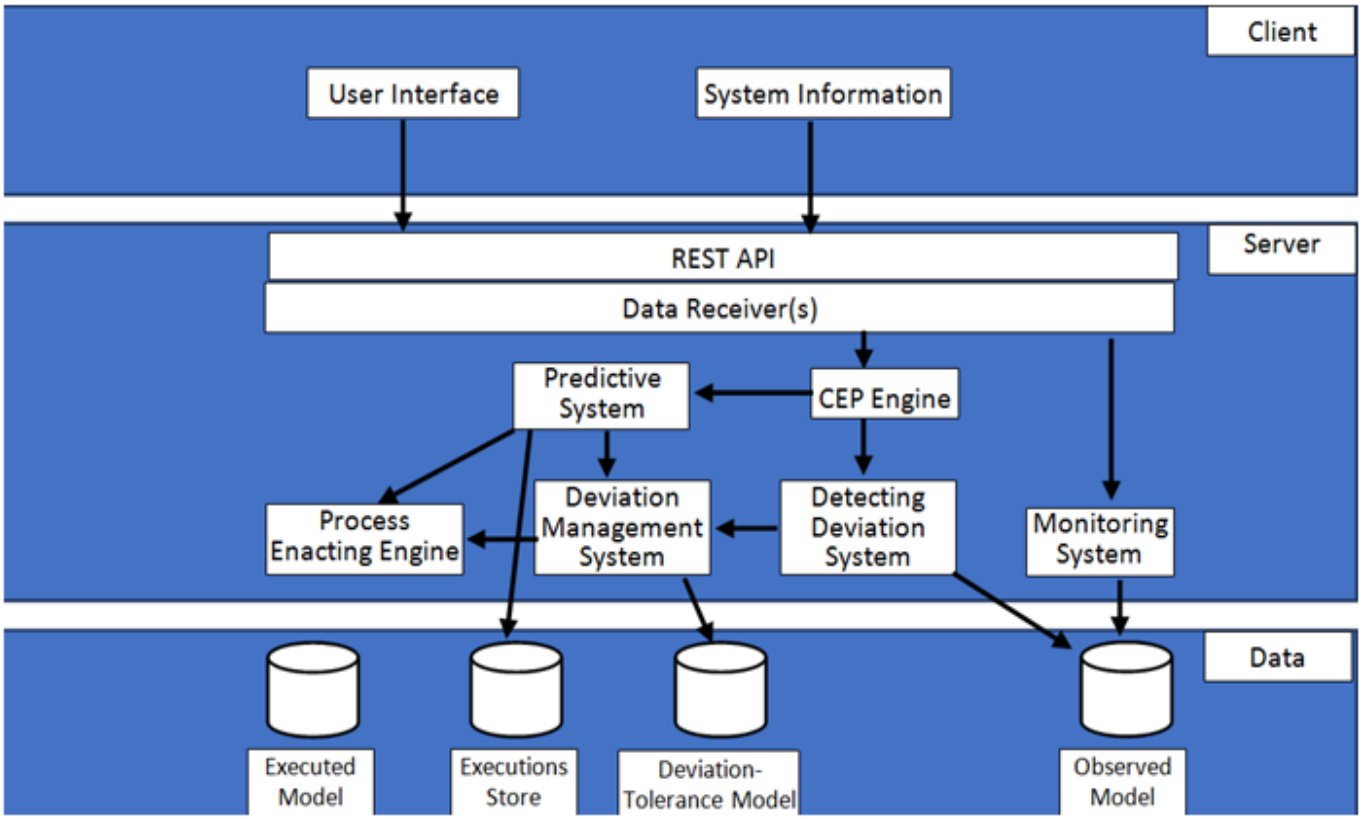


Figure 6. Architecture of the implemented system

The technology we used to materialize the approach is Wekaý [8] which is a collection of machine learning algorithms for data mining tasks. Weka contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization.

Classification methods address these class prediction problems. We will use the RandomTree classifier in our paper. To process the SPM dataset, we will automatically assign the last column of an ARFF file as the class variable; this dataset will store the updated process model in the last column.

• Building the Classifier: Classify SPM

The classifySPM() method begins by creating a list of possible classification outcomes. In the case of the SPM dataset, this is a list of SPM included in the original dataset (list of SPM output of optimized approach). These are each added to a FastVector object by using the FastVector'saddElement() method. Then, once the potential outcomes are stored in a FastVector object, this list is converted into a nominal variable by creating a new Attribute object with an attribute name of SPM and the FastVector of potential values as the arguments to the Attribute constructor. The FastVector must contain the outcomes list in the same order they were presented in the training set. From the ARFF file storing the initial SPMattributes, these are:

@attribute SPM {SPM0,'SPM1','SPM2'...'SPMN'}

And in Java, the potential SPM values are loaded in the same order:

dataClasses.addElement("SPM0");

dataClasses.addElement("SPM1");

dataClasses.addElement("SPM..")..

