User-centric Abstraction of Workflow Logic Applied to Software Engineering Processes

Gregor Grambow[1], Roy Oberhauser[1], Manfred Reichert[2]

[1] Computer Science Dept., Aalen University
{gregor.grambow, roy.oberhauser}@htw-aalen.de
[2]Institute for Databases and Information Systems, Ulm University, Germany
manfred.reichert@uni-ulm.de

Abstract. Software development is a dynamic, complicated, and labor-intensive undertaking. Numerous software engineering process models have been created and applied to address its complexity, schedule pressure, and product quality. These process models are rather abstract and not directly operationally relevant for the software engineers executing these processes, since they mostly provide relatively coarse-grained work packages and lack fine-grained user-centric workflows directly supporting users. Such user-centric workflows have been difficult to implement in an automated fashion as they are very dynamic and user acceptance for both modeling and prescribing such fine-grained activities is fairly low. This paper provides an approach to abstractly model user decisions influencing the actual trace of such automated workflows. By hiding internal complexity, communication with users is simplified while supporting required flexibility. This contributes towards removing hindrances and enabling the application of and user acceptance for automated user-centric workflows in software engineering and in domains exhibiting similar issues.

Keywords: Human-centric Process-Aware Information Systems; User-centric Workflows; Process-Centered Software Engineering Environments

1 Introduction

Software development is a complicated process, and software engineering (SE) projects continue to face challenges and struggle with inherent difficulties [1][2][3]. The high dynamicity inherent to that discipline and the intangibility of the developed product hamper consistent process management. Furthermore, the developer must address various requirements concerning intellectual efforts and dynamic collaboration with others. Much of this remains undetected, untraced, and ungoverned. However, in various domains it has been shown that process orientation and explicit process management can be beneficial [4][5]. Process management fosters project efficiency [6] and product quality [7].

Based on such knowledge, various process models for SE have been developed. Examples include the Unified Process [8] and its variants or the V-model XT [9].

They support establishing, documenting, and tracking the SE process. Yet several problems occur in process execution: Most SE process models remain too general by describing abstract work packages. On the operational level, where individuals interact with tools and among each other to actively create software, SE process models lack concrete intentional support [10]. Thus, it is problematic to govern the SE process using such a model and support the people in their everyday work, as the process does not really "touch" them. Thus, maintenance of the processes model incurs additional overhead for the participants, because alignment of the abstract process with operational reality has to be manually established and tracked. Furthermore, process execution is not actively integrated into project execution. The process is not automatically aligned to events happening in various project situations, while the integration of information between process management and other areas like quality or knowledge management remains difficult.

Our previous work has addressed a number of these challenges: Facilities were developed that enable the acquisition and integration of project context data with process execution [11]. Execution semantics for process management have been extended to enable the mapping of process models and the integration of different levels of user activities. The automated exchange of information between process execution and areas like knowledge management [12] and quality management [11] has also been established. However, SE processes remain too abstract, and a tool seeking to truly support software engineers holistically in their projects should be aware of what they are actually doing. Thus, user-centric workflows are required. This notion is explained in the following:

In our context, workflows denote concrete sequences of activities, while (SE) processes are rather abstract, concerning activity sequencing but also integrating other aspects (such as the team or organizational structure). User-centric workflows are workflows that describe or govern concrete activities of humans (like, in SE, creating a new software function). This implies, on the one hand, that all activities of such a workflow contain task information guiding users (for tasks like, in SE, creating a software build from some source code). On the other hand, as the workflow shall govern what the user concretely does, the latter needs to influence and control the actual trace of the workflow that is required to achieve the goal.

To enable reasonable automatic governance and support for user-centric workflows, the following requirements would have to be satisfied:

- The granularity of workflow activities in the workflow should be fine enough to make the governing system aware of what the user is really doing, yet coarse enough not to overburden the user with copious micro-activities.
- The user should have control over the decisions in the workflow as that workflow is used for governing his activities.
- Workflow interaction shall not distract the user from his or her actual work.

The remainder of this paper is structured as follows: Section 2 presents a concrete problem scenario, Section 3 gives insights on the developed approach, and Section 4 shows its practical application to the problem scenario. The paper concludes in Section 6 after discussing related work in Section 5.

2 Problem Scenario

This section demonstrates shortcomings of contemporary process management systems that can cause problems when trying to model user-centric workflows. As SE is a dynamic discipline, it epitomizes such a workflow. For this paper, we create a small example using experiences we gained while working with software development companies. The example concerns a workflow that governs activities utilized for creating a new piece of software. The workflow shown in Fig. 1 is a simplified version of a workflow we created during an interview with software developers at a company that produces software.

