Toward a Process Monitoring Automation: a Proposal

Nicola Boffoli, Danilo Caivano
Department of Informatics - University of Bari - Via E. Orabona 4 - 70126 - Bari – Italy, RCOST – Bari
email: {boffoli, caivano}@di.uniba.it

ABSTRACT
Software process monitoring is a complex activity. The predominant human factors that characterize these processes make automated monitoring a difficult task. This work presents and faces the main issues concerning process monitoring automation and suggests some solutions. Also, the paper presents the use of tools that support the automation of monitoring tasks in accordance to the solution proposed. Finally, the appendix presents an overview of the SPC-Toolkit, a kit of tools available via web that implement the proposed approaches.

KEY WORDS
Statistical Process Control, Six-Sigma, Decision Tables.

1. Introduction
A software process can be considered as a synergic blend of man, machine and methods in working activities whose execution leads to the production of desired outputs, starting from the available inputs [1, 2], let these be products or services. Product quality is tightly related to the quality of the processes used to produce them. Also, within the Software Engineering community a manner for “improving quality” is to improve software processes in order to improve software products. This implies the need for monitoring process execution, highlight process anomalies and quickly react to them.

Unfortunately, software processes are mainly human intensive and dominated by cognitive activities, this makes each process execution a creative and unique activity. The predominant human factor implies differences in process performances and thus multiple outputs. The phenomena known as “Process Diversity” [3, 4], determines difficulty in predicting, monitoring and improving a software process. This makes monitoring activities difficult to automate. For successful process monitoring, among others, there are mainly three problems to address that make the monitoring activity difficult to automate:

- **Problem1: Process Limits.** It is difficult to characterize the behaviour of the monitored process through baselines useful for evaluating its performances. Given the heterogeneity of the operative contexts, the different maturity levels of the processes in use, and the differences of previous knowledge on the monitored process, it is difficult to define an upper and lower limit for process performance variability.

- **Problem2: Process Anomalies.** Difficulty in attributing a meaning to “process anomalies” and in finding conceptual and operative tools for identifying them (what lays behind the concept of anomaly?)

- **Problem3: Sensibility.** Need to adapt the sensibility of monitoring activities to the continuous changing process performances.

The aim of this work is to generalize and put together the experiences collected by the authors in the previous studies [5, 6, 7, 8, 9, 10, 11], to contribute to the discussion on these issues and to propose potential solutions.

In order to do this, this paper presents the following proposal (structured in Figure 1):

- *a monitoring model* (section2) to overcome the problem presented above.

- *a monitoring automation* (section3) that automates the execution of process monitoring activities according to the proposed monitoring model

Finally conclusions are drawn.

2. Proposed Monitoring Model
In this section a pattern based approach will be followed, where the word “pattern” means a problem-solution couple. According to this, each problem will be presented and discussed and a potential solution will be proposed.
2.1. Problem 1: Process Limits

Software process monitoring is a crucial activity in the context of software process improvement initiatives. Monitoring involves measuring a quantifiable process characteristic (i.e., productivity, defect density, execution time...) over time and pointing out anomalies such as reduction in productivity, an exceptional defect density or an unattended execution time (too high or too slow).

A process must be characterized before being monitored, for example by using a couple of reasonable threshold values, one for the upper and one for the lower process performance limits. When the observed performance falls outside these limits, someone can argue that there is something wrong in the process.

Moreover, process characterization requires past knowledge but often this is not trivial. Many processes are based on paradigms, of a non-standard nature, involving the use of COTS, Open Sources, Web Services or other resources that affect the process performances in a non-predictable way. The process maturity, the project dimension, the number of people involved in a project and their experience together with thousands of other factors can affect process performances [6, 9, 12]. Thus it is challenging to define an upper and lower control limit to characterize process behavior. The only way is often to refer to expert judgment and use expert experience for estimating performances. Unfortunately, expert experience not necessarily includes the knowledge about the process in use. This implies that process monitoring seldom occurs.