After the SPM classes are prepared, the classifySPM() method will loop over the Dictionary object and perform two tasks with each iteration:

It will assemble a collection of keys, which are aggregated into a second FastVector object, using the FastVector's addElement() method. That FastVector object contains all the feature names to be supplied to the Classifier object.

It will get the value associated with each key. The values are floating-point numbers stored as strings, so they must be converted to a floating-point type, double in this case. An array of doubles holds each value as it is returned from the Dictionary object.

The array needs to hold the number of elements in the Dictionary object, plus one that will eventually hold the calculated class. Similarly, after the loop executes, the SPM Attribute, created at the start of the function, is added as the final element of the attributes FastVector.

The next step is to create the final object the classifier will operate on. The process begins with creating the Instances object. The first argument to the constructor is the name of the relationship. For a data instance to be classified, it is arbitrary. The second argument to the constructor is the FastVector containing the attributes list. The final argument is the capacity of the dataset. After the Instances object is created, the setClass() method adds the SPM object as a new attribute that will contain the class of the instances.

With the classifier and Instance, the classification process is provided by two classification methods of the Classifier object.

• ClassifyInstance() returns a double representing the class of an object, either true or false, numerically.

• DistributionForInstance(), which returns an array of doubles, representing the likelihood of the instance being a member of each class in a multi-class classifier.

We use distribution ForInstance(), which is called on the instance within the Instances object at index 0. With the distribution stored in a new double array, the classification is selected by finding the distribution with the highest value and determining what SPM that represents, returned as a String object.