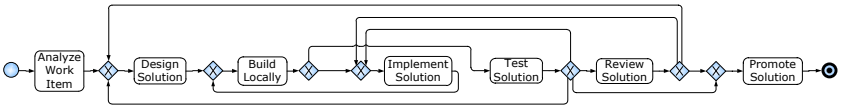

Fig. 1. Example workflow (unstructured)

The workflow starts with the analysis of the work item (assignment) to determine what has to be done to achieve the work item goal (e.g., create the desired functionality) followed by the design of the solution. Thereafter, the code of the system is checked out and a local build is created. Additionally included activities concern the implementation as well as testing and a review of the solution. The workflow concludes with the promotion of the developed solution, meaning the integration of the locally created source code into the mainline development branch within the shared version control system (VCS). As the described activities in most situations cannot be simply executed in a basic linear non-repeating sequence, the workflow contains several loops.

Note that the workflow is not well structured according to [13], since loops overlap in some cases, contradicting proper nesting of workflow elements. Unstructured workflows have been proven to be badly readable and error prone. Thus, the workflow was restructured to enable proper nesting of the elements as shown in Fig. 2. This has been done by duplicating the build activity to resolve the overlapping loops. This configuration was chosen based on domain knowledge, since a real difference between the two new building activities exists: The first one, *Initial Local Build*, is conducted once before any implementation changes are applied to verify that the checked out versions of the code files build on the local machine (a.k.a. sandbox or programmer's directory). The second activity, *Build Locally*, is conducted to verify that the changed code is buildable.

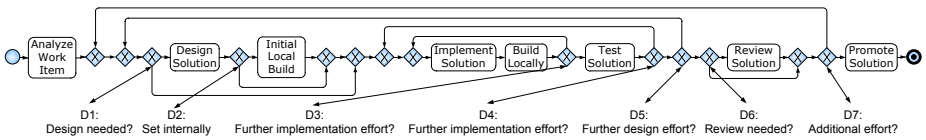

Fig. 2. Example workflow (structured)

The workflow shows the different decisions that have to be connected to certain variables to govern the sequencing of the activities. Each decision could be connected to a question towards the user to acquire necessary and missing information. Note that this realization of the workflow shows several flaws: while processing the workflow, the user always has to provide values for all decisions except D2 (this one can be set internally as the initial build activity should be only executed once). Consider activity *Test Solution:* it has four possible successors (*Review Solution*, *Promote Solution*, *Design Solution*, and *Implement Solution*), but the user has to be involved in all decisions in the workflow. This kind of workflow governance can be confusing and cumbersome for the user, provoking mistakes on user decision input that then result in workflow traces that are inconsistent with reality. This type of workflow interaction will most likely be perceived as burdensome by users, which may affect their overall impression and resulting acceptance or rejection of such an automated guidance system. To resolve this, a more user-centric way of communication is desirable, letting the user focus on what she or he is doing rather then being distracted by the supplementary information requirements of a workflow that should be assisting them.

3 Abstract User Decision Mapping

This section elucidates the solution for the aforementioned problem starting with a brief introduction on the on the framework that forms the basis for that solution.

3.1 Solution Basis

To address the aforementioned challenges, we have developed a framework called CoSEEEK (Context-aware Software Engineering Environment Event-driven frameworK) [14]. With holistic SE process support as a primary goal, the framework was designed to be an active component in that process supporting vertical and horizontal integration of activities in a project. In this context, vertical means abstract to concrete levels of activities while horizontal means integration of different project areas like, e.g., quality management and software development. The framework integrates various technologies to create an infrastructure that can extract, unify, and utilize context, process, and project information in an automated fashion. In the following, different components of the framework are briefly explained to introduce the background for the solution approach presented in this paper:

To be able to provide automatic workflow governance, the *Process Management* component integrates a Process-Aware Information System (PAIS) that enables workflow execution and enforces correctness criteria in this area. That way the basis for mapping parts of the SE process models to automatically governed and supported workflows is set. As SE workflows are dynamic, a dynamic PAIS (Aristaflow BPM suite [15]) has been integrated to enable the adaptation of running workflow instances.

