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

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

Manfred Reichert
Institute for Databases and Information Systems
Ulm University, 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 congruent with 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 software (SW) production. Increased automation support for SQA is promising for not only cost reduction and efficiency, but also for systematically improving product and process quality by means of repeatability, traceability, and consistency.

However, automation raises challenges, especially due to the dynamicity and adolescence of SW development as a discipline. Due to the significant impact of SQA on project costs [2], the cost-effectiveness of concrete SQA measures (i.e., actions) is essential, and should be based on contextually relevant SW engineering environment data for economical adaptation to changing environments and constraints. The timely assignment of SQA measures requires their tight integration in SW development processes, specifically concrete workflows that must necessarily be adapted since the exact measures are not foreknown but are contextually dependent. 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][24].

Process-Centered Software Engineering Environments (PCSEEs) [11] model and execute processes that coordinate human activities and SW development tools. One promising area for increasing the degree of automation in PCSEEs is the context-sensitive selection of SQA measures (e.g., refactoring) while taking into account people, processes, tools, and artifacts. It would include the analysis of quality metrics and measure selection for process and product artifacts to support SW engineers via automated measure suggestion at the appropriate time.

SW engineers currently lack automated guidance for SQA that can practically apply SQA cost-effectively using SQA models such as [2]. A solution is sought 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. Multiuser measures that require preparation (e.g., code inspections) should be contextually coordinated, meaning their activities appropriately included in the user's process. SQA measures should be systematically distributed, and SQA measures should be filtered based on the available time constraints and their known utility and effectiveness. 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 schedule impact. This necessitates adaptive processes that can cope with the inclusion of unforeseen activities.

In prior work, Automated Goal Question Metric (AGQM) [10] extends and automates GQM [3] by using a multi-agent system with behavior agents to select and prioritize reactive and proactive measures in alignment with goals. The current contribution builds on AGQM to contribute a holistic PCSEE integration of SQA with process management, comprising the detection of problems, the automated strategic adaptation and context-based selection of SQA measures, and adaptation of concrete process instances.

The remainder of this paper is organized as follows: the solution approach is described in Section II and Section III elucidates its realization. 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 assignment of SQA measures is addressed using various technologies for detection, mediation, and management that will now be described. The

infrastructure for these technologies is provided by the CoSEEEK framework (Context-aware Software Engineering Environment Event-driven frameworK) [17]. Figure 1 shows the simplified conceptual architecture.

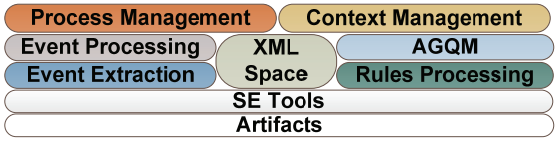

Figure 1. Conceptual Architecture

This architecture comprises the following components: *Artifacts* (e.g., source code) are processed using *SE tools* (e.g., static analysis and test tools). The *XML Space* provides communication and data storage for the event-based architecture. The *Event Extraction* module collects events from SE tools. The *Event Processing* module enriches events through contextual annotation and event aggregation via complex event processing (CEP). The *Rules Processing* module automatically analyzes metric tool reports for rule violations, providing consolidated reactive SQA measure and metric reports to the *AGQM* module [10] responsible for measure selection. The *Process Management* module applies PAIS (Process-Aware Information System) technology for the automated governance of the SW development process using workflows. Finally, the *Context Management* module aggregates information from different sources to create a holistic project context where an adaptive integration of the SQA measures selected by the AGQM module into the developer's concrete process becomes feasible.

The process of timely assigning SQA measures involves several steps as depicted in Figure 2. *Quality Opportunity Detection* detects user availability for SQA measures (so called *Q-Slots*) considering the estimated and actual durations of planned activities. *Problem Detection / Measure Proposal* comprises activities for automatic detection of problems relating to source code and for the proposal of related SQA measures in alignment with higher-level goals utilizing GQM. *Measure Tailoring* comprises steps for the context-based selection of measure and insertion point (so called *extension points* of a workflow) and the adaptation of the concrete user's workflow. Finally, the effectivity of the applied measures is assessed to improve future measure selection processes, adapting the system automatically.