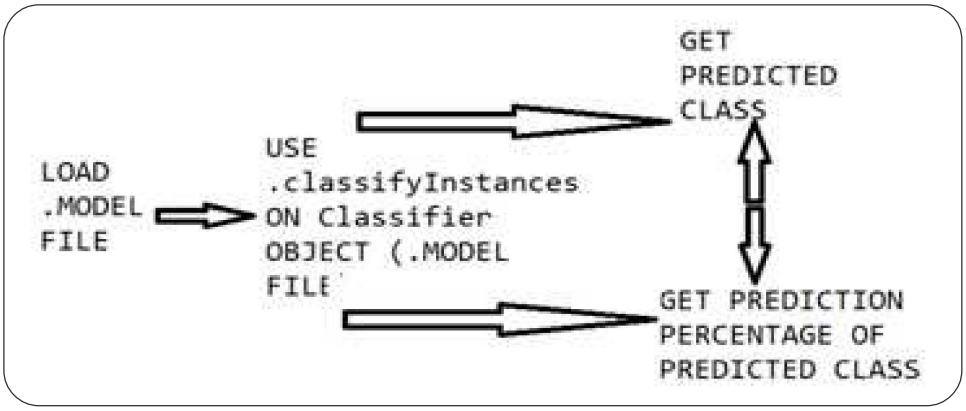


Figure 7. Representation of the classifier

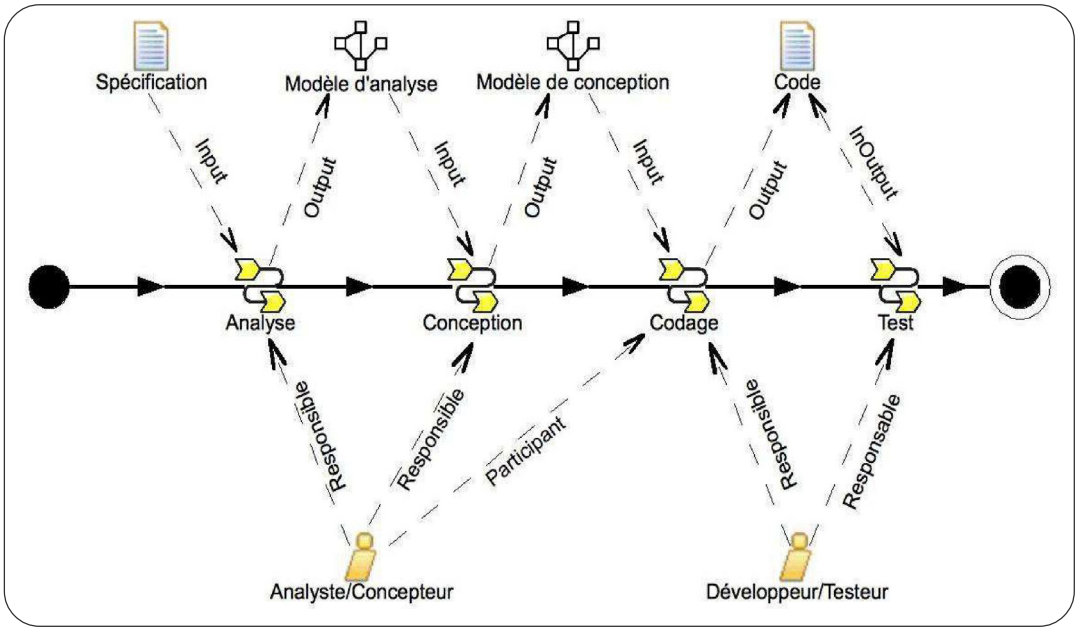


Figure 8. Software Process Model: StudentMini Project

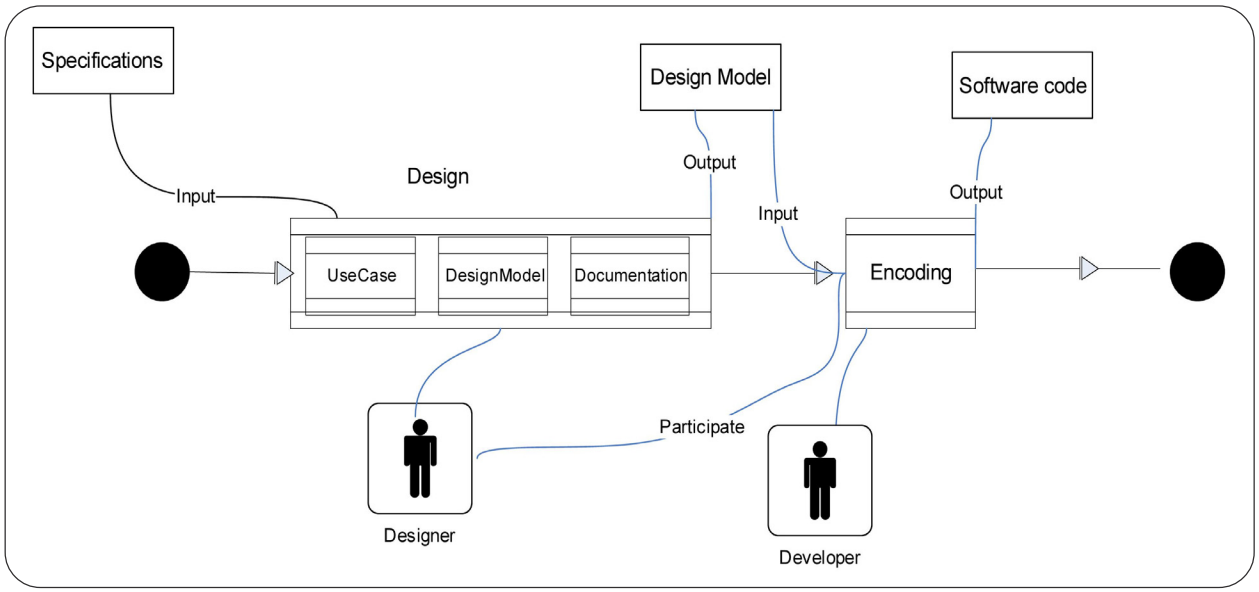


Figure 9. Representation of Student Mini Project in activities

• First scenario: Using Learning Machine to identify the most accurate SPM

To illustrate the first case study, we consider thestudent mini project presented in Figure 8 which is a basic model for software development process:

The representation of this basic model in term of activity will be as Figure 9taking in consideration that Encoding activity will be done over three iterations:

The Logical Formalization of Process Models:

An artefact which is Input/Output of a Work Definition which is a composition of subWorks is as well an artefact Input/Output of a subWork.

Self.subWorks ->notEmpty()

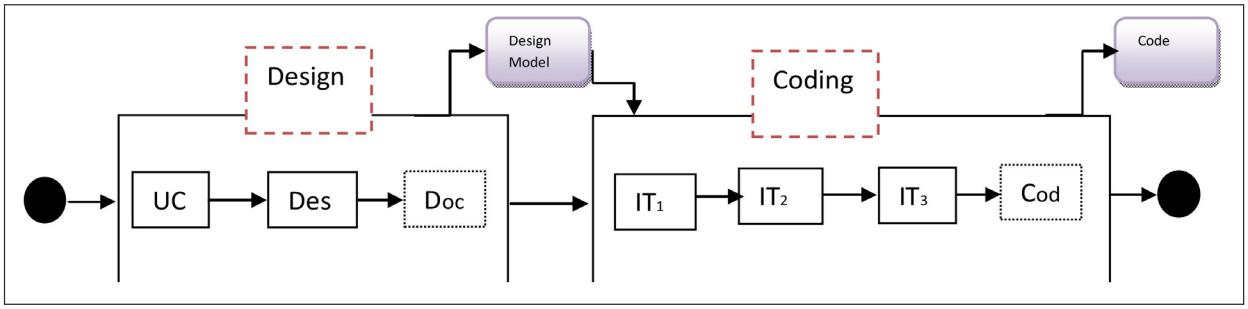
implies

self.output ->forAll(p | self.subWorks -> select(sw | sw.output -> exists(p)) -> size() = 1)

An Elementary subWork will be priority one if correlation_Index value is the highest in the set of subWorks:

Work Definition = SubWD1 v SubWD2 v SubWD3

The illustration of the SPM modeling will be as below:



The logical representation of the model will be:

Input (*Design, "Specifications"*)

Input (*Encoding, "Design Model"*)

Output (*Design, "Design Model"*)

Design.Elementary = False

DesignModel.OptionalToExecute = False

Design.subWorks = {UseCase, DesignModel, Documentation}

UseCase.mandatory = True

DesignModel.mandatory= True

Documentation.mandatory = False

Encoding.Elementary= False

Encofing.OptionalToExecute = False

Encofing.subWorks = {IT1, IT2, IT3}

Where $\{IT1, IT2, IT3\}$ represents the set of sub-activity encoding: three iterations of encoding phases.