The *Context Management* component constitutes the core of the framework. It not only enables information storage, but also active usage of that information for

automation. Integrated semantic technology (ontology and reasoning capabilities) enables automatic information processing. In this component, extensions to the process management concepts are stored that enable the mapping of SE process models with all of their diverse additional information and dependencies. Via these extensions, the *Context Management* component has direct access to process execution and can use additional context information to optimally align the process with the actual project context.

The *Event Management* component collects and processes environmental information. Event sensors integrated into various SE tools (like Integrated Development Environments or VCS) generate events for the *Event Management* component. That component, via complex event processing, combines and distributes events to the *Context Management* component for contextual assimilation. The *Quality Management* component, in turn, enables an active connection between software quality management and process management. That way, it is possible to automatically detect problems, determine proper software quality measures for them, and integrate them into running workflows. Finally, with the *Knowledge Management* component, there is also an active connection between project knowledge and executed process. Users can collect knowledge in a semantic Wiki and the system automatically injects that information into process execution where appropriate.

We have briefly explained how process models are mapped, how the process is integrated with real time information from the SE projects, and how the process is unified with other project areas like quality or knowledge in an automated fashion. However, to truly connect process execution with users in SE, user-centric workflows are required that guide users and associate their activities with the abstract process. Yet to overcome the obstacles described in Section 2, connections between the *Context Management* and *Process Management* components are utilized as illustrated in Fig. 3. All basic process management concepts are mirrored in the *Context Management* component and are tightly connected by the framework. These concepts enable the annotation of process management with contextual information, while enabling the framework to actively influence process management.

3.1 User Decision Modeling Concepts

Fig. 3 illustrates the mappings of the process management concepts existing in the *Context Management* component using a simple example: A *Work Unit Container Template* mirrors a workflow template and a *Work Unit Template* mirrors an activity of a workflow template. That way, the *Context Management* component is aware of the workflow templates and can automatically instantiate new workflows from them. The latter are also mirrored using the *Work Unit Container* and the *Work Unit*. These concepts give the *Context Management* component enhanced control over workflow execution and thus enable it to completely encapsulate and extend the *Process Management* component and to provide high level context-aware workflow governance to the users (see [12, 13] for further details).

The approach taken to enable abstraction from internal process logic and variables is grounded on these concepts. The *Work Unit Container Template* is extended by

Workflow Variable Templates that map the workflow variables defined in the workflow template. For these variable templates, the *Workflow Variable Values* provide possible pre-defined values. The latter can then be used to set the variables for a workflow instance during execution. These are mapped by the *Workflow Variables*. Each *Work Unit Container Template* has a set of *Workflow Variable Templates* that provide the initial values for the variables of a new workflow instance. To grant the user flexible and abstract control about the workflow variables, the *Workflow User Information* and *User Decision Alternative* are used: The former is connected to a *Work Unit Template* and used to inform the user executing that activity about a decision the user has to make. The latter represents one alternative of such a decision, and is, in turn, connected to *Workflow Variable Values* that are used when that alternative is chosen by the user.

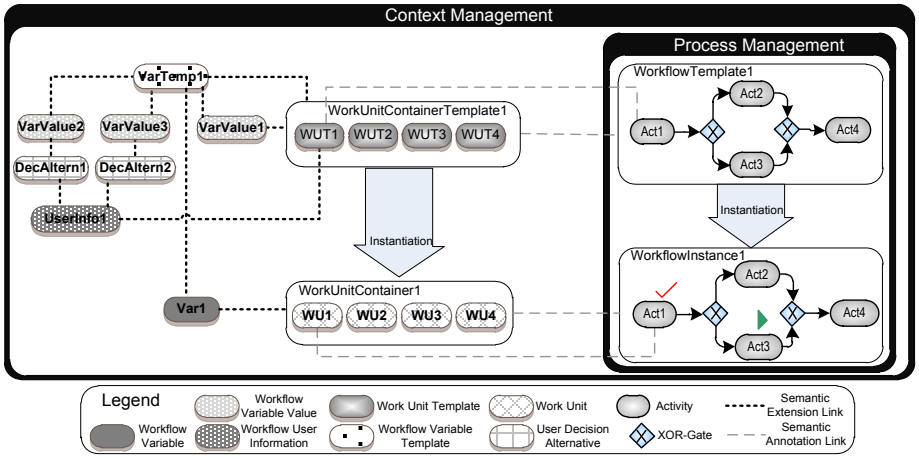

Fig. 3. Extensions to workflow governance.

