Employing Semantically Driven Adaptation for Amalgamating Software Quality Assurance with Process Management

Gregor Grambow and Roy Oberhauser
Computer Science Dept.
Aalen University
Aalen, Germany
{gregor.grambow, roy.oberhauser}@htw-aalen.de

Manfred Reichert
Institute for Databases and Information Systems
Ulm University
Ulm, Germany
manfred.reichert@uni-ulm.de

Abstract—**Often in software development processes, tighter and more systematic integration of quality assurance techniques and measurements in the operational processes is desirable. While some processes specify abstract quality assurance measures, concrete requisite measures directly relevant for specific product artifacts (e.g., code) or processes (e.g., testing) must be determined operationally and contemporaneously, yet are hitherto often determined manually and unsystematically. By employing semantically driven adaptation of software quality assurance and software development processes, an approach is described and applied in the software engineering domain for the detection, mediation, and management of just-in-time quality measure assignments. The approach shows promise for adapting automated process enactment to enable effective and efficient quality assurance integration.**

Keywords-Adaptive Process Management, Semantic Technology, Software Quality Assurance, Process-Centered Software Engineering Environments

I. INTRODUCTION

Software quality assurance (SQA) is often viewed as an area incurring additional costs and delays in delivery of a software project. As cost, functionality, quality, and delivery pressures on product development projects continue, software development processes and quality assurance should be examined for opportunities in the areas of greater automatability and cost reduction. Automation does not only reduce labor costs, but can also systematically improve product and process quality by means of repeatability, traceability, and consistency.

However, automation also raises challenges. Especially the software development domain manifests these due to its dynamic flux in technologies, tools, and platforms that often result in one-off projectized software engineering environments. Process-Centered Software Engineering Environments (PCSEEs) [11] model and execute processes that coordinate human activities and software development tools. Due to the significant impact of SQA on project costs [2], the cost-effectiveness of concrete quality assurance measures (i.e. actions) is essential, and should be based on contextually relevant SEE data for economical adaptation to changing environments and constraints. The timely and harmonious assignment of SQA measures requires their tight integration in software development processes, specifically concrete workflows that must necessarily be adapted since the exact measures are not foreknown but depend on the given context. Due to time constraints and efficiency, only a limited number of all possible quality measures can be applied, while the effectiveness and efficiency of a specific quality measure depends, among other factors, on the applicability of the measure, the project timing, worker competency, and correct fulfillment [2].

One promising area for increasing the degree of automation in PCSEEs is the context-sensitive triggering and selection of quality measures (e.g., refactoring) while taking into account people, processes, tools, artifacts, and interactions. It would include the analysis of quality metrics and measure selection for process and product artifacts to support software engineers via automated measure suggestion at the appropriate time. Moreover, SQA is proportional relative to overall project size [1], while the amount of effort allocated to SQA should be front-loaded (up to 25% of development effort) and be reduced later [2][30]. Manual enforcement of such lifecycle-time-dependent SQA overhead can be laborious and detract from more pressing project management issues.

Software engineers currently lack automated guidance for software quality assurance that can practically apply SQA cost-effectively using SQA models of development activities such as [2]. A solution is required that can analyze the project context on the fly, prioritize SQA measures, and adaptively assign these to the relevant resources at appropriate points in their process. Furthermore, multiuser measures that require preparation (e.g., code inspections) should be contextually coordinated, meaning their activities included at the appropriate point in the user's process. SQA measures should be systematically distributed, and quality measures should be filtered based on the available time constraints. This should be done without exceeding SQA expenditure limits while taking advantage of quality opportunities such as early activity completion where, for example, a suitable measure can be applied without impacting the schedule. This necessitates adaptive processes that can cope with the inclusion of unforeseen activities.

This solution contributes to a holistic PCSEE integration of SQA and SPM (software process management), comprising the detection of problems, the automated strategic adaptation of measures in conjunction with the Goal Question Metric technique (GQM) [3], context-based selection of SQA measures, leveraging quality opportunities, and adapting concrete process instances. Measures vary in their effectiveness (i.e. utility) and appropriateness, and measure assignment has to consider these factors.

The remainder of this paper is organized as follows: the solution approach is described in Section II and Section III elucidates the implementation thereof. The evaluation in Section IV is followed by a discussion of related work in Section V, with Section VI providing a conclusion.

II. Solution Approach

The goal of timely assigning quality measures is addressed utilizing a combination of various technologies for detection, mediation, and management, which will now be described. The infrastructure for these technologies is provided by the CoSEEEK framework (Context-aware Software Engineering Environment Event-driven frameworK) [22]. Figure 1 illustrates the conceptual architecture for the current approach.

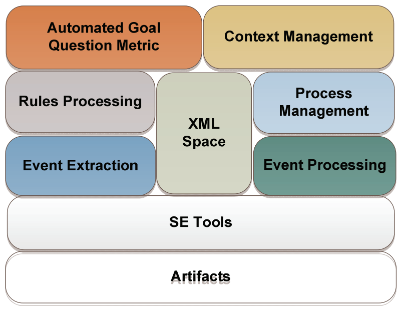

Figure 1. Conceptual Architecture

The architecture of the framework comprises the following components: *Artifacts* is a placeholder for various artifacts that can be processed in a software project, e.g., source code or test code. These artifacts are processed using heterogeneous *SE tools* (e.g., tools for static analysis). Communication and data storage for the event-based loose coupling architecture of the framework are provided by an *XML Space*. To enable integration of diverse external events from SE tools, the *Event Extraction* module is utilized. The *Event Processing* module, in turn, enriches these atomic events with more semantic value through contextual annotation and aggregation of the events based on complex event processing (CEP). A rules engine contained in the *Rules Processing* module automatically analyzes reports from tools for both static and dynamic code analysis, triggering events as necessary. The *Process Management* module applies PAIS (Process-Aware Information System) technology for the automated support and governance of the software development process. Software development processes (e.g., OpenUP) are concretely implemented as workflows in this module. The AGQM (Automated Goal Question Metric) module extends and automates the GQM technique by using a multi agent system (MAS) with behavior agents [10]. Finally, the *Context Management* module enables the aggregation of all information from different sources and thereby the creation of a holistic project context where an adaptive integration of the measures from [10] into the developer's concrete process becomes feasible.

