

A Framework to Detect Deviations during Process Enactment

Sean Thompson
La Trobe University
sean@nostin.com

Torab Torabi
La Trobe University
T.Torabi@latrobe.edu.au

Purva Joshi
La Trobe University
purva.joshi@gmail.com

Abstract

People enacting processes deviate from the process definition for a variety of different reasons, the consequences of which may be either positive or negative. Detecting deviations “on the fly” provides an opportunity to determine the precise point in the process where the deviation occurred plus a faster and more accurate diagnosis of any issues or problems and their rectification. We present a framework designed to detect deviations in enacting process “on the fly” consisting of a definition engine in which boundary values and process constraints are set, a monitoring engine which observes and records the process during enactment and a detection engine which compares reference values against actual values as they are recorded. The methods used are related to some of the work previously conducted in the process improvement and deviation detection domain, and brings these concepts together with our own ideas as a software tool. The developed model is tested with a simulated process and the findings are evaluated.

1. Introduction

Much work has been conducted, particularly onward from the early 1980's ([10], [11]), in the domain of process improvement. Considerably successful workflow products [15] emerged and the Workflow Management Coalition was established in 1993 [12]. Various software process improvement models were conceived and there was a migration into business process reengineering [24]. Also, in 1989, Watts Humphries conceived the Capability Maturity Model (CMM) by synthesizing various process and quality concepts into a model applicable in software development [25].

Research has been conducted in detecting deviations in processes sporadically from the mid 1990s up until now. The 1995 paper [7] on how to

deal with deviations during enactment was based on observing enactments and providing support to the handling of inconsistencies that occur as the enactment diverges from its process description. State machines were used to describe the process with legal state transitions between states constrained by preconditions assigned to each state. If a precondition is violated due to user interaction, the process is allowed to proceed as long as the invariant assertions that define safe states hold, and they refer to this as a “tolerable deviation”. If an unsafe state is entered, the process is terminated.

Later in 1999, Cooke and Wolf [6] took the approach to mine collected event data in software processes in an attempt to derive a formal model of that process which they call *process discovery*. Three algorithms were evaluated using different methods and they determined the superior model was based on the Markov Model - a statistical method based on probability used to determine a sequence model from a set of data [5].

Another method to detect deviations was described in [21] and is based on fuzzy-logic. The authors claim their technique gives a significant advantage over other approaches due to its “ability to naturally represent uncertain and imprecise information”. They provide a language for defining monitoring models which is executed by the OMEGA environments engine. The information handled by the monitoring system is represented with fuzzy sets theory.

The 2006 approach taken by Huo, Zhang and Jeffery [3] is similar to the work conducted by Cook and Wolf and relies on collecting low level process data during enactment of a predefined process so that the collected data may be mined to determine the process pattern. The enacted process pattern discovered by the mining is then compared to the predefined model to determine any deviations which may have occurred. The data derived from this also assists in refining the process.

Our approach makes use of the distinction made in [18] between “inconsistencies” and “deviations” where inconsistencies are a concept related to states whereas deviations are a concept related to *transition*. We therefore incorporate a set of preconditions (similarly to [7]) and post-conditions stipulating when an activity may be commenced and concluded. The framework is also comprised of attribute constraints which when violated constitute *inconsistencies* in the process. The boundaries for process attributes along with the conditions are defined in the definition engine and stored in a common database. When a defined process is executed, the values monitored and recorded by the monitoring engine are compared with the defined boundaries and constraints and any discrepancy between the two is logged and a deviation is flagged. Our approach is outlined in greater detail in section 2 of this paper. Section 3 outlines our model design, section 4 presents a case study and findings and section 5 the evaluation, future work and conclusion.

2. Approach

2.1 Overview

We define a *process* loosely as a conglomeration of people working with machines and methods in set tasks to achieve a desired outcome [2]. Since this research is aimed at generic processes, and some processes may have a heavy level of human involvement, an important distinction to be made when defining a process is the “human element” that is surmised well in [18] in the authors discussion of *human-centered systems*. These are considered systems that must have a significant level of human involvement and are defined in [16] as being “a combination of humans, organizational structures, rules, procedures and computerized tools”.