**Proposed Solution.** There is a technique for time series analysis known as Statistical Process Control (SPC) [13, 14] that has shown to be effective in manufacturing and recently also used in software contexts. It was developed by Shewhart in the 1920s and then used in many other contexts. It uses several “control charts” together with their indicators to establish operational limits for acceptable process variation. By using few data points, it is able to dynamically determine an upper and lower control limit of acceptable process performance variability. Such peculiarity makes SPC a suitable instrument to face problem 1. Process performance variations are mainly due to: common cause variations (the result of normal interactions of people, machines, environment, techniques used and so on); assignable cause variations (arise from events that are not part of the process and make it unstable). A process can be described by measurable characteristics that vary in time due to common or assignable cause variations. If the variation in process performances is only due to common causes, the process is said to be stable and its behavior is predictable within a certain error range; otherwise an assignable cause (external to the process) is assumed to be present and the process is considered unstable. A control chart usually adopts an indicator of the process performances central tendency (CL) and upper and lower control limits (UCLs and LCLs). Process performances are tracked overtime on a control chart, and if one or more of the values fall outside these limits, or exhibit a “non-random” behavior an assignable cause is assumed to be present. Many control charts exist, but in software processes, due to the scarcity of data and since measurements often occur only as individual values, the most used ones are the XmR i.e. individual and moving range charts (Figure 2) [15, 16, 17, 18]. They will be briefly introduced.

In the X chart: each point represents a single value of the measurable process characteristic under observation; CL\_X is calculated as the average of the all available values; UCL\_X and LCL\_X are set at 3sigma around the CL\_X; sigma\_X is the estimated standard deviation of the observed sample of values. Formally, given a set of observations, \( X = \{x_1, ..., x_m \} \):

\[ \bar{X} = \frac{1}{m} \sum_{i=1}^{m} x_i \] and

\[ 3\sigma_{X} \approx 2.660 \times \bar{mR} \]

\[ UCL_X = \bar{X} + 2.660 \times \bar{mR} \]

\[ LCL_X = \bar{X} - 2.660 \times \bar{mR} \]

In the mR chart: each point represents a moving range (i.e., the absolute difference between a successive pair of observations); CL\_mR is, the average of the moving ranges; UCL\_mR = CL\_mR + 3sigma\_mR and LCL\_mR = 0; sigma\_mR is the estimated standard deviation of the moving ranges sample. Formally:

\[ UCL_{mR} = 3.268 \times mR \]

\[ LCL_{mR} = 0 \]

**Figure 2: Example of Individual and moving ranges charts (XmR charts)**

Sigma is calculated by using a set of factors tabulated by statisticians (for more details refer to [19]) and it is based on statistical reasoning, simulations carried out and upon the heuristic experience that: “it works”. A good theoretical model for a control chart is the normal distribution showed in Figure 3 where: the percentage values reported express the percentage of observations that fall in the correspondent area; \( \mu \) is the theoretical mean; \( \sigma \) is the theoretical standard deviation. In the \([\mu - 3\sigma, \mu + 3\sigma]\) interval, fall 99.73% (i.e. 2.14 + 13.59 + 34.13 + 34.13 + 13.59 + 2.14) of the total observations. Thus only the 0.27% of the observations is admissible to fall outside the \([\mu - 3\sigma, \mu + 3\sigma]\) interval.

If we consider sigma in place of \( \sigma \), the meaning and rational behind a control chart results clear. For completeness it is necessary to say that the normal distribution is only a good theoretical model but, simulations carried out have shown that independently
from the data distribution, the following rules of thumb work:

- Rule 1: from 60% to 75% of the observations fall in the [CL-sigma, CL+sigma]
- Rule 2: from 90% to 98% of the observations fall in the [CL-2sigma, CL+2sigma]
- Rule 3: from 99% to 100% of the observations fall in the [CL-3sigma, CL+3sigma]

Figure 3: Normal Distribution

The control limits carried out using SPC are based on a process observation and they are expression of it. They are not the results of expert judgement and, furthermore, there is a clear way to obtain them.