The process of timely assigning apposite quality measures involves several steps as depicted in Figure 2. These are separated into four categories: *Quality Opportunity Detection* detects user availability for quality measures (so called Q-Slots). For this purpose, the estimated and actual durations of planned activities are considered. *Problem Detection / Measure Proposal* comprises activities for automatic detection of problems relating to source code and for the proposal of related quality measures in alignment with higher-level goals utilizing the GQM technique. To facilitate the automated applicability of measures, *Measure Tailoring* comprises three steps for the context-based selection of the measure and its insertion point (so called *extension points* of a workflow) as well as the adaptation of the concrete user's workflow. The procedure is completed by an assessment phase of the applied measures that impacts future measure selection processes, adapting the system automatically while still allowing human-based tuning.

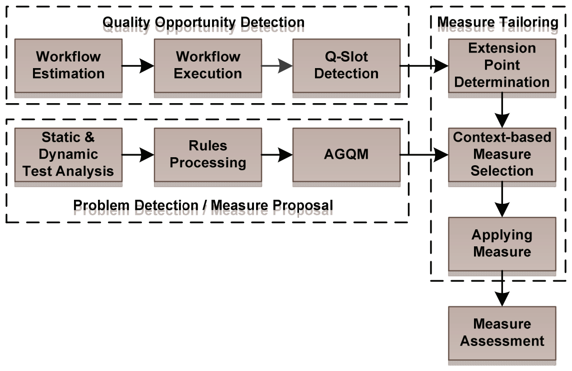
Quality Opportunity Detection
Workflow Estimation
Workflow Execution
Q-Slot Detection
Measure Tailoring
Extension Point Determination
Static & Dynamic Test Analysis
Rules Processing
AGQM
Context-based Measure Selection
Problem Detection / Measure Proposal
Applying Measure
Measure Assessment

Figure 2. Conceptual Process

A. *Quality opportunity detection*

Workflow creation, instantiation, and execution are conducted utilizing the *Context Management* as well as the *Process Management* module. The workflow templates and instances are governed by a PAIS, whereas the *Context Management* module contains semantic enhancements to the workflow concepts. These enhancements include temporal properties, making it possible to estimate durations of activities and match them later with actual execution times. To enable these calculations, an estimation of all durations of user activities (so called assignments) is required prior to workflow instantiation. Other constraints such as planned start date or activity dependencies can also be taken into account. Secondly, the specification of a quality overhead factor and a quality function are required. Via the quality overhead factor, it becomes possible to specify that a certain percentage of work done in a project should be subjected to quality measures. The quality function, in turn, enables control over the timely distribution of that quality effort, e.g., more up-front effort can be allocated to quality measures to reap the benefits described in [2][30]. When a workflow is started, each activity involving a user has a planned duration. Its completion then triggers the recording of the actual execution time. To facilitate the selection of quality measures corresponding to the current situation, the measures are stored in the *Context Module* as well*,* where they are enhanced with additional properties. These properties comprise an estimation of the duration of a measure and an association of the measure to a certain level of abstraction, meaning a certain workflow such as a project iteration, project phase, or concrete workflow.

Based on these properties, a calculation is applied that reveals *Q-Slots,* i.e. available opportunities for quality measures. During workflow execution, this calculation is conducted each time an activity completes. To determine the actual execution times of the abstract assignments they are connected to concrete tasks a user performs (so-called assignment activities) as depicted in Figure 3. In that example, the assignment activities ‘Ac1-Ac5’ relate to assignment ‘A2’ and the assignment activities ‘Ac6-Ac8’ to assignment ‘A3’. The measure proposal process can be started in two possible situations:

1. If activities finish earlier than expected, it is possible to insert a quality measure without delaying forthcoming activities. Therefore, the estimated durations of all processed activities are compared to their actual execution times.
2. If a quality overhead is specified and the quality effort of the current user is below the computed percentage, the insertion of a quality measure activity is also feasible.

B. Problem detection / measure proposal

Quality measures can be of proactive or reactive nature. To be able to assign reactive measures, an awareness of concrete problems is required. Many problems relating to source code can be detected automatically via static code analysis tools and dynamic test analysis. The *Event Extraction* module enables the CoSEEEK framework to be aware of the execution of such tools, and thus to automatically process reports generated by them. These reports contain information about metrics that were applied to source code. The *Rule Processing* module then processes these reports producing enhanced reports that contain violations to predefined thresholds according to the metrics. Furthermore, the reports include information about the artifacts in which the violations occur and about concrete measures the *Rule Processing* module assigned to them. These reports unify and abstract the reports obtained from all integrated analysis tools, containing reactive measures without order or priority. The *AGQM* module is then responsible for measure selection. Violations of metrics in certain artifacts are also recorded in the *Context Management* module. This enables the aforementioned selection of an *extension point* based on problems relating to artifacts processed by a user’s future activities.

The *AGQM* module manages both reactive and proactive (or preventative) measures. It utilizes a multi-agent system (MAS) that selects and prioritizes measures by extending and automating the GQM technique as described in [10]. The latter enables the module to analyze the concrete measures in conjunction with abstract project goals, thus facilitating a strategic measure selection. Reactive measures are prioritized in a cooperative bidding process among the agents, whereas proactive measures are proposed using a competitive bidding process. The module has been adapted to the current scenario to output an ordered list comprising reactive as well as proactive measures.

C. Measure tailoring

To achieve a high degree of utility for quality measures proposed by the system, their applicability to the current situation is essential. The CoSEEEK *Context Management* Module enables the accumulation and utilization of contextual information from different sources (e.g., the type of the measure or the skill level of the user) to ensure measure applicability. Via semantic enhancements, CoSEEEK enables the process engineer to specify certain *extension points* in the workflow where the application of quality measures is feasible. Examples include the end of an iteration or a phase in a project. Figure 3 illustrates this: It contains workflows on three different levels. *Extension point* ‘E1’ is attributed to a node of the ‘Phase’ workflow that, in turn, contains an ‘Iteration’ workflow. A quality activity can thus be inserted after the completion of that ‘Iteration’ workflow. Various properties can be included that act as a filter for the types of measures applicable at a certain point.

The *extension point* determination process for a concrete workflow instance is started when an activity is completed and either enough spare time for applying a quality measure exists or the quality effort has not exceeded the specified quality overhead yet. The system searches future assignments of the current user for *extension points* and makes a selection based on different properties. For instance, if the user works on a source code artifact for which the system has detected a problem concerning code quality, an *extension point* at that activity can be selected. Thus, two changes to that artifact can be applied whereas the process of checking out, testing, and checking in the artifact is performed only once. This scenario is illustrated in Figure 3. When the user finishes assignment ‘A2’ and there is the opportunity for a quality activity, the system checks future assignments, i.e., ‘A3’ in the given case. That assignment has an attributed sub-workflow, which contains the extension point ‘E3’, which in turn, is connected to the concrete assignment activity ‘Ac7’. That activity involves processing of an artifact for which a problem was detected.