Furthermore, a process can be described as a set of *activities* - small, modular parts of the process that may be executed sequentially, concurrently, simultaneously, overlapping and in parallel ([1], [3]). Activities can be considered as the individual tasks performed within a given process or as described in [23] a framework in which *actors* (the “doers”) carry out tasks. Chroust and Hardt [8] classify process activities as one of two types: algorithmic or creative. The authors define algorithmic activities as being deterministic in nature with known resource consumption and may be carried out without significant human intervention. Creative activities depend largely upon the person(s) conducting them and are influenced significantly by the

creativity of that person(s). Creative activities are therefore said to be unpredictable in nature. Process activities also use *resources*, which are either consumed for some kind of benefit or generated for use later or for other processes. Like [14], we hope the approach we have taken may be applicable in a range of processes in different domains.

2.2 Deviations

There will usually be observable differences between a process enactment and its definition which will vary in magnitude. As in [21] we expect there will be these “deviations” and concur that there should be some way of identifying which anomalies should be acceptable or tolerated and which should be scrutinized. We have decided to address this issue in our approach by specifying another standard to our *activity* definition – that they be one of two types: *element* or *exception*. Activity elements detail the “ideal” method in which the process should be executed. However after observing the process being executed multiple times, common deviations may start to appear, indeed deviations we may expect to occur before the process is even enacted at all. We can define *exceptions* for actions performed that are out of scope and consider them as “tolerated exceptions”, being an aspect of the process that ideally should not occur but is still possible or likely to occur.

When detecting deviations, it is critical that a precise part of the process is identified as the scene of the deviation. [19] Argues in their research on process defects that once a defect is detected in a process, in order to avoid repeating the defect in the future, we must be able to relate the defect to the offending process activity. The same is true of process deviations.

[18] Differentiates between an inconsistency and a deviation, defining an inconsistency as a concept related to states whereas a deviation is a concept related to *transition* between states, which we adopt for this research. They also define deviations as falling into one of two levels: domain level and environment level. A domain level deviation is when a process activity is executed which does not conform to its expected behavior. Environmental level deviations are actions performed when enacting where there is no existing definition to describe the behavior taken place.

It is important to also consider why deviations occur. [22] Gives the following reasons for deviations:

- The process definitions omit or do not allow for relevant project contingencies;
- sometimes risks are taken;
- some process definitions are more amenable to deviations;
- people have good ideas, some of which are better than the defined processes;
- the process does not make sense either because of individual differences or because of lack of training;
- A lack of commitment or interest.

We make two additions to this list: user error and malicious users. We are all sometimes prone to making mistakes which could lead to a process deviation. Also sometimes unfortunately people simply behave in a malevolent way.

2.3 Inconsistencies

2.3.1 Overview. A “deviation” defined in [17] as being the difference between the *actual* value of a system variable and the expected value is what we adopt for our definition of *inconsistency*. Since part of our framework deals in specifying boundaries on expected values returned by the process, a list of inconsistency types have been presented which are used by our current model. This list covers a concise collection of the more important inconsistencies that may occur during process enactment.

2.3.2 Wrong Actor. In a process, it is important when assigning activities that the actor compelled to fulfill it be in a competent position to do so (even if the activity is automated, there should still be a responsible human “actor” assigned). As asserted in [9], “in order to bind an agent to an activity, the agent must be able to understand the activity description”. The framework therefore facilitates in the definition tier one or more “actor groups” which would refer to a particular department in an organization or any group of people that can be classified as to having the same skill set, such as testers, designers or developers. When the actor who enacts the process activity is required to check in its completion, the “actor group” the actor belongs to should be recorded and stored in the data repository with the other process enactment data. If the actor who completed the activity belongs to a group not listed as acceptable in the process definition, we consider that an inconsistency, which is logged.