Correlation_Index will have the values:

0.3 forUseCase: Can adjust the output of Design activity but it is not blocking the Activity artefact: Design Model.

0.6 forDesignModel: Input of the Coding activity(Output of the Design Activity)

0.1 forDocumentation: Elementary activity that is producing documentation

IT1, IT2 and IT3 will have respectively the values: 0.3, 0.3 and 0.4

In the real work, the deviation is detected in most caseswhen the encoding precedes the completion of the Design activity.

Some other deviations can occur and each one will be treated using the optimized approach and will generate a different modified process model.

Considering the deviation detected when Encoding precedes the completion of Design activity, optimized approach will implement a new fragment: ReDesign that will replace Design Activity and ensures the below constraints:

$goal\ (Design) \rightarrow goal(\ ReDesign)$

$deviation\ (\text{"launch"},\ Design) \wedge (precedence(Design) \vee validated\ (ReDesign)) \rightarrow precedence\ (ReDesign)$

$output\ (output\ (Design),\ ReDesign)$

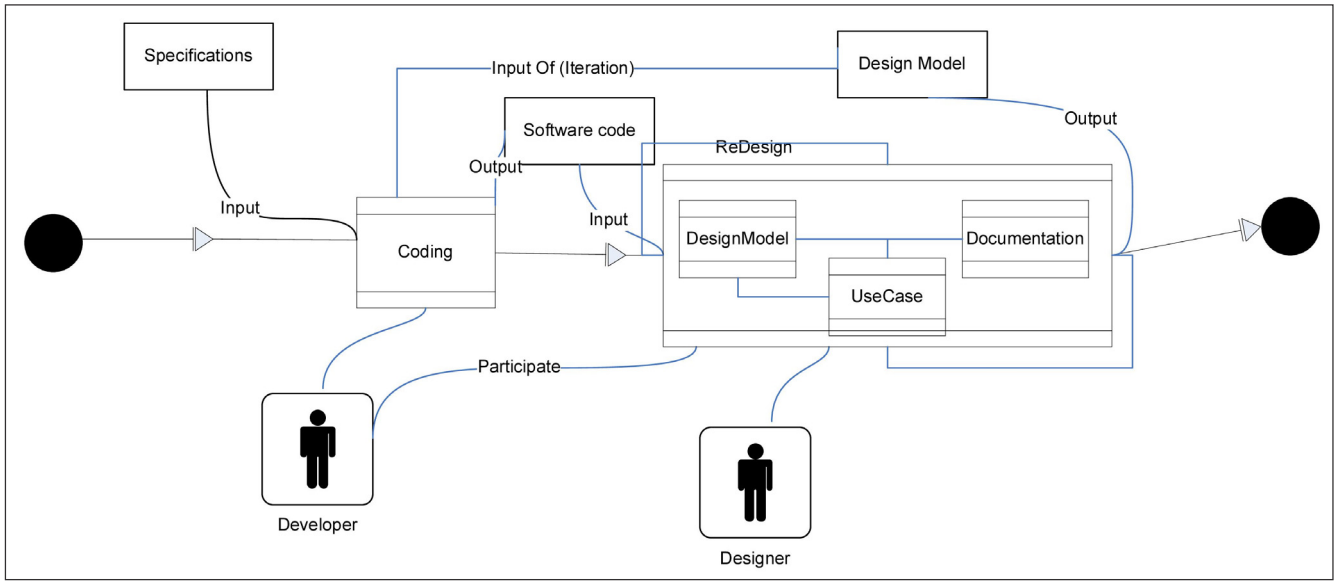


Figure 10. Process model deviation handling

inOutput (inOutput (Design), ReDesign)

input (Output (IT1), ReDesign)