To enable better distribution of quality measures over time and avoid the consideration of quality aspects only at the end of the project, each *extension point* is weighted according to its temporal proximity to current time (*imminence*). If no other constraints apply, the next upcoming extension point is selected.

Since the main purpose of the *AGQM* module is to select, propose, and weight measures in alignment to the goals of the project, an additional selection process is necessary to tailor the proposed measure to the current Q-Slot. This is done utilizing the semantic properties of measures contained in the *Context Module*. These properties include the abstraction level to which a measure applies and a time category. The abstraction level facilitates the selection of measures matching the workflow abstraction level, which is related to the selected extension point. The time category enables the selection of measures that consume approximately as much time as the current Q-Slot provides.

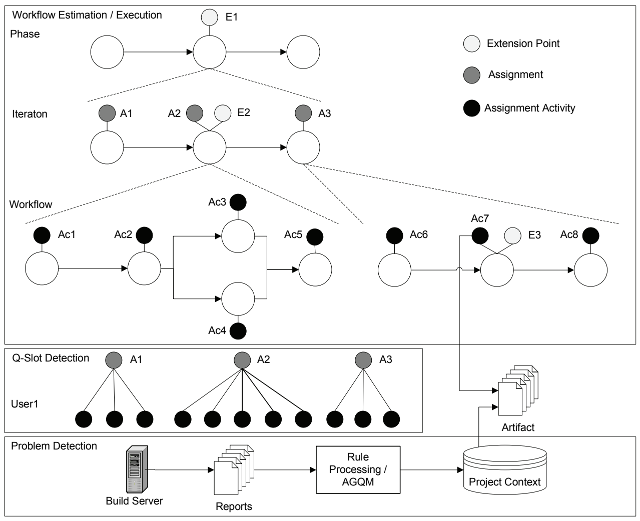

Figure 3. Workflow Enhancements / Q-Slot / Problem Detection

When the Q-Slot is detected, the *extension point* is selected and the appropriate measure is chosen. Further, an activity is generated and dynamically inserted in the workflow of the related user.

D. *Measure Assessment*

To further enhance usefulness of the applied measures, an assessment phase takes place after the successful application of the measures. In that phase, the measures are rated due to their impact on the qualities of the produced code. This is done using KPIs (key performance indicators), composite metrics that are continuously calculated by the *AGQM* module. The calculation is conducted each time a report of a testing or analysis tool is received by the system. The KPIs, their values, and the time of their computation are then stored in the *Context Management* module. Based on these values, the measure assessment process is carried out at certain points of time in the project, which can be specified a priori (e.g., at the end of a project or a project phase). The process uses the development of the KPIs over time and the times at which the Q-Slots took place to assess whether or not the measures had an impact on the KPIs. The measures have a usefulness (utility) property, indicating their impact on the KPIs, that is written by the assessment process and used in the measure selection phase.

III. IMPLEMENTATION

This section describes the technical realization of the different components and concretizes the concept.

A. *Technical realization*

The loosely-coupled event-based communication infrastructure for the different modules is provided by a Apache CXF web-service-based implementation of the tuple space paradigm [9] with the eXist XML database [20] for event storage. To facilitate automated problem detection and processing, a combination of the *Event Extraction* module, the *Event Processing* module, and the *Rules Processing* module is utilized. Event extraction is done via the Hackystat framework [13], which provides sensors for various tools. In the current scenario, events from static code analysis tools (e.g., PMD [6]) are used. Atomic events are aggregated using the CEP tool Esper [8]. These events and the reports from the static analysis tools form the input for the *Rules Processing* module that utilizes the rules engine JBoss Drools. Furthermore, that module features a web GUI enabling users to specify code quality measures and relate them to code quality problems. By means of these relations, the module then creates the aforementioned unified reports containing currently violated metrics and attributed measures. Since these are contextually unordered and there can be many violations, the measures are weighted by the *AQGM* module to enable automated measure assignment. That module is implemented using the JADE multi-agent system [4].

Q-Slot detection, context-aware tailoring of measures, and the concrete integration into users' workflows are managed by the *Context Management* and the *Process Management* modules. The Context Management module is realized by an OWL-DL (Web Ontology Language Description Logic) ontology and the semantic reasoner Bossam [12]. The *Process Management* module is based on the AristaFlow BPM Suite [26] for adaptive process support that originated from ADEPT research [7][27]. Due to its 'correctness by construction' principle, AristaFlow supports the efficient and correct modeling, adaptation, and deployment of processes. Its capabilities for ad-hoc workflow changes during runtime, in conjunction with instantaneous checking of correctness, not only enables the safe adaptation of workflow instances, but also offers a high degree of freedom in modeling executable workflows. Workflows allow a process engineer to work with and visualize familiar activity sequences, thus enabling better model understandability, checking, and transfer versus, e.g., a pure ontology (non-workflow) solution. Despite AristaFlow's comprehensive support for dynamic workflow changes, automated workflow adaptation has hitherto not been considered.

B. *Q-Slot detection*

Figure 4 shows the concepts in the ontology that enable Q-Slot detection, including one concrete instance (in red) for each class (in black).

The semantic enhancements to workflows are modeled by the concept of the *WorkUnitContainer* containing *WorkUnits* for the activities of a workflow. Since *WorkUnits* and *WorkUnitContainers* respectively do not necessarily have to be related to users, human tasks are modeled separately in the concept of the *AssignmentActivity* that is connected to a *Person*. To allow human estimation of activity durations prior to workflow execution, the concept of the *Assignment* is introduced. *Assignments* have properties for planned duration as well as for actual execution time. They can have multiple attributed *AssignmentActivities*. An example for an *Assignment* would be 'Develop Feature x' with *AssignmentActivities* like 'Write code' or 'Write Developer Test'. Since the *Assignments* and the *AssignmentActivities* are connected, the actual duration for the *Assignment* can be determined utilizing the durations of the *AssignmentActivities*. Thus, actual spare times for a certain user can be detected by a SPARQL [25] query (which is executed utilizing the Jena API [19]) returning all *Assignments* for that *Person* to compare the planed durations with the actual durations. The percentile quality effort of a user can thus also be determined, comparing the durations of finished *Assignments* with those of processed *Q-Slots*. Using the actual assignment spare time or the available quality overhead limit, if the smallest time category for a measure fits, a *Q-Slot* is generated without an assigned *Measure*. Should another *Assignment* complete before the *Q-Slot* can be inserted, the calculation for the *Q-Slot* is repeated, updating or deleting the *Q-Slot* dependent on available time. Thus, multi-user measures also become possible. If a user generates a *Q-Slot* and another user has already generated a *Q-Slot* but not yet started it, both *Q-Slots* will be connected and multi-user measures be additionally taken into account. The system regards activities such as code inspections as multi-user measures. While these meetings take place out of the scope of the system as part of the users' unestimated activities, any related preparation activities are assigned by the system to the users taking part in a multi-user measure. The scenario section will exemplify the selection of a multi-user measure.