2.3.3 Invalid Value. An *activity*, for the purposes of this research should return no more than one value. Therefore the boundary types we may define for the return value can be of type *range*, *list* or *none*. “List” denotes a list of acceptable values we store in the activity definition and specifies that the enacted activities return value must be one of the values defined in the list. Range denotes a numeric (or date/time) minimum and maximum and the enacted activities return value must fall inside this range. “None” denotes no constraints applied. When comparing the activity definition to its enactment, we compare the value returned firstly to the defined return type. If “None” is defined for the return value type, this does not mean the activity must not return a value, it only means we are not checking against a range or a list for an inconsistency. If the definition defines a List, the enacted return value will be compared against the defined list values to ensure it matches one of them, if not an inconsistency is recorded. Lastly, if the definition states a range, the value returned will be checked to see if it falls within the defined range. If not, an inconsistency is recorded.

2.3.4 Time Violation. Similar to the concept outlined in [3] where *tasks* come with a finite period of time attached to them, each activity in a process may have minimum and maximum time applied to determine how long the activity should run for when executed. If an activity runs for an unduly short or excessively long duration, an inconsistency will be recorded.

2.3.5 Excessive Activity Executions. In some processes, activities may be repeated. It is possible in our framework definition to limit the number of times any one activity can be repeated. If this number is exceeded, an inconsistency is recorded.

2.3.6 Illegal Number of Process Activities. In the overall process, we may want to place boundaries on how many total activities should be executed while enacting the process. Too few activities executed in a long process may be cause for concern and too many activities executed in a short process likewise. Therefore, the framework supports in the definition boundaries on the minimum and maximum number of total activities that may be executed in the process. An enactment which executes a number of activities outside these boundaries will flag an inconsistency.

2.3.7 Too Many Exceptions Executed. Exceptions, which we expect to occasionally occur, can also be

limited. The process may have a limit put on the number of exceptions allowable such that if an excessive number of exceptions are executed in an enactment, an inconsistency is flagged.

2.3.8 Invalid Number of Resources Consumed or Generated. Process activities may consume or generate resources. Boundaries may be defined on what type of resources activities may consume or generate and a numeric range specifying the quantity expected. If the number of resources consumed or generated for any enacted activity is outside the defined range, an inconsistency is recorded.

2.4 Conditions

Each activity in a process may reference various defined conditions which specify under which circumstances the activity may legally commence and/or conclude. We consider conditions specifying when an activity may commence to be *preconditions* and those which specify when an activity may conclude to be *post-conditions*. The same idea is discussed in Little-JIL [28] in reference to what they call “requisites” in “steps” in a process which the authors argue process definitions benefit substantially from. Our approach excludes the “sequencing badge” and incorporates sequencing as part of the conditions.

Activity conditions (being both preconditions and post-conditions), are broken down a further two levels - *rule sets* and then into *rules*. Firstly, a *rule* we define as an individual and singular stipulation that may return true or false. A *rule set* we define as a group of *N* number of *rules* and each rule in the rule set is also marked as to whether it should return true or false. If a *rule* in a *rule set* is marked to return true, and when the process activity is executed the rule is checked and it does return true, then the rule passes. The same of course, goes for if the rule is marked to return false and it does at execution. Every *rule* in a *rule set* must pass at execution for the *rule set* to pass.

Activity conditions are in turn made up of *N* number of *rule sets*. For a condition to pass, at least one *rule set* defined in the condition must pass. If all *rule sets* fail, then the condition fails and that constitutes a deviation. A simple example follows to aid in this explanation:

A process may exist for depositing cash at a bank. In this process, an activity may be defined which can be described as “Credit customers account” in which a

bank teller adds a certain deposited amount of money to a customers bank account. A *precondition* exists that should be satisfied before embarking upon this activity. This *precondition* is defined as follows: Either the amount to be deposited should be less than \$10,000 in value OR a police check should be issued on the customer. If the police check comes back clean, the money is deposited. We can formalize this precondition as outlined below in figure 1:

Precondition	
Rule Set	
Rule	Returns
Amount is < \$10,000	true
Rule Set	
Rule	Returns
Amount is < \$10,000	false
Activity "Police Check" Status == "Clean"	true

Figure 1 - Precondition Format

From the diagram, we can see two rule sets defined in the precondition each containing rules. Formalizing the conditions in this way serves as an AND OR type relationship. Each rule in a rule set has an AND relationship with every other rule in that set and every rule set in the condition has a non-exclusive OR relationship with every other rule set.