Consider the simple example in Fig. 3: WorkflowInstance1 contains four activities from which two (*Act2* and *Act3*) are mutually exclusive. For example, they could be two different code review activities (e.g., a peer review and a code inspection, both having different properties regarding required effort, duration, and error detection rate). As the user has to decide which one is appropriate in the current situation, a value for the variable that is used for the XOR-gate in the workflow has to be acquired from the user. In this example, *UserInfo1* that is connected to the *Work Unit Template* that maps the first activity in the workflow would gather information on the upcoming decision while the user executes the current activity (e.g., asking "How much time is available for reviewing your code?"). The two options for that decision are modeled by *DecAltern1* and *DecAltern2* that provide the values for the XOR-gate using *VarValue2* and *VarValue3* (e.g., Sufficient time left / High schedule pressure). This simple example is used for illustration; to address the problem scenario of Section 2, more complex mappings are shown in the next section.

To have a sustainable basis for the approach, the operational semantics of the described concepts rely on the definitions in Table 1. Due to space limitations, not all

definitions are shown in this paper. The basic mapping and extension of concepts within a PAIS has been described in previous work [11].

Table 1. Definitions

Concept	Description
Identifiers	All valid identifiers over a given alphabet. All concepts have a name ∈ *Identifiers*
Types	All definable object types. All concepts have a distinct type ∈ *Types* that is defined by the tuples in the following definitions.
WFTemplates	All workflow templates within a PAIS
ActivityTemplates	All activities within workflow templates in a PAIS
WFInstances	All workflow instances within a PAIS
ActivityInstances	All activities within workflow instances in a PAIS
WorkUnitContTempls	All *work unit container templates* that are used to map and extend the workflow templates of a PAIS
WorkUnitTempls	All *work unit templates* that are used to map and extend the activity templates of the integrated PAIS
WorkUnitConts	All *work unit containers* that are used to map and extend the workflow instances of the integrated PAIS
WorkUnitTempls	All *work units* that are used to map and extend the activities of a PAIS

On this basis, the following definitions are made for the concepts used for the mapping of internal workflow variables to high-level user decisions. These formal definitions support the exact description of each concept and its relations, and enable the further definition of conditions that guarantee correct executability. The definitions assume that each concept has a type ∈ *Types* and a name ∈ *Identifiers*.

Definition 1. A ***workflow user information*** represents one decision that a user can make when processing an activity of a workflow. It is a tuple workflowUserInfo = (type, name, decAlternSet, workUnitTempl, info) where

- decAlternSet is a finite set of decision alternatives with decAltern ∈ *DecAlternatives* (cf. Def. 2).
- workUnitTempl ∈ *WorkUnitTempls* is the work unit template to which workflowUserInfo belongs.
- info ∈ STRING is the information about the decision for the user.

WorkUserInfos describes the set of all definable workflow user information.

Definition 2. A ***user decision alternative*** represents one alternative for a decision decision a user can make when processing an activity of a workflow. It is a tuple decAlternative = (type, name, userInfo, varValueSet, decInfo, decId, standard) where

- userInfo ∈ *WorkUserInfos* is the workflow user information to which decAlternative belongs.
- varValueSet is a finite set of workflow variable values with varValue ∈ *VarValues*.
- decInfo ∈ STRING is the information for the decision alternative.
- decId ∈ INTEGER is the id for the decision alternative.
- standard ∈ BOOLEAN marks the initially selected decision alternative.

DecAlternatives describes the set of all definable user decision alternatives.

Definition 3. A ***workflow variable value*** represents a value to which a workflow variable can be set in a certain situation. It is a tuple varValue = (type, name, varTempl, workUnitContTemppl, decAlternative, value) where

- varTempl ∈ *VarTempls* is the workflow variable template whose corresponding workflow variable is set by varValue.
- workUnitContTempl ∈ *WorkUnitContTempls* ∪ {NULL} is the work unit container template for whose instances varValue provides an initial value for a variable; will be NULL in case varValue is supplied for a decision alternative
- decAlternative ∈ *DecAlternatives* ∪ {NULL} is the decision alternative for which varValue provides a value for a workflow variable; will be null in case varValue is supplied for a work unit container template
- value ∈ INTEGER is the value to be used for the variable.

VarValues describes the set of all definable workflow variable values.

Definition 4. A ***workflow variable template*** represents the definition of the mapping of a workflow variable that exists for a workflow template within a PAIS. It is a tuple varTempl = (type, name, workUnitContTempl, paisVarName) where

- workUnitContTempl ∈ *WorkUnitContTempls* is the work unit container for which varTempl represents a variable.
- paisVarName ∈ STRING is the value to be used for the variable.