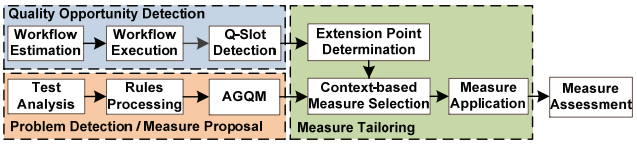

Figure 2. Conceptual Process

A. Quality opportunity detection

Workflow creation and execution are done utilizing the *Context Management* and 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 including temporal properties for estimating durations of activities and matching them later with actual execution times. 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 factor, it becomes possible to subject a certain percentage of work done in a project to SQA measures. The quality function enables control over the timely distribution of that quality effort, e.g., more up-front effort can be allocated to SQA measures to reap the benefits described in [2][24]. When a workflow is started, each activity involving a user has a planned duration. Its completion triggers the recording of the actual execution time. The SQA 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, such as a project iteration, project phase, or concrete workflow. Based on these properties, a calculation is applied revealing *Q-Slots*. 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 scenario, e.g., the assignment activities 'Ac1-Ac5' relate to assignment 'A2'. The measure proposal process is started in two possible situations: on the one hand, if activities finish earlier than expected, it is possible to insert a quality measure without affecting the schedule. Therefore, the estimated durations of all processed activities are compared to their actual execution times. On the other hand, 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

Details on proactive or reactive SQA measure detection and selection within the AGQM module of CoSEEEK were described in [10]. Metric violations are also incorporated in the *Context Management* module to enable the aforementioned selection of an *extension point* based on problems relating to artifacts to be processed by a user's future activities.

C. Measure tailoring

To achieve a high degree of utility for SQA measures, their applicability to the current situation is essential. The *Context Management* module enables the accumulation 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 SQA 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.

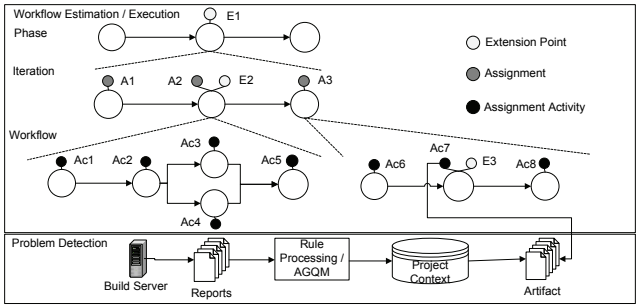

Figure 3. Workflow Enhancements / Problem Detection

The *extension point* determination process is started when an activity is completed and either enough spare time for applying a quality measure exists or the quality effort has not yet exceeded the specified quality overhead. The system searches future assignments of the current user for *extension points* and makes a selection based on different properties. E.g., if the user works on a source code artifact for which the system has detected a code quality problem, 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 for efficiency. This scenario is illustrated in Figure 3.

When the user finishes assignment 'A2' and there is an opportunity for a quality activity, the system checks future assignments, i.e. 'A3' in the given case. That assignment has an attributed sub-workflow that 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 SQA 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*).

Since the purpose of the *AGQM* module is to propose 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 using semantic enhancements in *Context Management*. These enhancements 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 as much time as the current *Q-Slot* provides. 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 the usefulness of the applied measures, an assessment phase takes place after the successful application of the measures. In that phase, the measures are rated based on their impact on the qualities of the produced artifacts. 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 stored in the *Context Management* module. Based on these values, measure assessment is conducted at certain time points in the project that can be specified a priori (e.g., at the end of a project). 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 stored by the assessment process to be 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 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 [15] for event storage. To facilitate automated problem processing, a combination of the *Event Extraction* module, the *Event Processing* module, and the *Rules Processing* module is used. Event extraction is done via the Hackystat framework [12] that 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. That module also features a web GUI, enabling users to specify code SQA measures and relate them to problems. By means of these relations, the module then creates unified reports containing violated metrics and attributed measures. Since these are unordered and many violations are conceivable, to enable automated measure assignment measures are weighted by the *AQGM* module which is realized with the JADE multi-agent system [4].