Since we are using a relational database to store all process data, including the definition data, SQL queries may be used to formalize each individual condition. If the query returns a result, the condition returns true. If not, the condition returns false. For example, consider the following database table description which denotes a simple enacted process activity repository data store:

Activity_Enactment	
PK	ActivityEnactID
	Start_Time
	Enc_Time
	Returnec_Value
FK	ProcessEnactID
	Status
	ActivityDefID
	ActorStaffID

Figure 2: Activity Enactment ER

From this we can deduce a sequence condition by the following SQL query:

```
SELECT * FROM Activity_Enactment
WHERE ActivityEnactID = '[Preceding_Activity_ID]'
AND STATUS = 'Complete';
```

The query checks whether or not a particular activity has been executed and if so, will return a data row. If it does, this means the condition is true and

holds, otherwise if the query returns nothing, the condition is false. We can use SQL queries to represent any kind of condition that we wish to apply to a process activity.

Conditions may also be used to sequence the process. For example, a precondition for a given activity may stipulate that another process activity must have been completed beforehand. We may define a precondition for an activity such that another activity in the process must have a status of “Complete” for the condition to be satisfied. Using conditions can help sequence a process in so that a process support engine may be able to allocate process activities at appropriate times given the activity conditions defined.

3. Architecture

The model we propose is comprised of three engines which operate off a relational database common to each.

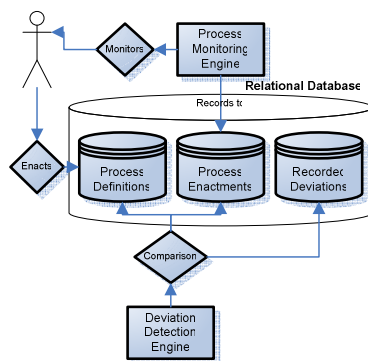


Figure 3: High Level Architecture

3.1 Definition Engine

The definition engine is software we have built to provide an interface to define new processes and modify existing ones. When defining a new process, it must first be broken up into individual activities. Some generic information about each activity is entered into the definition engine along with the boundary values and conditions that constrain the activity. Constraints may also be defined and modified for process wide inconsistencies (covered in 2.3). These constraints are the most important part of the definition as they define the values that will be compared against actual data so that detections may be made.

Process resources may also be defined using this engine along with the activities that produce or

consume them. Limitations may be set on the minimum and maximum number of which specific resources a given activity should produce and/or consume.

The constraints applicable to processes using this model are not compulsory, which gives the user the freedom to choose which constraints are appropriate for any given process or activity and how tightly they should be constrained. As more and more defined processes are executed, this engine may access the data history returned by them and use it to calculate better constraint values.

3.2 Monitoring Engine

The role of the monitoring engine is to record the data being returned by the process and then sort and correctly insert it into the process enactments repository. Once the process data is inserted into the enactments repository it can be easily compared with its definition to detect deviations and inconsistencies. To simulate this task, we developed an interface for the user to enter process data manually into the system which will record it correctly in real time.

As the data is being recorded, messages are sent through an API to the concurrently running detection engine, which tells it the nature of the data that is being recorded and the corresponding process and activity. All data recorded by the monitoring engine is also kept so that as more processes are enacted, more valuable data is stored which can be used to improve the process further and improve boundary values and constraints.

In order to record a process effectively in real time we assume a presently operating process support system such as the framework proposed in [27] which has the ability to allocate activities based on existing data and aid in the recording of other valuable process data.

3.3 Detection Engine

The deviation detection engine communicates with the monitoring engine via an API developed for the two applications. As the monitoring engine records and inserts new process data into the enactment repository, the detection engine compares the new data with the definition data. Whenever a process activity either commences or concludes, the detection engine will check all the appropriate pre or post conditions associated with the activity.

Detectec_Deviations	
PK	Deviation_ID
PK	Enactec_Process_ID
	Enactec_Activity_ID
	Timestamp
	Severity
	Value_Returned
	Type

Figure 4: Detected Deviations

When the monitoring engine records process data, the detection engine identifies it and compares it with the reference data in the definition. If an anomaly is found in the recorded data, a deviation is logged along with the relevant data shown in figure 4 (except severity which is not yet implemented).