VarTempls describes the set of all definable workflow variable templates.

Definition 5. A ***workflow variable*** represents the mapping of a workflow variable that exists for a certain workflow instance within a PAIS. It is a tuple workVar = (type, name, workUnitCont, varTempl, currentValue) where

- workUnitCont ∈ *WorkUnitConts* is the work unit container for which varTempl represents a variable.
- varTempl ∈ *VarTempls* is the variable template to which workVar belongs.
- currentValue ∈ INTEGER is the current value of the variable.

WorkVars describes the set of all definable workflow variables.

These five definitions provide the basis for the concept developed in this paper. To ensure executability, some basic conditions have to be satisfied. These have been defined in the following: for all variables in a workflow, mappings must exist (cf. Def. 6 b), their association to *Workflow Variable Templates* must be clear (cf. Def. 6 a), and initial values should be provided (cf. Def. 6 c) so that all workflows can be executed without user communication. The latter should also be well defined, meaning that for each decision, one or more alternatives exist (cf. Def. 6 d) from which one is set as the default (cf. Def. 6 e), and all have variable values to set if a choice exists (cf. Def. 6 f), since a user decision with no impact would be irrelevant.

Definition 6. Let workUnitCont = (type, name, wfInstance, workUnitSet, assignment, mandInputSet, outputSet, roleSet, flowVarSet, basis) ∈ WorkUnitConts be a *work unit container* belonging to a project within the system. Then:

a) ∀varTempl: varTempl.paisVarName = wfTemplate.variable.name, i.e., the names of the variables in the PAIS and the mapping with *variable templates* must match.
b) ∀WFtemplate.variable: ∃varTempl ≡ WFtemplate.variable, i.e., there exists a mapping *workflow variable template* for all variables in a workflow template.
c) ∀varTempl ∈ workUnitContTempl: ∃varValue with varValue.varTempl = varTempl, i.e., all *variable templates* within a *work unit container template* shall have an initial value.
d) ∀workflowUserInfo: |workflowUserInfo.decAlternSet| ≥ 1. i.e., every *workflow user information* must have at least one *decision alternative.*
e) ∀workflowUserInfo: |decAlternative ∈ workflowUserInfo.decAlternSet with decAlternative.standard = TRUE| = 1, i.e., for each *workflow user information* there has to be exactly one standard alternative.
f) ∀decAlternative with |workflowUserInfo.decAlternSet| > 1:
|decAlternative.varValueSet| ≥ 1, i.e., if a *workflow user information* has more than one *decision alternative*, each *decision alternative* must set at least one variable.

Utilizing these definitions, a workflow instance is always provided with the necessary values of all the variables used to govern its execution. When such an instance is created from a workflow template, all variables receive their initial values from the *workflow variable values* defined for the *work unit container*. These values shall be defined in a way to represent the typical trace of the workflow, minimizing the necessary user interaction required for appropriately governing the instance. That user interaction is illustrated in Fig. 4. As aforementioned, the *Context Management* component adds several types of information to the workflows of the *Process Management* component. In this case, information about the users' activities is relevant. It comprises an *Assignment* that represents information about the activity for which the workflow was initiated, e.g., 'Develop new feature X'. The different activities to reach that goal are represented by *Assignment Activities*, as, e.g., 'Implement Solution'. To enable better operation assistance, the concept of the *Atomic Task* was also added, representing activities like checking in source code or running unit tests. For more information on these concepts, we refer to [11].

The different steps taken by the system to extend workflow execution with more user-related semantics are now described, and illustrated in Fig. 4:

Step 1. An event (by a user or the system) causes a workflow to start.
Step 2. The workflow execution reaches one or more activities that become enabled. This information is distributed to the *Context Management* component.
Step 3. The *Context Management* component, in turn, distributes the relating *Assignment Activity* and potential additional information to the user.
Step 4. The user starts the processing of the *Assignment Activity.*
Step 5. The *Context Management* component retrieves the decision information and its alternatives and distributes it to the user.
Step 6. The user selects one decision alternative (if different from the pre-selected).

Step 7. The user finishes the processing of the *Assignment Activity.*
Step 8. The *Context Management* component informs the *Process Management* component that the active activity may complete now. This information incorporates values for the workflow instances variables.
Step 9. The Process Management component sets the values of the variables and then lets the activity complete and the workflow instance continue.