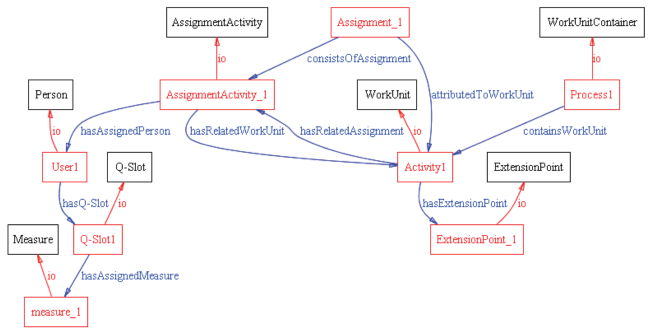

Figure 4. Ontology concepts for Q-Slot detection

C. Extension point determination

To enable context-based measure selection, possible future *ExtensionPoints* for a user must first be determined. As depicted in Figure 4 *ExtensionPoints* are connected to *WorkUnits* meaning that the *Measure* of a *Q-Slot* can take place after the *WorkUnit* that is connected to the chosen *ExtensionPoint*. To determine upcoming *ExtensionPoints*, the system executes the algorithm depicted in Listing 1, which takes an ordered list of future *Assignments* as input.

Listing 1. Extension Point Determination

```
For assignments
  getRelatedWorkUnit()
  checkForSubExtensionPoints(workUnit)
  checkForSuperExtensionPoints(workUnit)

checkForSubExtensionPoints(workUnit)
  check for extension point and add to list
  if(workUnit has subWorkUnitContainer)
    for each contained workUnit
      checkForSubExtensionPoints(workUnit)

checkForSuperExtensionPoints(workUnit)
  if(finalWorkUnit in WorkUnitContainer)
    getSuperWorkUnit
    check for extension point and add to list
    checkForSuperExtensionPoints(workUnit)
```

Since the *Assignment* can be connected to the *WorkUnit* at any level of abstraction, there can be *WorkUnits* on a level above and below it. The example depicted in Figure 3 illustrates that fact. If there was time for a measure after completion of *Assignment* 'A1', *ExtensionPoints* 'E1-E3' would be available, where 'E1' is one level above and 'E3' is one level below the *Assignments*. For each *Assignment*, the algorithm first checks recursively whether there are *ExtensionPoints* below the current *WorkUnit*. Subsequently, it recursively checks whether the current *WorkUnit* is the final one in its *WorkUnitContainer*. For this case, it checks the superordinate *WorkUnit* to which the *WorkUnitContainer* belongs for an *ExtensionPoint*. The output of the algorithm is a timely ordered list of *ExtensionPoints*.

D. Context-based measure selection

The context-based measure selection and the final *ExtensionPoint* selection both depend on a weighting of the measures. Prior to that, however, for each *ExtensionPoint,* a query is executed returning the *Measure* with the highest position in the list of the *AQGM* module that matches the properties of the *ExtensionPoint* and the *Q-Slot*. Listing 2 shows the SPARQL query using all of these properties.

Listing 2. Measure preselection

```
PREFIX project:
<http://www.htw-aalen.de/coseeek/context.owl#>

SELECT ?measure
WHERE
 {?list project:containsMeasure  ?measure;
        project:currentList   "1".
  ?measure project:timeCategory  "2" ;
           project:forNumberOfUsers  "1" ;
           project:hasMeasureType ?measureType .
  ?meaureType project:title "MeasureType_1"
  ?measure project:forAbstractionLevel
           ?abstractionLevel.
  ?abstractionLevel project:title
                    "AbstractionLevel_1".
  }
LIMIT 1
```

The properties are the abstraction level, applicable measure type, and user skill level for the *ExtensionPoint* (which do not all have to be set except the abstraction level) together with the time category and number of users for the *Q-Slot*.

The weighting depends on different factors that are all considered: first, the imminence of the *ExtensionPoint* (Imminence i: $0 < i <= 1$; initial value 0.9, for each following *ExtensionPoint* 90% of predecessor) is considered. Second is the strategic alignment sa of the *Measure* (i.e., the position in the ordered list of the *AGQM* module). Third is the utility of the *Measure* (Measure Utility mu: $0.5 < mu < 1.5$; initialized with 1) and defined in Formula (2), which is always updated in the measure assessment process. The fourth factor depends on the measure type and permits weighting of certain measure types (Measure Type Factor mtf: $0.5 < mtf < 1.5$; predefined, standard value 1).

An example for this is the measure type 'code', which denotes measures related to source code problems, and thus only can be applied if future activities of the current user involve artifacts for which problems have been detected (such as the activity 'Ac7' in Figure 3). These measures improve efficiency since they can be applied right after scheduled changes to the associated artifacts, avoiding additional overhead for checking out / in and testing. After this weighting procedure, the *ExtensionPoint* with the *Measure* having the highest weight is chosen for insertion. The calculation of the Measure Weight mw is shown in Formula (1). For the calculation of the imminence, the index for each upcoming *ExtensionPoints* is n, stating with 0 for the next *ExtensionPoint*. The strategic alignment is computed via the number of measures in the *AGQM* list (listItems) and the position of each measure in the list (listPosition).

$$\begin{aligned} i &= 0.9^{n} \\ sa &= \left(1 - \frac{listPosition}{listItems}\right) \qquad (1) \\ mw &= i * sa * mu * mtf \end{aligned}$$

E. *Applying measures*

After all parameters have been determined, the point in the workflow where the activity insertion process shall be started is stored in the ontology as parameter of the *WorkUnit*. The activity is not inserted immediately because if the insertion point lies some time ahead, it is possible that not enough time is left for the measure due to delayed activities processed in the meantime. When the point is reached and there is sufficient time, two possibilities for measure integration exist: initiation of a new workflow instance or direct integration of one or multiple activities in the current workflow. Figure 5 illustrates the insertion of an activity into a running workflow instance using AristaFlow where the green node represents the current activity of the user.