4. Case Study

4.1 Example Process

The case study presented to illustrate the model is based upon a simple process of how cash is deposited at a bank. The process is described in figure 5:

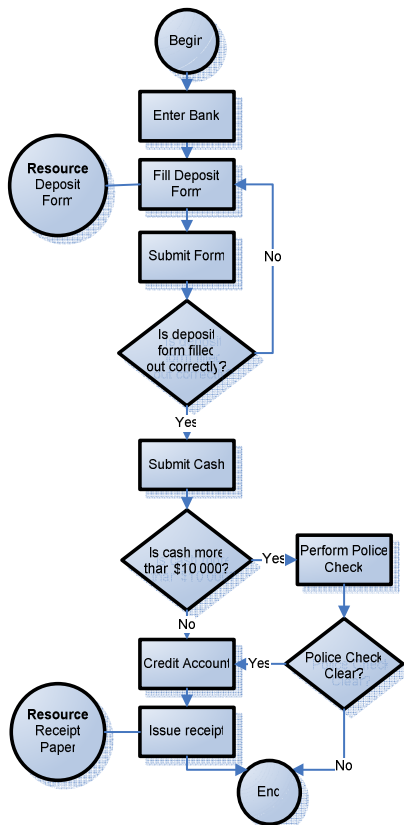


Figure 5: Example Process Flow Chart

This process was described using our definition engine with appropriate constraints placed upon the activities, such as time limits, actor constraints (customer filling the deposit form and teller doing the police check) and appropriate conditions such as a precondition for the “Submit Form” activity being the “Fill Deposit Form” activity having a status of “complete”. Also implemented was the precondition for the “Credit Account” activity described in figure 1. A screenshot of the interface used to apply condition constraints to activities is shown in figure 6.

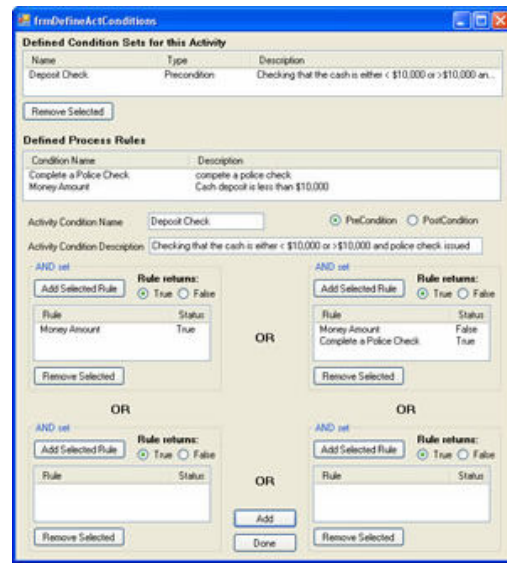


Figure 6: Defining Conditions

4.2 Findings

The cash deposit process was simulated using our simulation engine simultaneously with the deviation detection engine. We simulated the process manually (including timestamps) and detected deviations on the fly using an API running between the monitoring and detection engines. Every time process data was entered into the simulator, a message was sent to the detection engine telling it to check what was just entered against the definition constraints. Every time an activity was simulated and completed, the status would be changed to “complete” and the detection engine would then scan the activity data for possible deviations. Inconsistencies such as too many process activities or exceptions were calculated at process completion.

Given this simulation, the engines managed to pick up all deviations deliberately simulated in this process. As each deviation or inconsistency was detected, an

alert appears and they were logged to be displayed in the end process report.

4.3 Evaluation

The deviations picked up on the fly in the process simulation can be used to improve the process where necessary and avoid time and resource wasting. For example, one of the deviations picked up was a “Wrong Actor” inconsistency. The “Credit Account” activity is allocated to the “Teller” group in the process definition however in this simulation the activity was performed by someone belonging to the “Manager” group. Since this deviation was picked up straight away, we can easily deduce the reason without later investigation. Perhaps the teller had to step out for a moment and the manager temporarily took the tellers place at the desk. If this inconsistency was detected much later, the cause may not have been so clear.