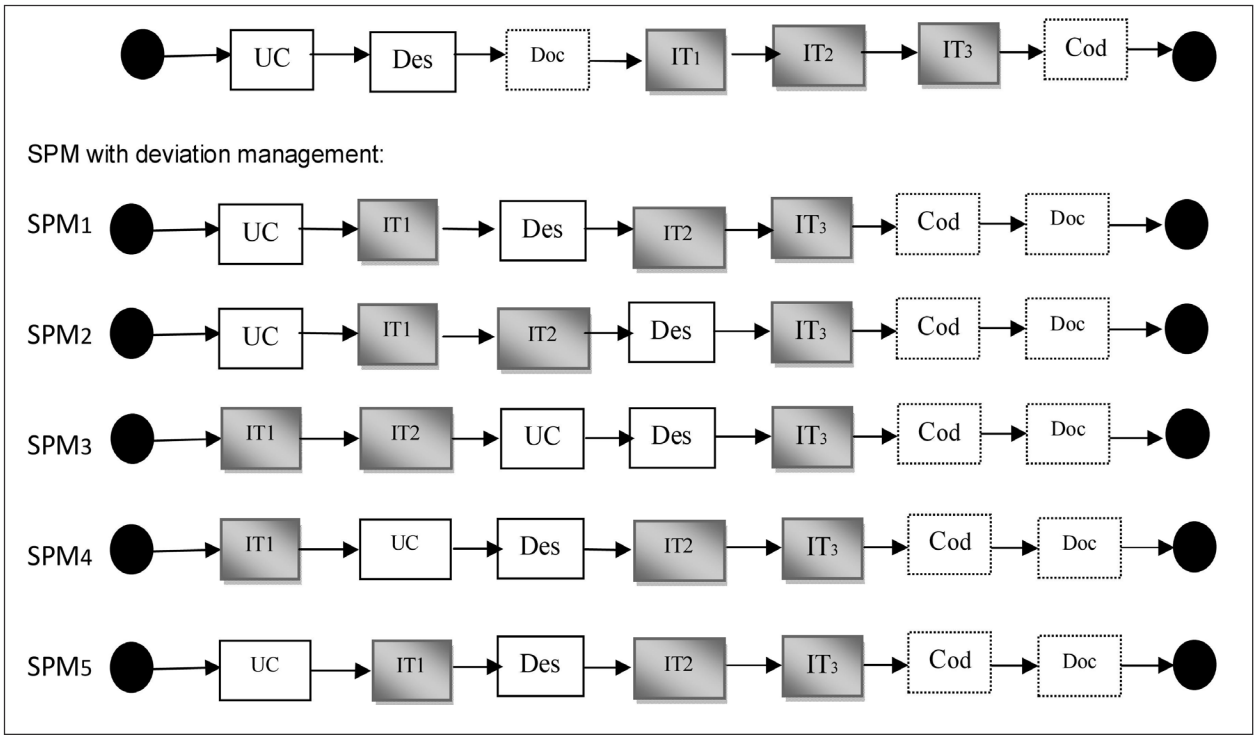
input (inOutput (Coding), ReDesign) where Coding = IT1

deviation("launch",Coding) ∧ validated (ReDesign) → inOutput ((output (Coding), Coding)

Since DesignModel is having the highest correlation-Index value and also considered as mandatory, it will be the first to be executed, then as we defined that UseCase should produce an output to DesignModel, it will be executed and an iteration will be used to update the DesignModel, documentation will be skipped or let till the end of the execution as per the rules we will consider. Figure 10 presents the process model deviation handling.

The process engineer can define a certain number of executions in order to constitute a set of adapted process models which can be structured in a defined data file. The file will be enriched by the flow of executions taking in consideration all the possibilities.

This flow can be illustrated as below where SPM0 represents the reference model where no deviation is detected:



Process model engineer will define a number of executions to create the set of data which will be the input of our machine learning system.

In order to use Weka, we need to create and load data set first the Attribute-Relation File Format (ARFF), the file structure will be as below:

@relation SPM

@attribute
@data

The attributes will be: 'UseCase'; 'Desc'; 'Documentation'; 'Iteration1', 'Iteration2', 'Iteration3', 'Code' all of type real, their values define the order of the activity during the process model execution. If the activity is skipped the value will be 0.

The last attribute is SPMwhich will have one of the values $\{SPM0, SPM1, SPM2, SPM3, SPM4, \text{and } SPM5\}$, it contains the software process model generated by the optimized approach when a deviation is detected.

The representation of the process model execution will be as below:

$1, 2, 3, 4, 5, 6, 7, SPM0$

$1, 3, 7, 2, 4, 5, 6, SPM1$

$1, 4, 7, 2, 3, 5, 6, SPM2$

$3, 4, 7, 1, 2, 5, 6, SPM3$

$2, 3, 7, 1, 4, 5, 6, SPM4$

$1, 3, 7, 2, 4, 5, 6, SPM5$

In our example, after 28 executions, Weka will produce a classifier. The classifier is listed under Results List as trees. RandomTree with the time the modeling process started. Weka will keep multiple models in memory for quick comparisons. It will also display in the box Classifier output some model performance metrics.

Relation: SPM

Instances: 28

Attributes: 8

'UseCase / Desc / Documentation /Iteration1 / Iteration2 / Iteration3 / Code / SPM

RandomTree

*Iteration*1 < 1.5

| '*UseCase*' <2.5 : *SPM*4 (5/0)

| '*UseCase*' >= 2.5 : *SPM*3 (3/0)

*Iteration*1 >= 1.5

| *Iteration*1 < 3

| | *Desc*<3.5 : *SPM*5 (16/1)

| | *Desc*>= 3.5 : *SPM*2 (3/0)

| *Iteration*1 >= 3 : *SPM*0 (1/0)

The table below represents the detailed accuracy by class and the confusion matrix:

Figure 11 presents the classifier tree output of the set of execution; the values taken by each attribute will guide the process model engineer in predicting the most appropriate SPM to be considered.