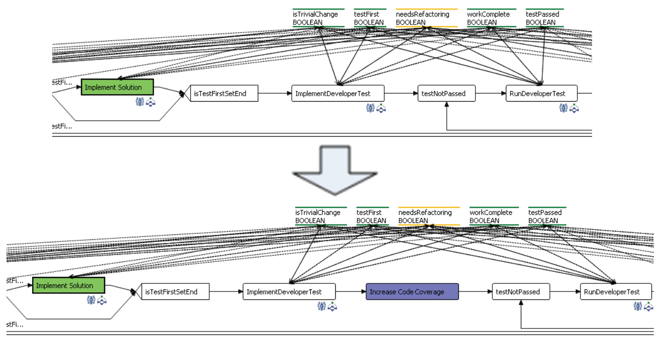

Figure 5. Activity Insertion

The automatic insertion process utilizes the adaptability features of AristaFlow. This allows for seamless integration of the activity into the workflow, while not distracting the current user from his work. AristaFlow also guarantees structural correctness before and after a dynamic insertion.

F. *Measure Assessment*

After a measure is applied, related information is stored in the ontology via the *Q-Slot*, which records the time and duration as well as the applied measure and the assigned person. The timepoint of the measure execution is then associated to the impact on KPI trends for utility analysis. Figure 6 depicts a subset of the concepts in the ontology and some concrete instances illustrating the connections between the concepts.

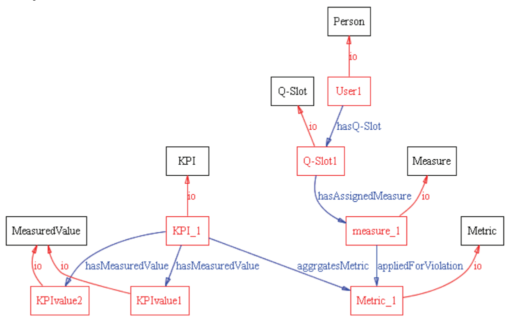

Figure 6. Measure Utility Assessment

KPIs are calculated continuously by the *AGQM* module. Each *KPI* has multiple *MeasuredValues* that record the value of one *KPI* calculation at a certain point in time. The assessment procedure works as follows: from the point of time when a measure is applied, up to 10 *MeasuredValues* of the *KPIs* are selected for trend analysis using *KPI* deltas. Since a multitude of factors can influence evolution of the *KPIs,* and since the *Measures* can be of a diverse nature, all 10 values of each *KPI* are used with decreasing influence. The skill level of the person applying the measure is also taken into account, reflecting the higher probability of ineffectively applying a measure by less skilled persons (Skill Factor sf: $0 < \text{sf} < 1$; predefined, 1 for highest skill level). The equation for calculating mu is shown in Formula (2) where n is the index for the *KPI* deltas starting with 0 for the first delta.

$$mu = mu * \left(1 + \frac{\sum_0^9 kpi\Delta_n * \left(1 - \frac{n}{10}\right)}{10} \right) * sf \qquad (2)$$

G. *Modeling Effort*

To keep the modeling effort reasonable, some functions and standardized definitions are provided. The semantic enhancements to process management (*WorkUnitContainers*, *WorkUnits*, *AssignmentActivities*) are generated automatically from the workflow templates of the *Process Management* module (i.e., AristaFlow). Thus, only the level of the *Assignments* has to be explicitly defined or can be imported from an external work breakdown structure. The connections between the *AssignmentActivities* and the *Assignments* are then automatically established by the system.

A set of standard quality measures can be provided including parameters for common static analysis tools such as PMD or FindBugs. Therefore, the quality manager only has to define the *ExtensionPoints* for the processes and can adapt the values for certain *Measures* if needed. The data provided also includes a standard set of *KPIs* used for *Measure* assessment and for the *AGQM* module, which can also be adjusted.

IV. EVALUATION

This section covers scalability and performance measurements and a concrete laboratory scenario exemplifying the application of the proposed approach. In planned future work, the validity of the approach will be additionally evaluated in multiple longer-term software production case studies.

A. *Scenario*

To illustrate the application of the concept, the OpenUP [23] software development process was chosen with the scenario focusing on the Construction Phase. The 'Develop Solution Increment' process was planned. Figure 7 illustrates the configuration for this scenario.

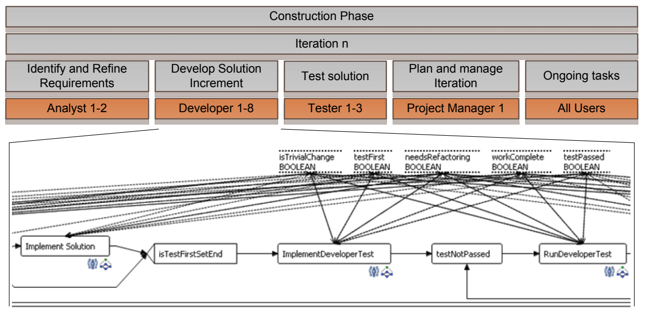

Figure 7. Scenario Configuration

There are four different scheduled and estimated activities *(Assignments*) that are performed by different project members plus activity 'Ongoing Tasks' that represents the base load of each user (i.e., activities which are not part of the project schedule). The scheduled activities are all based on workflows. A snippet of the workflow for the 'Develop Solution Increment' activity is shown in Figure 7 that contains multiple *AssignmentActivities* for the developer such as 'Implement Solution', 'Implement Developer Test', or 'Integrate and Build'.

For the construction iteration, four *ExtensionPoints* have been defined. One takes place after the 'Implement Solution' *Assignment,* having the most concrete abstraction level '0' and the measure type 'code' (referred to as 'Code' in the following). The second takes place after the 'Implement Developer Test' *Assignment* with measure type 'test' ('Test'). Both *ExtensionPoints* are directly related to the processed source or test code the user currently works on and are thus only taken into account if measures relating to the source or test code processed by that activities have been proposed by the *AGQM* module. The third *ExtensionPoint* is attributed to the 'Develop Solution Increment' activity enabling quality measures between the estimated activities ('Activity'). The last *ExtensionPoint is* assigned to the iteration and used for quality measures that only fit at the end of an iteration ('Iteration').