2.2. Problem 2: Process Anomalies

In the monitoring activity a fundamental task is to point out process anomalies in order to quickly react to them. But, what are process anomalies and how can we find them? The anomalies are in general some kind of noise in the process performances, the results of an unknown cause in action that implies unattended variation (in better or worse) and thus a lower process predictability. In the software context, relevant examples in this sense are: the introduction of a new case tool that speeds up the development; the use of a new testing or inspection technique that reduces the post release defects; the degradation of system maintainability due to the lack of documentation; the presence of an unknown bug in the hardware platform or in the operative system that leads to a crash of the application and unattended disservices; the involvement of low profile programmers in the project team that determine lower productivity; an unexpected absence of a key person that implies project failure, and so on. Anomalies are consequences of the presence of causes in the process that determine unattended performance variations (in better or worse). The aim of monitoring activity is to point out the anomalies and stimulate the search of the possible causes. The aim of Software Process Improvement is to find the causes, eliminate them if detrimental or, otherwise, make them part of the process. The possible causes of variation can be many such as their effect on process performances and, consequently, the observable anomalies. A standard mechanism is needed able to characterize an anomaly and, at the same time, to point out it.

Proposed Solution. The proposed solution is based on previous research of the authors who have proposed as solution a set of Run-Tests for selecting SPC indicators. According to the set, an appropriate Run-Test Interpretation concerning “what is going on” in the process, is also provided.

Run-Test Set. It is a selection of a set of indicators, among those presented in SPC literature, along with their arrangement in logical classes: sigma, limit and trend (Table 1).

<table>
<thead>
<tr>
<th>Class</th>
<th>Run-Test</th>
<th>Description</th>
<th>Zone Based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sigma</td>
<td>RT1: Three Sigma</td>
<td>1 point beyond a control limit (±3sigma)</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>RT2: Two Sigma</td>
<td>2 out of 3 points in a row beyond ±2sigma</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>RT3: One Sigma</td>
<td>4 out of 5 points in a row beyond ±sigma</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>RT4: Run above or below the CL</td>
<td>7, 8 or 9 consecutive points above or below the centerline</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>RT5: Mixing/Overcontrol</td>
<td>8 points in a row on both sides of the centerline avoiding ±sigma area</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>RT6: Stratification</td>
<td>15 points in a row within ±sigma area</td>
<td>Yes</td>
</tr>
<tr>
<td>Trend</td>
<td>RT7: Oscillatory Trend</td>
<td>14 alternating up and down points in a row</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>RT8: Linear Trend</td>
<td>6 points in a row steady increasing or decreasing</td>
<td>No</td>
</tr>
</tbody>
</table>

Note that the sigma tests, whose unreliability is estimated of less than 1% have been supported by other tests (in agreement to software process characteristics) which are not based on sigma, and that represent the probability of a “rare event”.

For example, leaving out rigorous discussions, test4 indicates the probability that 7 to 9 points fall on the same side of the central line. So, considering that the probability that a point falls on one side or another of the central line is of 0.5, the probability that at least 7 points fall on the same side is of 0.0078 (0.5^7). As so, the probability of a “rare event” is approximately that of a sigma test.

Finally, the indicators have been classified as zone based or not. This because the zone based indicators are no longer significant on Moving Range (mR) charts and their use is therefore not advised on such charts.

The next figure shows some patterns of observations detected by each Run-Test of the set:
Run-Test Interpretation. It considers the interpretation from the software process point of view, of every test in each class, in particular a class-level description of the run test interpretation is here presented (more details in [5]):

- **Sigma Tests.** They provide an “early” alarm indicator that must stimulate searching possible assignable causes and, if the case, their identification and further elimination. One, Two and Three sigma tests point out a potential anomalous “trend” that “may” undertake assignable causes. In general, due to the high variance in software process especially when we manage individual rather than sample data, the faults highlighted by these tests could be numerous but less meaningful than in manufacturing contexts. For example, in a manufacturing process a party of poor quality raw material may be a potential assignable cause that must be investigated and removed. In software processes a possible assignable cause may be an excessive computer crash due to a malfunctioning peripheral but also to a headache of the developer. Therefore the signals that Sigma tests detect may express a general behaviour determined by an assignable cause or passing phenomena.