Q-Slot detection, context-aware tailoring of measures and the integration into users' workflows are managed by the *Context* and *Process Management* modules. The *Context Management* module is realized with an OWL-DL (Web Ontology Language Description Logic) ontology. The *Process Management* module is based on the AristaFlow BPM Suite [21] for adaptive process support that originated from ADEPT research [7][22]. 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 correctness checking enables the safe adaptation of workflow instances. 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.

B. Q-Slot detection

Figure 4 shows one class for each concept in the ontology enabling Q-Slot detection.

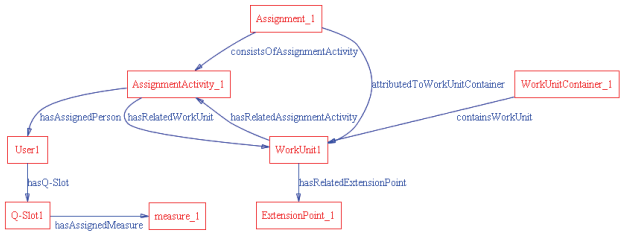

Figure 4. Ontology concepts for Q-Slot detection

The semantic enhancements to workflows and their nodes are modeled by the concept of the *WorkUnitContainer* containing *WorkUnits*. Since both of them 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 a priori, the concept of the *Assignment* is introduced. *Assignments* have properties for planned duration and for actual execution time. They can have multiple attributed *AssignmentActivities*. An example for an *Assignment* would be 'Develop Feature x' with *Assignment-Activities* like 'Write code' or 'Write Developer Test'. Since the *Assignments* and the *AssignmentActivities* are connected, the actual duration for the *Assignment* can be determined by durations of the *AssignmentActivities*. Thus, actual spare times for a certain user can be detected by a SPARQL [20] query returning all *Assignments* for that *Person* to compare the planed and 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, if the smallest time category of a measure fits, a *Q-Slot* is generated. Should another *Assignment* complete before the *Q-Slot* can be inserted, the calculation is repeated, updating or deleting the *Q-Slot* dependent on the available time. Thus, multi-user measures become possible. If a user generates a *Q-Slot* and another user has already generated one but not yet begun, both *Q-Slots* will be connected and multi-user measures be 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 to the users taking part in a multi-user measure. The scenario section will exemplify the selection of a multi-user measure.

C. Extension point determination

To enable context-based measure selection, possible future *ExtensionPoints* for a user must be determined. As depicted in Figure 4, *ExtensionPoints* are connected to *WorkUnit,s* meaning that the *Measure* of a *Q-Slot* can take place after the respective *WorkUnit*. To determine upcoming *ExtensionPoints*, the system executes the algorithm shown in Listing 1 that takes an ordered list of *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 an *Assignment* can be connected to a *WorkUnit* at any level of abstraction, there can be *WorkUnits* on a level above and below it. The example 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 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 super ordinate *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* and *ExtensionPoint* selection both depend on a weighting of the *Measures*. Prior to that, for each *ExtensionPoint,* a query is executed returning the *Measure* with the highest position in the list of the *AQGM* module matching the properties of the *ExtensionPoint* and the *Q-Slot*. Listing 2 shows the respective SPARQL query.