The main goal of process model engineer is to minimize the number of deviations as they are subject of time and cost wasting.

Based on our sample, the likelihood of occurrence for SPM as shown in the performance metrics is SPM5.Process engineer can decide whether to update the process model in order to replace the SPM0 or to reinforce controls to ensure that Iteration1 should not be started unless the previous steps are finalized. In our sample we used a machine learning algorithmas a decision tool to predict the appropriate process model of SPM deviation.

• Second Scenario: Machine Learning to Dynamically Define the SPM:

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0	0	0	0	0	0.5	SPM0
	0	0	0	0	0	0.222	SPM1
	1	0	1	1	1	1	SPM2
	1	0	1	1	1	1	SPM3
	1	0	1	1	1	1	SPM4
	1	0.154	0.882	1	0.938	0.864	SPM5
Weighted Avg.	0.929	0.082	0.866	0.929	0.895	0.882	

=== Confusion Matrix ===

a	b	c	d	e	f	<— classified as
0	0	0	0	0	1	a = SPM0
0	0	0	0	0	1	b = SPM1
0	0	3	0	0	0	c = SPM2
0	0	0	3	0	0	d = SPM3
0	0	0	0	5	0	e = SPM4
0	0	0	0	0	15	f = SPM5

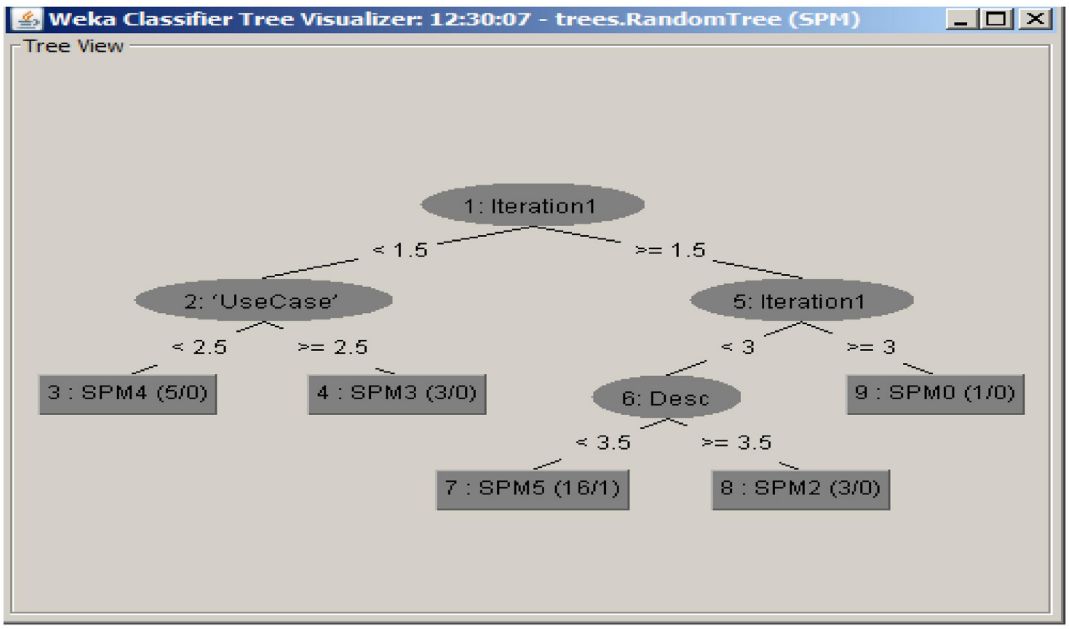
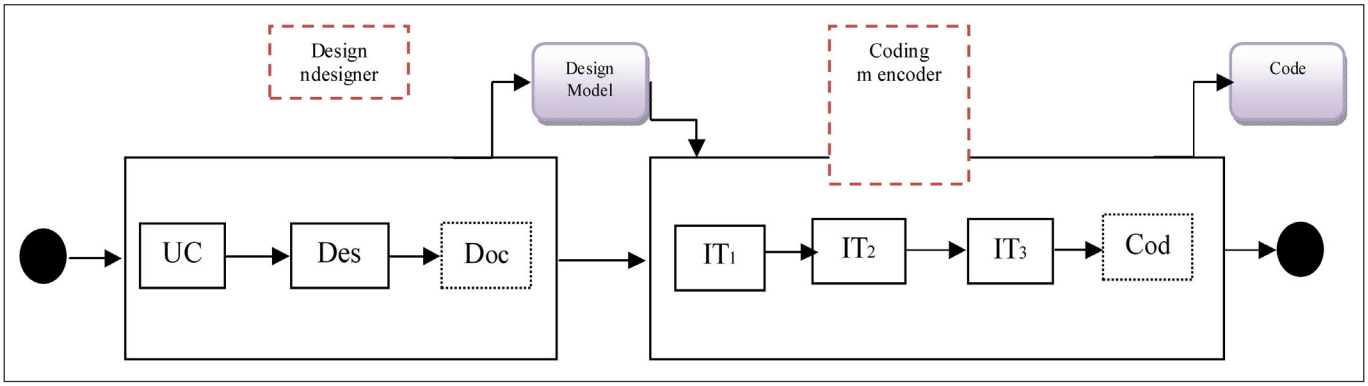


Figure 11. Classifier tree (First scenario)



The conditions of the project execution, project size, composition of each team and expertise level of each team (designers and encoders) can be used to predict the appropriate process model and anticipate its execution.