- **Limit Tests.** This class of tests point out an occurred shift in process performance. They highlight the need to recalculate the control limits when the actual ones are inadequate, because they are too tiny or larger than required. In software process monitoring and improvement we represent a measurable characteristic that expresses a human related activity outcome (time spent, productivity, defect found during inspection etc.) on a control chart. Thus a “sequence” of points that Limit Tests detect means that something has changed within the process (i.e. performance mean or variability).

- **Trend Tests.** While the previous tests class points out the presence of an occurred shift, this one highlights an ongoing or just occurred phenomena that is resulting in an ongoing shift that needs to be investigated. Typically a failure in this test class can be the result of both spontaneous or induced process improvement initiatives.

More interpretation details about each run-test are summarized in Table 2.

**Table 2: Run-Test Interpretation details**

<table>
<thead>
<tr>
<th>Run-Test signals detected</th>
<th>Run-Test Interpretation what happens in the process</th>
</tr>
</thead>
<tbody>
<tr>
<td>No RT</td>
<td>“controlled” Process</td>
</tr>
<tr>
<td>RT1</td>
<td>just early alarms</td>
</tr>
<tr>
<td>RT2</td>
<td>just early alarms</td>
</tr>
<tr>
<td>RT3</td>
<td>just early alarms</td>
</tr>
<tr>
<td>RT4</td>
<td>performance mean changed</td>
</tr>
<tr>
<td>RT5</td>
<td>performance variability increased</td>
</tr>
<tr>
<td>RT6</td>
<td>performance variability decreased</td>
</tr>
<tr>
<td>RT7</td>
<td>source of performance variability changed</td>
</tr>
<tr>
<td>RT8</td>
<td>ongoing phenomena</td>
</tr>
</tbody>
</table>

Finally, according to the interpretations here described (and more detailed in [5]) we are able to define the following function:

\[ \Phi: \text{[Run-Test Set]} \Rightarrow \text{[Run-Test Interpretation]} \]

such function \( \Phi \) for each failed run-test assigns the appropriate interpretation and thus it is able to relate the statistical “signal”, detected during monitoring activities, to “what happens” within the process.

### 2.3. Problem 3: Sensibility

The concept of Process Diversity means that a process varies between different organizations, different projects and also during the execution of a project [3]. This implies estimation model diversity [20]. The need for recalibrating a model is well known in the software estimation community. Process changes impact on the parameters and drivers predicted by the estimation models in use and, as a consequence, they determine estimations inaccuracy. Thus, even when the predictors and corrective parameters are adequately estimated at the beginning of the project, their values tend to change during execution as the context changes. Typical in this sense is the so called “maturity effect”, i.e. an improvement of human performances due to the experience collected during process execution: a better knowledge on the techniques in use, a better confidence with the development tool etc.. Hence, even though a good initial estimation of the process performance is done, it will not prevent estimation errors during project execution.

A further confirmation in this sense comes from COCOMOII [21] that implies the use of different cost drivers for each development process stage (Application Composition, Early Design, Post Architecture).

All these considerations imply the difficulty in correctly characterizing process behaviour from the start to the end.
in terms of upper and lower control limit. The process performance limits must be recalibrated according to relevant process performance changes pointed out by process anomalies. The sensibility of the monitoring activity have to be tuned continuously. The risk of not tuning sensibility is to miss anomalies as the result of using larger limits than necessary or having several false alarms.