In the current scenario, eight simulated developers are involved, each of them having eight estimated 'Develop Solution Increment' assignments. For simplicity, the scenario only relies on execution time deviation and no specified quality overhead to enable *Q-Slots*. Table I shows the actual execution time deviations from what was estimated, where positive values indicate that an activity took longer than estimated, negative values indicate shorter actual execution times, and grey boxes indicate *Assignments* where after the measure proposal process is started for the respective developer. Since the *Q-Slot* detection only relies on the execution time deviations, it is independent of the initial duration of the *Assignments* or the work hours per day. To keep the current scenario simple, it is assumed that all *Assignments* take one workday. As the table shows, some *Assignments* take longer, while others finish earlier. For this scenario, the quality measures that consume the least time take two hours, which makes quality measures possible for developers 2, 3, 5, and 7.

Table I. Assignment Duration Deviations to Plan (hours)

Developer	**Assignment**							
	1	***2***	***3***	***4***	***5***	***6***	***7***	***8***
dev1	0	1	-1	-1	0	1	0	1
dev2	1	0	2	0	0	-1	1	1
dev3	1	-1	1	0	2	0	0	0
dev4	0	1	0	0	-2	2	0	-1
dev5	1	0	-1	0	1	2	0	0
dev6	-4	1	1	1	0	0	0	-1
dev7	0	1	0	0	0	0	2	0
dev8	1	0	0	0	-1	-2	1	1

Table II illustrates future *ExtensionPoints* and their values for each user at the point of time where the measure proposal is started for the first time whereas the shaded cells show which *ExtensionPoint* was selected due to the highest weight. For user ‘dev2’, this occurs at the end of Activity 3 where measure ‘m1’ is selected. The value of 0.7 is computed from the following values: i = 1, mu = 1, mtf = 1 and sa = 0.7 (the measure being at position 30 in a list of 100 proposed measures from the *AGQM* module). All other ‘Activity’ *ExtensionPoints* have the same values except for Imminence, which decreases for each additional activity. For the ‘Iteration’ *ExtensionPoint,* the same values apply except that it has a Strategic Alignment of 0.99 and a Type Factor of 1.2. For user ‘dev2’, it is assumed that it was detected that (s)he will process a source code artifact for which a problem has been detected as part of Activity 4. Therefore, measure ‘m2’ is chosen for the ‘Code’ *ExtensionPoint* of Activity 4. The measure has a Strategic Alignment of 0.75 and a higher Type Factor of 1.4. It therefore is selected for the user.

For user ‘dev3’, the calculation starts after Activity 5. Since measures ‘m1’ and ‘m3’ have not been proposed yet, they are still proposed here due to the same parameters. Thus, ‘m3’ at the end of the iteration has the higher weighting and is planned for that user.

Table II. ExtensionPoint properties

Dev	Prop.								
dev2	type	A	A	C	A	A	A	A	I
	a	3	4	4	5	6	7	8	
	m	m1	m1	m2	m1	m1	m1	m1	m3
	w	0.7	0.63	0.99	0.52	0.46	0.41	0.37	0.48
dev3	type	A	A	A	A	I			
	a	5	6	7	8				
	m	m1	m1	m1	m1	m3			
	w	0.7	0.63	0.56	0.52	0.71			
dev5	type	A	A	A	I				
	a	6	7	8					
	m	m4	m4	m4	m3				
	w	0.85	0.77	0.69	0.83				
dev7	type	A	A	I					
	a	7	8						
	m	m1	m1	m3					
	w	0.7	0.63	0.79					

A=Activity, C=Code, I=Iteration, a=assignment, m=measure, w=weight, m1=analyze resource usage, m2=refactor code, m3= verify documentation, m4=code inspection preparation

When the calculation for user ‘dev5’ starts at the end of Activity 6, there is also the Q-Slot of user ‘dev3’ that has been planned but not used yet. Therefore, two connected *Q-Slots* are available, causing measure ‘m4’ to be proposed, which is assumed to be a multi-user measure and has a higher Strategic Alignment of 0.85. Because of the higher weight, measure ‘m4’ is then integrated for users ‘dev3’ and ‘dev5’. Therefore, when the calculation starts for user ‘dev7’ at the end of Activity 7, measure ‘m3’ for the ‘Iteration’ *ExtensionPoint* is still available and is used for that user having the highest weight now.

At the end of the iteration, after all measures have been applied successfully, the assessment process is carried out by the system. This process relies on the development of the KPIs that, in turn, rely on received reports from analysis tools. These reports are generated as part of a nightly build process that builds the code, executes all tests, and applies all metrics to the code in the current scenario. Thus, there are ten points in this scenario where the KPIs are calculated, each taking place after one of the estimated assignments has completed. Table III illustrates the development of the related KPIs. KPIs are computed by the AGQM module to have a uniform value that lies between 0 (extremely bad) and 1 (perfect).

Table III. KPI development

Measure	KPI	1	2	3	4	5	6	7	8
m2	kpi2	0.5	0.46	0.45	0.6	0.6	0.6	0.62	0.62
m4	kpi4	0.6	0.62	0.6	0.58	0.59	0.6	0.67	0.74
m3	kpi3	0.4	0.43	0.42	0.45	0.46	0.46	0.46	0.6

Since this scenario only covers one iteration with no activities ahead of this iteration, not all ten time points were available for the measure assessment procedure. *Measure* ‘m2’ was applied at the end of time point 4 but before KPI calculation. Therefore, the first reports of the analysis tools that reflect the impact of that measure are generated at time point 4. Thus, the first delta of the related KPI that is of interest is from time point 3 to time point 4, which is a positive change of 0.15. For Measure ‘m4’, the first delta used is from time point 5 to 6 and for Measure ‘m3’ from time point 7 to 8. Since all measures were initialized with a value of 1 for Measure Utility, applying the calculation depicted in Formula (2), the new values for the measures are as follows: 1.1662 for Measure ‘m2’, 1.154 for ‘m4’, and 1.14 for ‘m3’. The calculations show meaningful values for the synthetic scenario, which does not necessarily mean that the same applies for real world scenarios. Therefore real case studies that will be conducted in the future will not only be used to evaluate the results but also to further refine the calculations.