Suppose the precondition shown previously in figure 1 held for this process. If the teller was performing unnecessary police checks (i.e. if the amount deposited was less than \$10,000) the precondition would fail and a deviation would be detected instantly and a solution could be implemented such as informing the teller not to perform such checks unless the deposit amount was over \$10,000. Being able to detect such deviations and inconsistencies in the process on the fly not only take care of problems that could potentially escalate sooner, but aid in implementing solutions since managers can deal with issues caused by deviations as they arise.

5. Conclusion and Future Work

5.1 Summary

In this paper, we have outlined how constraints and conditions may be applied to a process in order to detect deviations and inconsistencies. We have shown how these constraints and conditions can be compared with real values returned by an enacted process and how the findings of discrepancies between these values may aid in the improvement of the process. We have also provided the definition for seven types of possible inconsistencies that may arise in a process along with a framework for constraining process activities with pre and post conditions. A case study was also examined and how the implementation of these techniques can be used to determine information about the process which can in turn be used to improve it.

5.2 Limitations

This model does not yet support the functionality to ascertain the severity of deviations/inconsistencies. The conditions and boundaries are also set by the user and for a new or rarely used process, this could cause problems either by inappropriate boundary values due to lack of knowledge, an inexperienced user or it simply may be too hard to gauge appropriate values due to a lack of existing process data.

5.3 Future Work

We plan to implement concepts used in Statistical Process Control to improve the way conditions and boundaries are defined in our future research. When enough data is collected from previously executed processes, we can apply a 3σ (σ being standard deviation) gap on either side of the average value recorded for each process boundary. This limit has worked well in the past in other business areas such as hardware manufacturing [13] and the 3σ range is suitable for minimizing both loss due to out of control processes and false alarms [2, 4, 20]. In turn, the more data collected, the more precise these boundary values will become. The values may also accommodate process evolution aiding in continuous improvement which is invaluable to any organization [26].

References

- [1] Rezgui, Y.; Marir, F.; Cooper, G.; Yip, J.; Brandon, P.; “A Case-Based Approach to Construction Process Activity Specification”, Intelligent Information Systems, 1997. IIS '97. Proceedings, 8-10 Dec. 1997 Page(s):293 – 297.
- [2] W.A. Florac and A.D. Carleton, “Measuring the Software Process: Statistical Process Control for Process Improvement”, Addison-Wesley, 1999.
- [3] Huo, M.; Zhang, H.; Jeffery, R.; “An Exploratory Study of Process Enactment as Input to Software Process Improvement”, International Conference on Software Engineering, 2006.
- [4] Jalote, P.; Saxena, A.; “Optimum control limits for employing statistical process control in software process”, IEEE Transactions on Software Engineering, Volume 28, Issue 12, Dec. 2002 Page(s):1126 – 1134.
- [5] Mohamed, M.A.; Gader, P.; “Generalized hidden Markov models. I. Theoretical frameworks”, IEEE Transactions on Fuzzy Systems, Volume 8, Issue 1, Feb. 2000 Page(s):67 – 81.