**Proposed Solution.** The monitoring activity based on SPC is carried out with control limits as baselines within which the process can vary. The latter is monitored according to specific characteristics (of the measurement object) selected by the SPC manager. Given the characteristics of the proposed monitoring activity, the inappropriateness of the SPC monitoring model, following to process changes can be attributed to the following factors:

- control limits are no longer appropriate: too tight, too wide or the central line is no longer representative of the average process performances;
- measurement object is no longer representative: the measures used may no longer express the process variability.

In both cases, in order for SPC definitions to be updated it is necessary to:

1. identify the process events that require such updates;
2. quickly intercept the situations in order to move on to the updates.

Point (1) has been faced in a generalization activity carried out on the experience acquired during empirical validation of the approach in a previous study [5]. This experience has allowed to generalize a set of relations between “what happens” in the process and what the best actions to undertake are. Table 3 synthesizes these relations.

<table>
<thead>
<tr>
<th>What Happens</th>
<th>SPC Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>“controlled” Process</td>
<td>no action</td>
</tr>
<tr>
<td>just early alarms</td>
<td>no action</td>
</tr>
<tr>
<td>performance mean changed</td>
<td>identify a new threshold (new reference set)</td>
</tr>
<tr>
<td>performance variability increased</td>
<td>identify a new threshold (new reference set)</td>
</tr>
<tr>
<td>performance variability decreased</td>
<td>identify a new threshold (new reference set)</td>
</tr>
<tr>
<td>source of performance variability changed</td>
<td>identify a new measurement object</td>
</tr>
<tr>
<td>ongoing phenomena</td>
<td>no action</td>
</tr>
</tbody>
</table>

Using such relations and considering that the “what happens” in the process has been described through the “run-test interpretation” we are able to define the following function:

\[ \Psi: \{\text{Run-Test Interpretation}\} \rightarrow \{\text{SPC Settings}\} \]

“what happens”  “what to do”

such function \(\Psi\) for each new event occurred in the process assigns the appropriate SPC Setting to update in order to preserve the sensibility of the process monitoring.

Point (2) is faced composing functions \(\Phi\) and \(\Psi\) previously defined, in order to obtain a new function \(\rho = (\Psi \circ \Phi)\):

\[ \rho: \{\text{Run-Test Set}\} \rightarrow \{\text{SPC Settings}\} \]

“the signals” “what to do”

such function \(\rho\) for each statistical “signal” is able to suggest the suitable action to undertake in order to preserve the sensibility of the process monitoring. Summarizing, the authors suggest a quick and effective solution that takes into account the issue of process monitoring sensibility and allows the proposed monitoring tool to identify the process anomalies through SPC signals, distinguish among anomalies and identify the most appropriate actions in order to update the SPC definitions, redefine the monitoring model in use and be careful to preserve its sensibility.

### 3. Proposed Monitoring Automation

The solutions presented in section 2 represent a proposal for overcoming the barriers that obstruct automation of software process monitoring. Given these solutions, this section presents the tools for monitoring automation. In particular two tools will be commented:

- **SPC-DecisionTable**: tool for representing and identifying the solutions proposed in section 2
- **SPC-InvestigationProcess**: the process that coordinates the monitoring activities based on the solutions in section 2

#### 3.1 SPC-DecisionTable

The authors propose a decision table as means for formalizing the above contents and make the concepts proposed in the previous paragraphs operative. More precisely, the table not only is able to identify the anomalous signals in the process, but it is able to distinguish different levels of warning among the anomalous signals and, identify the related actions to undertake.

Through the SPC-DecisionTable we are able to:

- formalize the relations between the “signals” that SPC detects and the “events” actually occurred within the process
- suggest the appropriate “actions” to preserve the approach sensibility
- update such relations whenever deep changes in the monitored process lead to new interpretations
- solve conflicts that arise among the “advice” suggested by SPC due to multiple and simultaneous events identified by the approach
- allow the interpretation of two control charts, X-Chart and mR-Chart, which is highly recommended in literature.