In other words, before the start of the project execution, based on the elements defining the project, the machine learning system can predict the pattern of the execution as well as the possible deviations that can be generated as well as acceptance of the deviation, this will help to modify some parameters by the process model engineer.

In the second scenario, we consider the size of the project, number of designers and encoders, their expertise level and the acceptance level of the process model.

The format of the dataset file will be presented as below:

@relation PredictSPM

@attribute

@data

The attributes will be: 'Size'; 'Designers; Encoders'; DesignerLevel, EncoderLevel and Acceptance

Project size list will have the values {'small', 'medium', 'big'}

Designers and Encoders are two attributes that will be of type real

DesignerLevel and EncoderLevel will have values in the list {'beginner', 'junior', 'senior'}

Acceptance will have a value in the list {'low','medium','high'}

The last attribute is SPM which will have one of the values {*SPM*0, *SPM*1, *SPM*2, *SPM*3, *SPM*4 and *SPM*5} which contain all the possible SPM generated by the optimized approach during the training phase.

The data will be the execution classification of the elementary activities as below:

Small, 1, 2, beginner, junior, low, SPM1

Medium, 2, 4, junior, junior, medium, SPM2

Big, 3, 6, senior, senior, high, SPM3

....

Small, 1, 1, beginner, beginner, low, SPMn

In our second example, we choose to materialize the machine learning system after 18 executions, Weka will display in the box Classifier output some model performance metrics.

Relation: PredictSPM

Instances: 18

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0	0	0	0	0	?	SPM0
	0.5	0.063	0.5	0.5	0.5	0.719	SPM1
	1	0	1	1	1	1	SPM2
	1	0.154	0.714	1	0.833	0.962	SPM3
	0	0.059	0	0	0	0.471	SPM4
	0	0	0	0	0	0.5	SPM5
	0	0	0	0	0	0.471	SPM6
Weighted Avg.	0.778	0.053	0.698	0.778	0.731	0.871	

=== Confusion Matrix ===

a	b	c	d	e	f	<— classified as
0	0	0	0	0	0	| a = SPM0
0	1	0	0	1	0	| b = SPM1
0	0	8	0	0	0	| c = SPM2
0	0	0	5	0	0	| d = SPM3
0	1	0	0	0	0	| e = SPM4
0	0	0	1	0	0	0 | f = SPM5
0	0	0	1	0	0	0 | f = SPM6

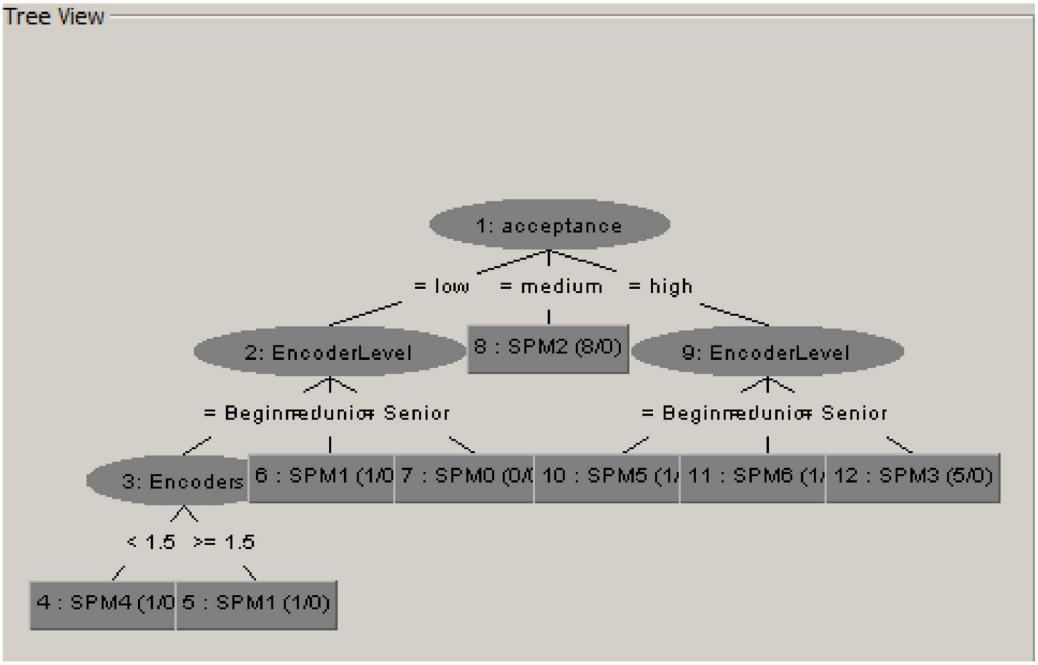


Figure 12. Classifier tree (Second scenario)