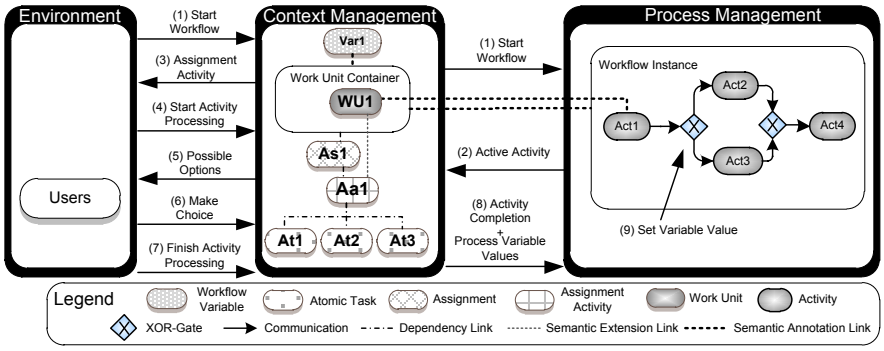

Fig. 4. Workflow enactment with extensions

This concept offers flexibility for transferring workflow governance control to users by mapping internal workflow variables to user decisions. There can be multiple ways of defining and connecting different concepts to match projects needs:

- A simple 1-1 mapping of *User Decision Alternatives* to variables. With such a mapping, a user could directly set the value for each workflow variable. In this case the *Workflow User Information* would be used to store an appropriate question or statement to inform the user what he is about to control now. The *User Decision Alternatives*, in turn, would store textual information on *decision alternatives* such as 'Is additional implementation effort still required? – Yes/No' which is less error-prone than setting a variable like 'AdditionalImplEffort – true/false';
- A more complex n-m mapping where each *User Decision Alternative* sets multiple variables to provide the user with support for a more abstract decision. Each of the alternatives could even set different sets of variables, allowing completely different traces to be produced. This way of mapping could make big complicated workflows easier to handle for users having one consistent decision with a limited set of alternatives per activity instead of having to set multiple variables all the time;
- With the abstraction from the variables and the explicit modeling of user decisions, there is the possibility of restricting certain options of a decision that would be available if there was direct access to the workflow; and
- Since the user-decision modeling is user centric and abstracted from the workflow internals, it can also serve for hiding technical complexity inherent to the workflows. Well-structured workflows are often bigger than not well-structured workflows describing the same situation (as shown in Section 2 with an additional activity). They may be more comprehensible for the modeler, but

could be more complicated for the executing person. This can be compensated in our approach via the additional abstraction layer introduced towards the user.

4 Application Scenario

This section shows the proposed approach applied to a concrete example in regard to the scenario of Section 2. In that example, five user decisions were required to properly govern the workflow (D2 can be set internally and D3 and D4 could be consolidated), all of which had to be shown to the user during the entire workflow execution. To simplify this, the following mapping of the internal workflow decisions was created: for each possible successor of an activity, a decision alternative was created. That way the user can directly choose which activity to do next while executing an activity. Thus, no additional information on the decision is necessary (and the info string of the workflow user information can remain empty). Each of the decision alternatives then sets exactly the set of workflow variables required to activate the chosen activity. Additionally, a set of initial workflow variables sets the workflow variables to the most likely trace to minimize the required user interactions. Fig. 5 shows the CoSEEEK user GUI associated with the Section 2 workflow governed and the relevant concepts connecting workflow governance with the GUI.