B. *Measurements*

Initial performance evaluations were used to assess preliminary sufficiency of the approach for practical use in the field. The critical areas selected are the context-based measure selection and extension point determination in the *Context Management* module and the concrete insertion of activities into a running workflow using AristaFlow in the *Process Management* module. For performance testing, the test system consisted of an AMD dual core Opteron 2.4 GHz processor, 3.2GB RAM, Windows XP Pro SP3, and Java Runtime Environment 1.5.0_20. AristaFlow was used in version 1.0.0beta - r73, its server was running externally in a virtual machine (VM) infrastructure of the university (cluster with VMware ESX server 4.0, 1 GB RAM was allocated for the VM, dynamic CPU power allocation). All measurements were executed five times consecutively using the average of the last three measurements.

The extension point determination and the context based measure selection are conducted together as one part of the whole process. They provide the functionality of the context module. As depicted in Table IV, the measurements were conducted with different numbers of involved *ExtensionPoints*, *Assignments,* and *Measures*. For these measurements, a context was created such that for each measurement run, the number of all involved concepts remained constant.

Table IV. Context module latencies

Total number of involved Assignments, ExtensionPoints, Measures	ExtensionPoint Determination (sec)	Measure Selection (sec)
5	4	13
25	19	63
50	38	125

Since the activities inserted in a concrete workflow can consist of multiple nodes, the latency for inserting different numbers of nodes was measured as shown in Table V.

Table V. Process module latency

Number of inserted activities	Time (ms)
5	1052
25	1453
50	2203

The measurement results show acceptable scalability for the CoSEEEK approach, as can be seen from the approximately linear increase in computation times.

V. Related Work

In the area of automation and support for process adaptability, one approach that considers support of users applying ad hoc changes is ProCycle [32]. ProCycle applies case-base reasoning techniques to assist end users in the re-use of process instance changes that were applied in a similar problem context in the past. This way, the quality of process changes can be increased and process learning be fostered. Caramba [33] features support for ad hoc workflows utilizing connections between different artifacts, resources, and processes to provide coordination of virtual teams. The approach presented in [21] utilizes a combination of agent technology and process management technology to enable automatic process adaptations. These adaptations are used to cope with exceptions in the process at runtime. These approaches do not utilize semantic web technology and do not incorporate a holistic project-context unifying knowledge from various project areas as CoSEEEK does.

In related work, several approaches combining semantic and process management technology exist. In [18] a semantic business process repository is presented to automate the business process lifecycle. It features checking in and out as well as locking capabilities and options for simple querying and complex reasoning. An ontology for business process analysis was developed in [24] for improving process compliance analysis for standards or laws like Sarbanes-Oxley act. In [31] the combination of semantic and agent technology is proposed to monitor business processes, yielding an effective method for managing and evaluating business processes. The approach described in [17] aims at facilitating process models across various model representations and languages. It features multiple levels of semantic annotations as the meta-model annotation, the model content annotation, and the model profile annotation as well as a process template modeling language. All of these approaches utilize semantic technology to facilitate the integration of process management into projects. In contrast to this, the CoSEEEK approach does not only use semantic technology to integrate different project areas, but also fosters the automatic and suitable insertion of quality measures into the workflows of project participants.

Various approaches provide integration of the GQM technique into a project. The ISMS (Intelligent Software Measurement System) [5][14] uses different groups of agents for user assistance and determination of different parts of the GQM plan. In [15] a tool was developed that allows the creation of GQM plans using predefined forms as well as the verification of the structural consistency of the plan and the reuse of its components. Furthermore, the tool supports data interpretation and analysis through aggregation of collected data. The approach presented in [28][29] also utilizes agents, for the requirements process of the SW-CMM (Software Capability Maturity Model) model. The focus is the measurement and analysis of software processes using agents and fuzzy logic. cGQM [16] uses the Hackystat framework [13] for GQM, applying continuous measurement with short feedback loops. In contrast to the aforementioned approaches, CoSEEEK focuses on utilizing context knowledge for the automatic adaptation of quality measures based on the GQM technique as well as the adaptation of running workflows for automated concrete assignment of these measures to users.

VI. Conclusion

Due to its dynamic nature and its adolescence as a discipline, the software engineering domain manifests various challenges for adaptable process and quality automation. Hitherto, development process models have typically remained abstract and not been used for automated executable workflow guidance, while SQA provided abstract quality guidance that relied to a great extent on human triggering, analysis, and concretization of activities. Ideally, opportunities for SQA should be leveraged and applied at the appropriate time with minimal disruption for developers to minimize overhead. Integrating techniques and sensors in the software engineering environment for the operational detection of SQA problems would provide a basis for a context-sensitive mediation of SQA measures and adaptation of automated development process guidance.

CoSEEEK contributes an approach implying semantically-driven adaptation utilizing process management with contextual awareness to diminish the aforementioned challenges. Using sensors and metrics for detecting SQA problems, an extended agent-based GQM technique mediates SQA measures according to KPIs. Opportunities for SQA activities are contextually analyzed, automated measure assignment is adapted, and multi-user measures are coordinated. Adaptable process management enables the system to cope with the dynamicity inherent to SE projects while semantic technology enables contextual adaptation. These result in tighter and timely integration of concretized SQA in the process, improving SQA via the automated prioritization and just-in-time assignment, applying SQA consistently, and reducing SQA overhead by reducing context switches, improving effectiveness by weighing appropriateness (e.g., competencies [2]) and measure utility, and detecting and leveraging commonality opportunities (such as artifacts or multi-user). Additionally, the distribution of SQA effort can be adjusted (e.g., frontloading as propagated in [2][30]) and monitored, avoiding issues such as too little too late or overemphasis with the associated negative impact on profitability.

Future work will include case studies in industrial settings, which will be used to assess the applicability and effectiveness of the approach as well as to further refine the model.

ACKNOWLEDGMENT

The authors wish to acknowledge Stefan Lorenz for his assistance with the implementation and evaluation. This work was sponsored by BMBF (Federal Ministry of Education and Research) of the Federal Republic of Germany under Contract No. 17N4809.

REFERENCES

[1] Abdel-Hamid , T. and Madnick, S., 'Software Project Dynamics: An Integrated Approach', Prentice Hall, ISBN 0138220409, 1991

[2] Abdel-Hamid, T., 'The economics of software quality assurance: a simulation-based case study,' MIS Q. 12, 3, pp. 395-411, (Sep. 1988)

[3] Basili, V., Caldiera, G., and Rombach, H.D., 'Goal Question Metric Approach,' Encycl. of Software Engineering, John Wiley & Sons, Inc., 1994, pp. 528-532.

[4] Bellifemine, F., Poggi, A., and Rimassa, G., 'JADE - A FIPA-compliant Agent Framework,' Proceedings of the 4th Intl. Conference and Exhibition on The Practical Application of Intelligent Agents and Multi-Agents. London, 1999.