- [6] Jonathan E. Cook and Alexander L. Wolf; “Discovering models of software processes from event-based data”, ACM Transactions on Software Engineering and Methodology (TOSEM), Volume 7 Issue 3, July 1998.
- [7] G. Cugola, E. Di Nitto, C. Ghezzi, M. Mantione; “How to deal with deviations during process model enactment”, Proceedings of the 17th international conference on Software engineering, ACM Press, April 1995.
- [8] Chroust, G.; Hardt, S.; “Executing process models: activity and project management”, IEEE Symposium and Workshop on Engineering of Computer-Based Systems, 1996. Proceedings, 11-15 March 1996 Page(s):364 – 370.
- [9] Alho, K.; Lassenius, C.; Sulonen, R.; “Process enactment support in a distributed environment”, Proceedings of the Fourth Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, 1995, 20-22 April 1995 Page(s):54 – 61.
- [10] Victor R. Basili, Frank E. McGarry, Rose Pajerski, Marvin V. Zelkowitz; “Lessons learned from 25 years of process improvement: the rise and fall of the NASA software engineering laboratory”, Proceedings of the 24th International Conference on Software Engineering, ACM Press, May 2002.
- [11] Alfonso Fuggetta, “Software Process: A Roadmap”, Proceedings of the Conference on The Future of Software Engineering, ACM Press, May 2000.
- [12] WfMC website <http://www.wfmc.org>, Workflow Management Coalition.
- [13] Card, D.; “Statistical process control for software?”, IEEE Software, Volume 11, Issue 3, May 1994 Page(s):95 – 97.
- [14] Fernstrom, C.; “Process WEAVER: adding process support to UNIX”, Second International Conference on the Software Process, 1993. ‘Continuous Software Process Improvement’, 25-26 Feb. 1993 Page(s):12 – 26.
- [15] D. Avriilionis, N. Belkhatir and P. Y. Cunin; “A unified framework for software process enactment and improvement”, 4th International Conference on the Software Process, Brighton, England, 3-5 December, 1996.
- [16] Andrew Blyth, “Business process re-engineering: What is it?”, ACM SIGGROUP Bulletin, Volume 18 Issue 1, April 1997.
- [17] Jon Damon Reese, Nancy G. Leveson; “Software deviation analysis”, Proceedings of the 19th international conference on Software engineering, ACM Press, May 1997.
- [18] Gianpaolo Cugola, Elisabetta Di Nitto, Alfonso Fuggetta, Carlo Ghezzi; “A framework for formalizing inconsistencies and deviations in human-centered systems”, ACM Transactions on Software Engineering and Methodology (TOSEM), Volume 5 Issue 3, ACM Press, July 1996.
- [19] Inderpal Bhandari, Norman Roth; “Post-process feedback with and without attribute focusing: a comparative evaluation”, Proceedings of the 15th international conference on Software Engineering, IEEE Computer Society Press, May 1993.
- [20] W.A. Florac, A.D. Carleton, and J.R. Barnard, “Statistical Process Control: Analyzing a Space Shuttle Onboard Software Process,” IEEE Software, vol. 17, no. 4, July/Aug. 2000, pp. 97–106.
- [21] Sorana Cîmpan, Flavio Oquendo; “Dealing with software process deviations using fuzzy logic based monitoring”, ACM SIGAPP Applied Computing Review, Volume 8 Issue 2, ACM Press, December 2000.
- [22] Dewayne E. Perry, Alexander L. Wolf; “Session 1: People, Processes, and Practice”, Proceedings of the 9th International Software Process Workshop, IEEE Computer Society Press, October 5-7 1994, Airlie, Virginia, USA.
- [23] Mark Dowson, Brian Nejme, William Riddle; “Concepts for Process Definition and Support”, Proceedings of the 6th International Software Process Workshop, IEEE Computer Society Press, October 28-31 1990, Hakodate, Japan.
- [24] Thomas Barothy, Markus Peterhans, Kurt Bauknecht; “Business process reengineering: emergence of a new research field”, ACM SIGOIS Bulletin, Volume 16 Issue 1, August 1995.
- [25] Bill Curtis, “Software Process Improvement: Methods and Lessons Learned”, Proceedings of the 19th international conference on Software engineering, ACM Press, May 1997.
- [26] K. Schneider, T. Schwinn, “Maturing experience base concepts at DaimlerChrysler”, Software Process: Improvement and Practice, 6(2), 2001, pp. 85-96.
- [27] T. Torabi, T. Dillon, W. Rahayu, “A Software Process Framework”, Proc. IASTED Software Engineering Application Conference, Special Session on Process Modelling, MIT, Cambridge, USA, 6-9 Nov., 2002.
- [28] Jamieson M. Cobleigh, Lori A. Clark, Leon J. Osterweil; “Verifying Properties of Process Definitions”, ACM SIGSOFT Software Engineering Notes, Proceedings of the 2000 ACM SIGSOFT international symposium on Software testing and analysis ISSTA '00, Volume 25 Issue 5, ACM Press, August 2000.