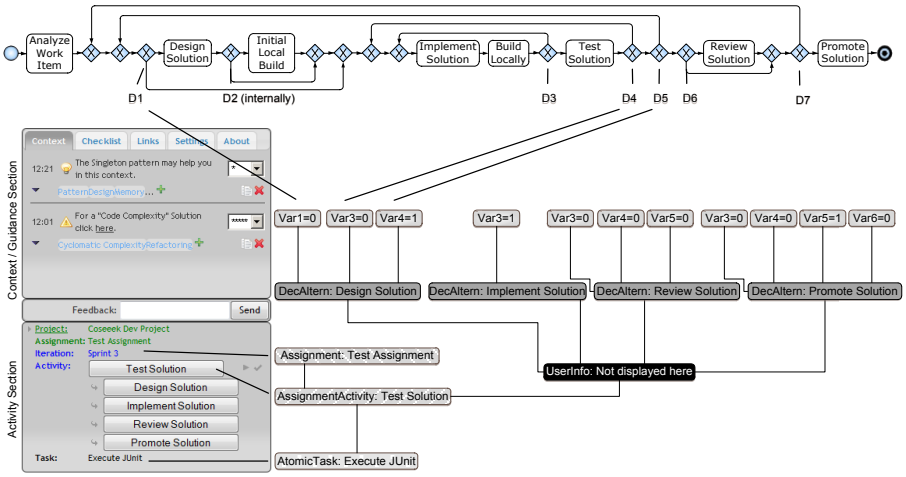

Fig. 5. Workflow GUI

The upper section of the GUI is reserved for special context and guidance information, e.g., for coordination [16] or quality [11] support. The lower part of the GUI, which is the focus of this paper, shows the activity section. The latter provides the user with valuable information about the process in which the user assignment is contained. This section also provides the user with current and future activity information: The current activity is shown as well the options for the next activity.

The user can then simply click on one of these to choose it as successor of the current one. That way, internal workflow variables are completely hidden from the user. Fig. 5 shows the GUI when the user processes the *Test Solution* activity: the user has four possible next choices and each of them affects a different set of variables. For example, by choosing Design Solution as the successor, only decisions D1, D4, and D5 need to be set with the correct variable values.

5 Related Work

In related work, there have been attempts to enable automatic support for users in the software development process. This started in the 1990s with Computer-Aided Software Engineering (CASE). Approaches like [18] tried to combine various tools for different activities to create more holistic support, yet a process component was absent. This issue was addressed by Process-Centered Software Engineering Environments (PCSEEs) like [19], which not only supported single activities but also considered their relations and impact on the produced product. Recent developments include Application Lifecycle Management (ALM) tools that connect different areas of the projects via the produced product, notably IBM Jazz [20]. While these approaches aim for active and holistic automated support, none incorporates user-centric support for workflows as shown in this paper.

Concerning abstraction from process management internals, there has been a fair amount of related work: For example, [21][22] provides users with abstractions from technical process specifications using views. Similarly, [23] and [24] simplify process models to generate views for better comprehension. [23] abstracts parts of the model using semantic similarity of activities using structural information of the models. [24] applies a two-phase procedure for aggregating parts of process models. Although these approaches deal with the abstraction of process models, they only aid the users by providing better views of the models and not in abstracting and simplifying the communication with users. In contrast, the approach presented in this paper contributes a technique for transferring workflow control towards the users by making communication with the process more convenient and less error-prone.

The approach presented in [25] incorporates several extensions to process management concepts similar to this approach. However, the former deals with design-time process model configurations in contrast to the dynamic runtime process support CoSEEEK offers.

6 Conclusion

SE is a dynamic discipline that highly relies on the participating individuals, their intellectual work, and their collaborations. It thus poses a challenge towards automating process management that implements SE process models to an operational level. In our previous work, we implemented the basis for extending process management concepts to enable the enrichment of the processes. This paper adds a way of communicating between the process models and the users that is oriented

towards user needs and preferences. By abstracting from process management internals, the communication can be modeled in a way that better assists users. The mapping from internal process variables to user decisions adds flexibility and simplifies the communication with the user. The application to a user-centric software development workflow yields the following benefits: Easy pre-configuration of the variables is possible. The amount of communication with the user can be reduced. Complex mappings of variables to decision alternatives become possible. Moreover, the system has improved options for traceability and support, since it is aware of the users' intent so that dynamic workflows can be governed without taking the users' freedom away.

Based on this support for abstract user decision modeling, future work will not only include currently ongoing industrial investigation, but also extend this concept to enable the system to exploit more of its available context data to improve the user experience. This includes better alignment of the workflow to the current situation (e.g., skill of the user); extending the initial set of variable values with multiple sets of initial values (note that since such properties cannot be known a priori, the dynamic alignment is applied just in time when the workflow is started); supporting additional influencing factors (e.g., the quality goals of a project) to enable the same user decision to produce different traces for different goals (like, e.g., choosing a more effective review activity if goals like reliability and maintainability are important); and the inclusion of other situational properties (schedule pressure, criticality, etc.) that have been modeled in [17] and will be also connected to the user decision modeling.

