Architecture-based Change Impact Analysis in Information Systems and Business Processes

Kiana Rostami, Robert Heinrich, Axel Busch, and Ralf Reussner

Karlsruhe Institute of Technology
Karlsruhe, Germany
Email: {rostami, heinrich, busch, reussner}@kit.edu

***Abstract*—Business processes as well as software systems face various changes during their lifetime. As they mutually influence each other, business processes and software systems have to be modified in co-evolution. Thus, to adequately predict the change impact, it is important to consider the complex mutual dependencies of both domains. However, existing approaches are limited to analyzing the change propagation in software systems or business processes in isolation. In this paper, we present a tool-supported approach to estimate the change propagation caused by a change request in business processes or software systems based on the software architecture and the process design. We focus on the mutual dependencies regarding the change propagation between both domains. In the evaluation, we apply our approach to a community case study to demonstrate the quality of results in terms of precision, recall, and coverage.**

I. Introduction

Most organizations use enterprise Information Systems (IS) to support their Business Processes (BP). Longevity is one of the most important characteristics of both domains. Long-living systems have to be adapted due to changes during their life cycle. Thus, it is important to understand and predict the impact of a change to these systems. In modern organizations the IS is an integral part of the BP. Hence, there are mutual dependencies between both domains regarding their maintainability: On the one hand, IS has to be adapted to reflect the changes caused in the corresponding BP (e.g., supporting new activities in a BP can cause changes in the corresponding IS). On the other hand, a BP needs to be modified due to changes in the corresponding IS (e.g., replacing software components with components of alternative vendors can cause some services to the customers to be changed). We use a supermarket example to demonstrate the change propagation between both domains according to their mutual dependencies: Exchanging a two-dimensional barcode for product identification by another machine-readable representation of data such as RFID tags leads to changes in the BP (e.g., inventory of products), but also in the IS. Changing the IS again affects the ability of the cashiers who use the IS to sale products (e.g., to register the products which the customers buy). Thus, a BP and the corresponding IS are co-evolving. During the co-evolution a change to one of the both domains can propagate to the other domain due to these mutual dependencies. Thus, to adequately estimate the impact of a change in one domain we need to consider the impact of changes on the other domain. However, the mutual dependencies make the analysis of the change propagation challenging.

Business process modeling is used to better understand, structure, and analyze the processes of an organization. Hence, considering only the architecture of an IS or the design of a BP is not sufficient while analyzing the change propagation between the domain of IS and BP, as it neglects the effects of the corresponding domain. Thus, the co-design of BP and IS [1] plays an important role when considering the change propagation during the co-evolution. To reflect the co-evolution of the BP and the corresponding IS we need to metamodel both domains to represents their relationships [2]. Thus, the enterprise architecture [3] is the main artifact during the co-evolution, as it provides a holistic perspective of a BP and of the IS. It allows analyzing the change propagation between activities in an organization and the corresponding IS. To this end, the activities of a BP can be divided into activities that are performed completely manually by actors, called "actor steps" and activities that are performed completely automatically by the IS, called "system steps" [4]. Our analysis involves the process (organization layer), the software architecture (software layer), and the infrastructure layer of the enterprise architecture.

Existing approaches to the change propagation analysis in the IS (e.g., [5],[6]) are based on the architecture of the IS without taking the mutual dependencies to the BP into account. However, these mutual dependencies play a key role in the co-evolution, as changes to an IS highly impacts the activities of the BP and how the customers are able to meet their goals [7], [1]. Works on change impact analysis in BP consider change propagation from the metamodel of a BP to its instances (e.g., [8], [9]), in collaborative processes (e.g., [10], [11]), or a set of models of a BP at different abstraction levels (e.g., [12]). But they neither support the change propagation in one instance of a BP nor in co-evolution with an IS. Several approaches exist that support the analysis of the co-evolution process of BP and IS: There are approaches that describe the importance of considering the mutual dependencies between both domains (e.g., [1], [13]) or present two related metamodels (cf. [2]), but do not take the change propagation into account. Approaches to the change impact analysis in enterprise architectures (e.g., [14], [15]) describe a set of coarse-grained rules for the change propagation using the relationship between elements of both domains. However, they either do not support estimating the

change efforts as an activity list to implement a change request, or do not offer an automated change impact analysis.