Listing 2. Measure preselection

```
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* and the time category and number of users for the *Q-Slot*.

The weighting depends on different factors: first, the imminence *i* of the *ExtensionPoint* ($0 < i <= 1$; initial value 0.9, for each following *ExtensionPoint* 90% of predecessor) is considered. Second, the strategic alignment *sa* of the *Measure* (i.e., the position in the ordered list of the *AGQM* module). Third, the *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). For instance, the measure type 'code' denotes measures related to source code problems, and can thus only be applied if future activities 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 after scheduled changes to artifacts, avoiding additional overhead for checking out / in and testing. After this weighting procedure, the *ExtensionPoint* with the highest weighted *Measure* is chosen for insertion. The calculation of the Measure Weight *mw* is shown in Formula (1).

$$i = 0.9^{n}$$
$$sa = \left(1 - \frac{listPosition}{listItems}\right) \quad (1)$$
$$mw = i * sa * mu * mtf$$

For the calculation of the imminence, the index for upcoming *ExtensionPoints* is n, starting with 0 for the first one. 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).

E. Applying measures

After all parameters have been determined, the point 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 integration exist: initiation of a new workflow instance or direct integration of one or multiple activities in the current workflow. 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 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 time of the measure execution is associated to the impact on KPI trends for utility analysis. Figure 5 shows instances of the relating concepts in the ontology illustrating the connections between the concepts.

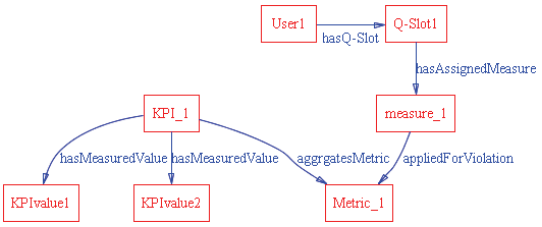

Figure 5. Measure Utility Assessment

KPIs are calculated by the *AGQM* module. Each *KPI* has multiple *MeasuredValues* that record the value of one *KPI* calculation. 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 involved person is also considered, reflecting the higher probability of ineffectively applying a measure by less skilled persons (Skill Factor *sf*: $0 < sf < 1$; 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.

$$mu = mu * \left(1 + \frac{\sum_{0}^{9} kpi\Delta_n * \left(1 - \frac{n}{10}\right)}{10}\right) * sf \quad (2)$$

G. Modeling Effort

To keep the modeling effort reasonable, some functions and standardized definitions are provided. The semantic enhancements to process management are generated automatically from the workflow templates of the *Process Management* module. 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 automatically established by the system.

A set of standard SQA measures can be provided including parameters for common static analysis tools such as PMD. 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.

IV. Evaluation

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

A. Scenario

To illustrate the application of the concept, the OpenUP [18] SW development process was chosen with the scenario focusing on the Construction Phase. The 'Develop Solution

Increment' process was planned. Figure 6 illustrates the configuration for this scenario.

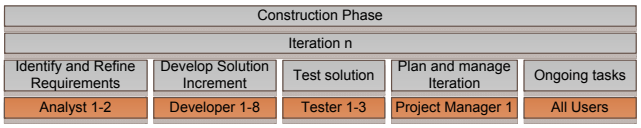

Figure 6. Scenario Configuration

There are four different estimated activities *(Assignments*) that are performed by different project members plus activity 'Ongoing Tasks' that represents activities, which are not part of the project schedule. The scheduled activities are all based on workflows. The workflow for the 'Develop Solution Increment' activity contains *AssignmentActivities* 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' *AssignmentActivity,* with 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' *AssignmentActivity* with measure type 'test' ('Test'). Both *ExtensionPoints* are related to the code the user currently works on and are thus only taken into account if measures relating to the code processed by that activities have been proposed by the *AGQM* module. The third *ExtensionPoint* is attributed to the 'Develop Solution Increment' activity enabling SQA measures between the estimated activities ('Activity'). The last *ExtensionPoint is* assigned to the iteration and used for SQA measures that only fit at the end of an iteration ('Iteration').

In the scenario, eight developers are involved, each having eight 'Develop Solution Increment' *Assignments*. The scenario relies on execution time deviation and no specified quality overhead. Since the *Q-Slot* detection utilizes execution time deviations, it is independent of the duration of the *Assignments* or the work hours per day. To keep the scenario simple, it is assumed that all *Assignments* take one workday. For this scenario, the SQA measures consuming the least time take two hours. As some *Assignments* take longer and others finish earlier, it is assumed that a quality measure insertion becomes possible four times: For dev2 after *Assignment* 3, for dev3 after *Assignment* 5, for dev5 after *Assignment* 6 and for dev7 after *Assignment* 7.

Table I illustrates future *ExtensionPoints* and their values for each user at the point of time where the measure proposal is started for the first time. The shaded cells show which *ExtensionPoint* was selected due to the highest weight.

For user 'dev2', this occurs at the end of *Assignment* 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 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 ofsa= 0.99 and a Type Factor of tf=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 *Assignment* 4. Therefore, measure 'm2' is chosen for the 'Code' *ExtensionPoint* of *Assignment* 4. The measure has a sa=0.75 and a higher Type Factor of tf=1.4. It therefore is selected for the user.

Table .I 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

For user 'dev3', the calculation starts after *Assignment* 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 weight and is planned for that user. When the calculation for user 'dev5' starts at the end of *Assignment* 6, there is also the *Q-Slot* of user 'dev3' that has been planned but not used yet. Thus, two connected *Q-Slots* are available, causing measure 'm4' to proposal, which is a multi-user measure and has sa=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 then used having the highest weight now.