References

1. Brooks, F.P.: No Silver Bullet: Essence and Accidents of Software Engineering, Information Processing, 1986.
2. Yourdon E., 'Death March,' 2nd Ed. Prentice Hall, Upper Saddle River, NJ, 2004.
3. Jones C.: Get Software Quality Right. Dr Dobb's Journal, June 28, 2010.
4. Lenz, R., Reichert, M.: IT support for healthcare processes-premises, challenges, perspectives. Data & Knowledge Engineering, 61(1), pp. 39-58, 2007.
5. Müller, D., Herbst, J., Hammori, M., Reichert, M.: IT support for release management processes in the automotive industry. Proc. 4th Int'l Conf. on Business Process Management, pp. 368-377, 2006.
6. Gibson, D.L., Goldenson, D.R., Kost, K.: Performance results of CMMI-based process improvement. Technical Report, Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, 2006.
7. Heravizadeh, M.: Quality-aware business process management. PhD Thesis, Queensland University of Technology, 2009.
8. Jacobson, I., Booch, G., Rumbaugh, J.: The unified software development process. Addison-Wesley, 1999.
9. Rausch, A., Bartelt, C., Ternité, T., Kuhrmann, M.: The V-Modell XT Applied–Model-Driven and Document-Centric Development. Proc. 3rd World Congress for Software Quality, VOLUME III, pp. 131-138, 2005.
10. Wallmüller, E.: SPI-Software Process Improvement mit Cmmi und ISO 15504. Hanser Verlag, 2007.

11. Grambow, G., Oberhauser, R., Reichert, M.: Contextual Injection of Quality Measures into Software Engineering Processes. Int'l Journal on Advances in Software, 4(1 & 2), pp. 76-99, 2011.
12. Grambow, G., Oberhauser, R., Reichert, M.: Towards Dynamic Knowledge Support in Software Engineering Processes Proc. 6th Int'l Workshop on Applications of Semantic Technologies (AST'11), held in conjunction with INFORMATIK'11 (accepted for publication), 2011.
13. Mendling, J., Reijers, H.A., van der Aalst, W.M.P.: Seven process modeling guidelines (7pmg). Information and Software Technology, 52(2), pp. 127-136, 2010.
14. Oberhauser, R.: Leveraging Semantic Web Computing for Context-Aware Software Engineering Environments. Semantic Web. In-Tech, Vienna, Austria, pp. 157-179, 2010.
15. Dadam, P., Reichert, M.: The ADEPT project: a decade of research and development for robust and flexible process support. Computer Science-Research and Development, 23(2), pp. 81-97, 2009.
16. Grambow, G., Oberhauser, R., Reichert, M.: Enabling Automatic Process-aware Collaboration Support in Software Engineering Projects. Selected Papers of the ICSOFT'11 Conference. Communications in Computer and Information Science (CCIS) (accepted for publication), 2012.
17. Grambow, G., Oberhauser, R., Reichert, M.: Contextual Generation of Declarative Workflows and their Application to Software Engineering Processes. Int'l Journal On Advances in Intelligent Systems, 4(3 & 4) (accepted for publication), 2012.
18. Emmerich, W., Kroha, P., Schäfer, W.: Object-oriented database management systems for construction of CASE environments. Database and Expert System Applications, LNCS, 720, pp. 631-642, 1993.
19. Conradi, R., Hagaseth, M., Larsen, J.O., Nguyen, M., Munch, B., Westby, P., Zhu, W., Jaccheri, M., Liu, C.: Object-oriented and cooperative process modelling in EPOS. Software Process Modelling and Technology. Research Studies Press Ltd., Taunton, UK, pp. 9–32, 1994.
20. IBM Jazz - https://jazz.net/ [received Jan. 2012]
21. Bobrik, R., Reichert, M., Bauer, T.: View-based process visualization. Proc. 5th Int'l Conf. on Business Process Management, pp. 88-95, 2007.
22. Reichert, M., Kolb, J., Bobrik, R., Bauer, T.: Enabling Personalized Visualization of Large Business Processes through Parameterizable Views. Proc. 27th ACM Symposium On Applied Computing, 2012 (accepted for publication).
23. Smirnov, S., Reijers, H., Weske, M.: A semantic approach for business process model abstraction. Advanced Information Systems Engineering, 6741, pp. 497-511, 2011.
24. Eshuis, R., Grefen, P.: Constructing customized process views. Data & Knowledge Engineering, 64(2), pp. 419-438, 2008.
25. La Rosa, M., Dumas, M., Ter Hofstede, A.H.M., Mendling, J.: Configurable multi-perspective business process models. Information Systems, 36(2), pp. 313-340, 2011.