Consultation of each SPC-DecisionTable, starting from the Run-Tests (the “signals”) failed in the adopted control charts, extracts a list of advice for the process manager, in particular each consultation extracts:

- according to the \( \Phi \) function, the information about “what happens” within the monitored process
- according to the \( \rho \) function, the actions to carry out (in figure6 in ITALIC character) by the SPC-manager for updating the SPC settings and preserve its sensibility (i.e. recalculating control limits, selecting a different control chart, choosing a new measurement object, continuing the monitoring with the same control limits, …)

In order to well understand the implemented table, in the following sections the authors firstly introduce the basic concepts of decision tables and then describe the implemented SPC-DecisionTable.

**Decision Tables “pills”**. A decision table is a tabular representation of a procedural decision situation, where the state of a number of conditions determines the execution of a set of actions [22, 23, 24]. In general, a decision table is a table divided by a double line, horizontal and vertical, into four quadrants (Figure 5). The horizontal line divides the table in a conditional part (the top part) and an action part (the bottom part). Moreover, the vertical line divides the input values (left side) from the rules and combination of conditional states (right side). The table is defined so that each combination of conditions (conditional states) corresponds to a set of actions to carry out (rule). A tabular representation of a decision situation is characterized by a separation between conditions and actions on one end, and between rules and conditional expressions on the other. Each column of the table (decision column) identifies which actions should (or shouldn’t) be carried out for a specific combination of conditional states. The conditional oriented approach of a decision table allows to express all the knowledge related to the problem being considered.

<table>
<thead>
<tr>
<th>Individuals</th>
<th>mR Chart</th>
<th>Null</th>
<th>T1 or T2 or T3</th>
<th>T4</th>
<th>T5</th>
<th>T6</th>
<th>T7</th>
<th>T8</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cond</strong></td>
<td><strong>C1</strong></td>
<td><strong>C2</strong></td>
<td><strong>C3</strong></td>
<td><strong>C4</strong></td>
<td><strong>C5</strong></td>
<td><strong>C6</strong></td>
<td><strong>C7</strong></td>
<td><strong>C8</strong></td>
</tr>
<tr>
<td>Cond1 &amp; Cond2</td>
<td>C1 &amp; C2</td>
<td>C3 &amp; C4</td>
<td>C5 &amp; C6</td>
<td>C7 &amp; C8</td>
<td>C9 &amp; C10</td>
<td>C11 &amp; C12</td>
<td>C13 &amp; C14</td>
<td>C15 &amp; C16</td>
</tr>
<tr>
<td><strong>Act</strong></td>
<td><strong>Act1</strong></td>
<td><strong>Act2</strong></td>
<td><strong>Act3</strong></td>
<td><strong>Act4</strong></td>
<td><strong>Act5</strong></td>
<td><strong>Act6</strong></td>
<td><strong>Act7</strong></td>
<td><strong>Act8</strong></td>
</tr>
<tr>
<td><strong>Cond3</strong></td>
<td><strong>Cond4</strong></td>
<td><strong>Cond5</strong></td>
<td><strong>Cond6</strong></td>
<td><strong>Cond7</strong></td>
<td><strong>Cond8</strong></td>
<td><strong>Cond9</strong></td>
<td><strong>Cond10</strong></td>
<td><strong>Cond11</strong></td>
</tr>
<tr>
<td><strong>Act1</strong></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>Act2</strong></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>Act3</strong></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>Act4</strong></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>Act5</strong></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>Act6</strong></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>Act7</strong></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>Act8</strong></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Figure 5: an example of decision-table

**Figure 6**: Decision-Table supporting Run-Test Evaluation (compact version)
SPC Implementation. Following to the generic presentation of the decision tables, we now focus our attention on presenting, how they have been implemented in SPC context (Figure 6).

The CONDITION quadrant includes the two different control charts that must be interpreted jointly for identifying appropriate actions. The CONDITIONAL STATES quadrant contains, for each chart, the chart’s applicable tests that may fail. So each conditional state means “tests failed” or “no tests failed” in the case of a “null” value. The ACTIONS quadrant lists the set “occurred events” + “actions to carry out”. The RULES quadrant contains rules that associate each possible combination of conditional states (tests failed) to a set of appropriate actions.