At the end of the iteration, the assessment is performed by the system. This relies on the development of the KPIs that, in turn, rely on 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 scenario. Thus, there are eight points where KPIs are calculated, each after the completion of one *Assignment*. Table II illustrates the development of the related KPIs. KPIs are computed by the AGQM module to have a value that lies between 0 (bad) and 1 (perfect).

Table II. 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 the scenario only covers one iteration with no activities ahead of the iteration, not all eight time points were

available for the measure assessment. *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 KPI that is of interest is from time point 3 to 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 '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. Thus, real case studies conducted in the future will be used to evaluate the results and to refine the calculations.

B. Measurements

Initial performance evaluations were used to assess preliminary sufficiency of the approach for practical use. 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 in the *Process Management* module. The test system consisted of an AMD dual core Opteron 2.4 GHz processor, 3.2GB RAM, WinXP Pro SP3, and JRE 1.5.0_20. AristaFlow was used in version 1.0.0beta - r73, its server was running 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 using the average of the last three measurements.

The *extension point* determination and the context based measure selection are conducted together in the context module. As depicted in Table III, the measurements were conducted with different numbers of *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 III. 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 IV.

Table IV. 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 support for process adaptability, one approach that considers support of users for ad hoc changes is ProCycle [26]. ProCycle applies case-base reasoning to assist end users in the re-use of process instance changes that were applied in a similar problem context in the past. Caramba [27] features support for ad hoc workflows using connections between artifacts, resources, and processes to coordinate virtual teams. The approach presented in [16] utilizes a combination of agent and process management technology to enable automatic process adaptations, which 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.

Several approaches combine semantic and process management technology. An ontology for business process analysis was developed in [19] for improving process compliance analysis for standards or laws like Sarbanes-Oxley act. In [25] 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 [14] aims at facilitating process models across various model representations and languages. It features multiple levels of semantic annotations 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 insertion of SQA measures into the workflows of users.

Various approaches provide integration of the GQM technique into a project. The Intelligent Software Measurement System [5] uses groups of agents for user assistance and determination of different parts of the GQM plan. In [13] a tool was developed allowing the creation of GQM plans using predefined forms and the verification of its structural consistency and the reuse of its components. The approach presented in [23] also utilizes agents, for the requirements process of the SW-CMM (Software Capability Maturity Model) model. The focus is the measurement and analysis of SW processes using agents and fuzzy logic. In contrast to the aforementioned approaches, CoSEEEK focuses on utilizing context knowledge for the automatic adaptation of SQA 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, the SW engineering domain manifests various challenges for adaptable process and quality automation. Development process models have typically remained abstract and not been used for automated workflow guidance, while SQA provided abstract quality

guidance that relied on human triggering, analysis, and concretization of activities. Ideally, opportunities for SQA should be applied at the appropriate time with minimal disruption for developers to minimize overhead. Integrating techniques and sensors in the PCSEE 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 agent-based GQM technique mediates SQA measures according to KPIs. Opportunities for SQA 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 integration of concretized SQA in the process, improving SQA via 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 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][24]) 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 and 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, 1994, pp. 528-532.

[4] Bellifemine, F., Poggi, A., and Rimassa, G., 'JADE - A FIPA-compliant Agent Framework,' Proc. of PAAM'99, 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, Nov. 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 SQA measures,' In Proc. of ICSEA 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),, pp. 363-382.

[12] 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,, 2007, pp. 81-90.

[13] 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.

[14] Lin, Y. and Strasunskas, D., 'Ontology-based Semantic Annotation of Process Templates for Reuse,' in Proc. of EMMSAD'05, 2005

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

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

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

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

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

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

[21] Reichert, M. et al, 'Enabling Poka-Yoke Workflows with the AristaFlow BPM Suite,' In: CEUR proc. of the BPM'09 Demonstration Track, 2009.

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

[23] 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.

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

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

[26] Weber, B., Reichert, M., Wild, W., and 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.

[27] 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.