Attributes: 7

Size/ Designers / Encoders / DesignerLevel / EncoderLevel / acceptance / SPM

RandomTree

==========

acceptance = low

| EncoderLevel = Beginner

| | Encoders <1.5 : SPM4 (1/0)

| | Encoders >= 1.5 : SPM1 (1/0)

| EncoderLevel = Junior : SPM1 (1/0)

| EncoderLevel = Senior : SPM0 (0/0)

acceptance = medium : SPM2 (8/0)

acceptance = high

| EncoderLevel = Beginner : SPM5 (1/0)

| EncoderLevel = Junior : SPM6 (1/0)

| EncoderLevel = Senior : SPM3 (5/0)

Figure 12 present the classifier tree output of the simulation; this will constitute a decision tree to predict based on initial parameters the path that execution will follow and the appropriate SPM that will be executed based on the project conditions.

Weka will help to predict the SPM output of the execution and decide based on the acceptance list whether to move with it or to modify the program parameters, a web service can be put in place to update the parameters before starting the execution to avoid the deviation before its occurrence having knowledge of the previous executions. The tuning will update the output SPM based on the changes we will operate on the parameters to define the most appropriate SPM to our execution.

5. Conclusion

Treating the deviation on the fly using various approaches was limited to replacing fragment and launching adapted process model. The new challenge was how to take profit from the execution history under different conditions in order to define the most appropriate process model depending on the characteristics of each project.

Machine learning techniques have been suggested to answer this need: Implementing a predictive method based on the history of the various executions and their conditions: Different adapted models which were the output of the deviation handling.

The approach we have proposed is certainly optimal and predictive, but it does not completely solve the problem of deviation management. Indeed, it focuses on deviation management linked to the development context, through the use of decision rules that express guidelines which can be strengthened by redefining rules.

However, in practice, a general approach must be defined to build this base of rules and ensure its coherence. The choice of rules will have to be defined according to the context.

Our approach must also be perfected to be applicable through a real industrial project. Its implementation with industrialsoftware would allow a real validation to this work.

Another perspective is to automate the SPM model by

interfacing Sirius, Drools and Weka to implement a dynamic monitoring system based on the process model execution. This will reduce the time consuming by updating the monitoring system SPM, minimizing deviations occurrence and adopting the most accurate process model.

References

[1] Bandinelli, S., et al.(1994). SPADE: An Environment for Software Process Analysis, Design and Enactment. In: Finkelstein, A., et al. (eds.) Software Process Modeling and Technology, p. 223–244. Wiley, London.

[2] Cugola, G. (1998). Tolerating Deviations in Process Support Systems via Flexible Enactment of Process Models. *J. IEEE Transactions on Soft. Eng.* 24 (11) 982–1001.

[3] Kabbaj, Mohammed., Lbath, Redouane., Coulette, Bernard (2008). A deviation management system for handling software process enactment evolution. *In:* Proceedings of the Software process, 2008 international conference on Making globally distributed software development a success story (ICSP'08), Qing Wang, Dietmar Pfahl, and David M. Raffo (Eds.). Springer-Verlag, Berlin, Heidelberg, 186-197.

[4] Kabbaj , I.M (2009). Deviation Management in software process enactment (Gestion des déviations dans la mise en oeuvre des procédés logiciel, Université de Toulouse.

[5] Almeida da Silva, M.A., Bendraou, R., Robin, J., Blanc, X. (2011). Flexible Deviation Handling during Software Process Enactment, Enterprise Distributed Object Computing Conference Workshops (EDOCW), 2011 15th IEEE International , p.34,41, Aug. 29 2011-Sept. 2.

[6] Chaghrouchni, T., Kabbaj, M.I., Bakkoury, Z. (2014). Towards Dynamic Adaptation of the Software Process(IEEE), *In*: SITA'14 "-9th International Conference on Intelligent Systems: Theories and Applications",INPT, Rabat, Morocco.

[7] Chaghrouchni, T., Kabbaj, M.I., Bakkoury, Z. (2015). Optimized Approach for Dynamic Adaptation of Process Models (Springer), MedICT'15 "-Mediterranean Conference on Information & Communication Technologies' 2015",Saidia, Morocco.

[8] Frank, Eibe., Hall, Mark., Holmes, Geoffrey., Kirkby, Bernhard Pfahringer, Richard., Witte, Ian H., Trigg, Len (2010). Weka-A Machine Learning Workbench for Data Mining. Springer.

[9] Castro, Gabriella., Costa, Barbosa (2016). Using Data Provenance to Improve Software Process Enactment, Monitoring, and Analysis, 2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C), Austin, TX, USA