3.2 SPC-InvestigationProcess

For successful software process monitoring a software process must be constantly monitored and evaluated in order to determine its stability. This allows, on one hand, to quickly react with improvement initiatives in case process performances slow down, on the other hand, to verify the validity of the improvements made. The investigation process here proposed is able to constantly monitor process performance and point out its variation either it be spontaneous or induced by improvement initiatives.

The investigation process (Figure 7) includes:

- Determination of the Measurement Object: i.e. selection of both processes to evaluate and measurable characteristics that describe process performances.
- Determination of the Reference-Set: i.e. a set of observations of the measurable characteristics that are representative of the process performances and the control limits calculated using such observations.
- Process Monitoring and Stability Evaluation: the control limits are fixed, data is tracked over time on the charts, and the tests are executed for each new plotted data point. Whenever a test fails, presence of an assignable cause is investigated.
- Run-Test Evaluation: when a run-test fails, it suggests the actions to carry out. Such activity is supported by the SPC-DecisionTable consultation (Figure 6).

4. Conclusion and Future Works

This paper has faced the three main issues that represent a barrier to software process monitoring automation. For each of these, the authors have suggested a solution that represent an important contribution for overcoming them. Also, authors propose two tools that support monitoring automation based on the 3 solutions suggested. In particular, a DecisionTable for formalizing the 3 solutions proposed; the Investigation Process for coordinating and executing the monitoring activities presented.

Nevertheless, a complete automation of software process monitoring still represents an open issue. As discussed in [25], there are many aspects related to software process measurement such as the difficulty of collecting metrics, their reliability, their selection as measurement objects that leave a lot of space for subjective management decisions that can influence the success/failure of monitoring activities. In this sense, the authors are carrying out studies on identifying the most appropriate metrics for the automated approach proposed.

Finally, in the appendix the authors illustrate an overview of a software toolkit, SPC-Toolkit, developed ad hoc to support the execution of the proposed process monitoring automation.

Appendix. Overview of SPC-Toolkit

SPC-Toolkit is a kit of software components able to support the execution of the proposed process monitoring automation (available at: http://serlab01.dl.uniba.it/SPC-Toolkit).

The architecture follows a Model-View-Controller (MVC) pattern. It has been developed using J2EE technology. In particular:

- Java Server Pages for View area: presentation logic
- Java Servlet for Control area: control logic
- Java-Beans for Model area: business logic
The toolkit is made up of three different archives:
- SPC-Engine.jar
- SPC-DecisionTable.jar
- SPC-Wizard.war

Figure 8: Architecture of SPC-Toolkit

**SPC-Engine.jar**

According to the first part of the proposed theoretical model, such archive automates the activities of our proposal. In particular, the archive is made up of two different components: **Kernel** and **GraphBuilder**.

**Kernel**. According to our proposal such component coordinates how to use the control charts, the run-tests, and their interpretations. Furthermore, it embeds the control flows described in the Investigation Process.

**GraphBuilder**. Such component builds the interactive graphs representing the SPC control charts and their failures. In particular, such graphs are implemented using the Scalable Vector Graphics (SVG) technology, that is a graphics file format and Web development language based on XML.

**SPC-DecisionTable.jar**

Such archive supports the consultation of the SPC-DecisionTable in order to interpret the SPC signals and relate them to “what really happens” within the process. Such activity implements the last step of the Investigation Process. Furthermore, there is an editor for updating the SPC-DecisionTable in order to collect and to formalize the experience acquired during its use.

**SPC-Wizard.war**

It is a web archive that implements the presentation logic for supporting the user interaction with the jar archives. In other words SPC-Wizard contains appropriate Java Server Pages and Java Servlets that, linked to the two jar files, make the whole SPC-Toolkit a web-application.

**References**