[5] Chen, T., Homayoun Far, B., and Wang, Y., 'Development of an Intelligent Agent-Based GQM Software Measurement System,' doi:10.1.1.146.5373.

[6] Copeland, T., 'PMD Applied,' Centennial Books, ISBN 0-9762214-1-1, November 2005

[7] Dadam, P. and Reichert, M., 'The ADEPT Project: A Decade of Research and Development for Robust and Flexible Process Support – Challenges and Achievements,' Springer, Computer Science - Research and Development, Vol. 23, No. 2, pp. 81-97, 2009

[8] Espertech Event Stream Intelligence. http://www.espertech.com/products/esper.php [June 2010]

[9] Gelernter, D., 'Generative communication in Linda,' ACM Transactions on Programming Languages and Systems, vol. 7, nr. 1, January 1985, pp. 80-112.

[10] Grambow, G. and Oberhauser, R., 'Towards Automated Context-Aware Selection of Software Quality Measures,' Accepted for publication in Proc. of the Fifth Intl. Conf. on Software Engineering Advances (ICSEA 2010). IEEE Computer Society Press, 2010

[11] Gruhn, V., 'Process-Centered Software Engineering Environments, A Brief History and Future Challenges,' In: Annals of Software Engineering, Springer Netherlands, Vol. 14, Nrs. 1-4 / (December 2002), J. C. Baltzer AG, Science Publishers Red Bank, NJ, USA. ISSN:1022-7091, pp. 363-382.

[12] Jang, M. and J.-C. Sohn., 'Bossam: An Extended Rule Engine for OWL Inferencing,' Rules and Rule Markup Languages for the Semantic Web, Springer Berlin / Heidelberg: 128-138, 2004

[13] Johnson, P. M., 'Requirement and Design Trade-offs in Hackystat: An In-Process Software Engineering Measurement and Analysis System,' Proceedings of the First Intl. Symposium on Empirical Software Engineering and Measurement, IEEE Computer Society, 2007, pp. 81-90.

[14] Junling Huang Far, B.H., 'Intelligent software measurement system (ISMS),' Canadian Conf. on Electrical and Computer Engineering, 2005, pp. 1033 – 1036.

[15] Lavazza, L., 'Providing automated support for the GQM measurement process,' Software, IEEE, Volume 17, Issue 3, May-June 2000, pp.:56 – 62, doi: 10.1109/52.896250.

[16] Lofi C., 'cGQM - Ein zielorientierter Ansatz für kontinuierliche, automatisierte Messzyklen,' Proc. of the 4th National Conf. on Software Measurement and Metrics (DASMA MetriKon 2005), 2005.

[17] Lin, Y., Strasunskas, D., 'Ontology-based Semantic Annotation of Process Templates for Reuse,' in Proceedings of the 10th Intl. Workshop on Exploring Modeling Methods for Systems Analysis and Design (EMMSAD'05), 2005

[18] Ma, Z., Wetzstein, B., Anicic, D., Heymans, S., and Leymann, F.,. 'Semantic Business Process Repository' In Proceedings of the Workshop on Semantic Business Process and Product Lifecycle Management, 2007

[19] McBride, B., 'Jena: a semantic web toolkit,' Internet Computing, Dec. 2002

[20] Meier, W., 'eXist: An Open Source Native XML Database,' Web, Web-Services, and Database Systems, Springer Berlin / Heidelberg, 2009 pp. 169-183.

[21] Müller, R., Greiner, U., Rahm, E.: 'AGENTWORK: A Workflow-System Supporting Rule-Based Workflow Adaptation,' In Data Knowl. Eng. 51(2), pp. 223-256 (2004)

[22] Oberhauser, R.. 'Leveraging Semantic Web Computing for Context-Aware Software Engineering Environments,' In "Semantic Web", Gang Wu (ed.), In-Tech, Vienna, Austria, 2010.

[23] OpenUp http://epf.eclipse.org/wikis/openup/ [June 2010]

[24] Pedrinaci, C., Domingue, J., and Alves de Medeiros, A.: 'A Core Ontology for Business Process Analysis' (2008), pp. 49-64

[25] Prud'hommeaux, E. and Seaborne, A., 'SPARQL Query Language for RDF,' W3C WD 4 October 2006.

[26] Reichert, M., Dadam, P.,Rinderle-Ma, S., Lanz, A., Pryss, R., Predeschly, M., Kolb, J., Ly, L.T., Jurisch, M., Kreher, U., Goeser, K, 'Enabling Poka-Yoke Workflows with the AristaFlow BPM Suite,' In: CEUR proceedings of the BPM'09 Demonstration Track, Business Process Management Conference 2009 (BPM'09), September 2009, Ulm, Germany, 2009

[27] Reichert, M. and Dadam, P., 'Enabling Adaptive Process-aware Information Sytems with ADEPT2,' In: Handbook of Research on Business Process Modeling. Information Science Reference, Hershey, New York, pp. 173-203, 2009

[28] Seyyedi, M.A., Teshnehlab, M., and Shams, F., 'Measuring software processes performance based on the fuzzy multi agent measurements,' In Proc. Intl Conf. on Information Technology: Coding and Computing (ITCC'05) - Volume II, 2005. IEEE Computer Society, Washington, DC, 410-415.

[29] Seyyedi, M.A., Shams, F., and Teshnehlab, M., 'A New Method For Measuring Software Processes Within Software Capability Maturity Model Based On the Fuzzy Multi- Agent Measurements,' Proc. of World Academy Of Science, Engineering and Technology Vol. 4, 2005, pp. 257-262.

[30] Slaughter, S. A., Harter, D. E., and Krishnan, M. S. 'Evaluating the cost of software quality,' Commun. ACM 41, 8 (Aug. 1998), 67-73.

[31] Thomas, M. Redmond, R. Yoon, V. Singh, R., 'A Semantic Approach to Monitor Business Process Performance,' Communications of the ACM, pp. 55-59, 2005.

[32] Weber, B., Reichert, M., Wild, W., Rinderle-Ma, S., 'Providing Integrated Life Cycle Support in Process-Aware Information Systems,' Int'l Journal of Cooperative Information Systems, 18(1): 115-165, 2009.

[33] Dustdar, S., 'Caramba—A Process-Aware Collaboration System Supporting Ad hoc and Collaborative Processes in Virtual Teams,' in Distributed and Parallel Databases Vol. 15, Issue 1, Kluwer Academic Publishers Hingham, MA, USA (2004)