In this paper, we present a tool-supported approach the Karlsruhe Architectural Maintainability Prediction for Business Processes (KAMP4BP), to automatically analyze the change propagation in a BP and the corresponding IS caused by an initial change request in one of the domains. Our approach is an extension of the Karlsruhe Architectural Maintainability Prediction (KAMP) [6]. KAMP is a scenario-based approach to calculate the change propagation but is limited to analyzing the IS. In contrast to KAMP, KAMP4BP calculates a more fine-grained change propagation for the IS based on the software architecture. In addition to structural changes in the IS, we consider the change propagation in the BP and between the BP and the corresponding IS to support the co-evolution of both related domains. Our approach results in a task list containing the estimated tasks to be carried out for the change. Thus, KAMP4BP supports change effort estimation and the decision making process by comparing the design alternatives. The contribution of this paper is threefold: i) We describe an approach that considers mutual dependencies between a BP and the corresponding IS regarding the maintainability. ii) We show an automated analysis of change propagation due to an initial change request in a BP and/or the corresponding IS. iii) We evaluate our approach using a community case study to demonstrate its applicability and its practical benefits. For the evaluation we use different change scenarios that represent several equivalence classes. Each equivalence class refers to model elements of the BP design and of the IS architecture that are relevant for the change propagation. Using these change scenarios we assess the quality of the task lists for implementing change requests in terms of precision, recall, and coverage.

The reminder of this paper is organized as follows: Sec. 2 summarizes the foundation of our approach. In Sec. 3, we introduce our running example to illustrate our approach. Sec. 4 gives an overview of the approach and the application of our approach to the running example. We present the evaluation of our approach in Sec. 5. Sec. 6 summarizes related work. The final Section concludes the paper and discusses future work.

II. Foundation

A. Change Propagation Analysis

When implementing a change request three sets of changing elements can be distinguished: i) *Change set* is a collection of elements, that are directly affected by a change request. ii) *Affected set* is a superset of the change set and contains all elements, that have to be changed to implement a change request.iii) *Impact set* is a superset of the affected set and includes all elements, that a change request potentially affects [16].

Our Approach extends the KAMP approach. KAMP is a scenario-based change propagation analysis approach for the IS, which derives an impact set to implement a change request. KAMP extends the Palladio approach by maintainability analysis of software architectures. Palladio [17] predicts the performance of an IS and is based on the Palladio Component Model (PCM). PCM is a domain-specific metamodel enabling software architects to model the architecture of a component-based IS especially regarding the performance quality attributes. Reussner et al. [17] defined a software component as a "contractually specified building block for software, which can be composed, deployed, and adapted without understanding its internals". A software component can be either a basic component or a composite component. A composite component is a composition of basic components and/or other composite components. Components provide or require interfaces. The relation of an interface to components defines its role. Thus, these relations are called provided or required roles, respectively [17].

In our approach, we extended the metamodels of PCM in a modular way, in order to allow metamodeling of maintainability of the BP and IS domain. Our metamodel extensions are based on a reference structure for metamodels for architecture description languages. This reference architecture can be divided into layers of paradigm, domain, quality, and analysis to ease model extension and customization [18].

B. Integrated Business IT Impact Simulation (IntBIIS)

The IntBIIS approach [4] extends PCM by metamodels of BPs regarding the performance impact of the BP and the corresponding IS. The BP metamodel extends the usage profile of PCM and consists of a set of linked activities, where an activity can be either an actor step or a system step, as illustrated in Fig 1 (cf. [4]). An actor step can be completed by an actor, whereas a system step is a call to a provided role of a component by the user, called entry level system call [17], [4]. However, neither Palladio nor IntBIIS, nor KAMP consider the maintainability of a BP and the corresponding IS in terms of change propagation.

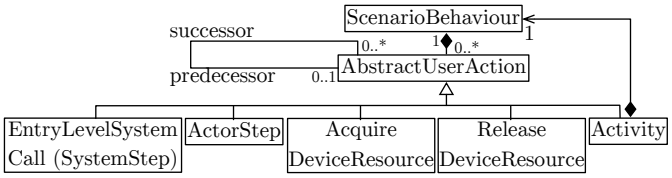

Fig. 1. Excerpt from PCM usage model and IntBIIS metamodel extension [4]

III. Example Scenario

In this section, we use the hybrid cloud-based Common Component Modeling Example (CoCoME) (cf. [20], [21]) as our running example to illustrate the idea of our approach.

A. System Overview

Fig. 2 shows a simplified component diagram of the IS architecture of the hybrid cloud-based CoCoME and the corresponding BP design of the sale scenario. CoCoME has been set up in a GI Dagstuhl research seminar as a demonstrator for component-based software engineering and serves as a community case study to which several approaches can be applied. Using CoCoME as a common demonstrator allows comparing the research results regarding evolution. CoCoME is a fully implemented application and represents a trading system of a supermarket and consists of a set of stores. Each

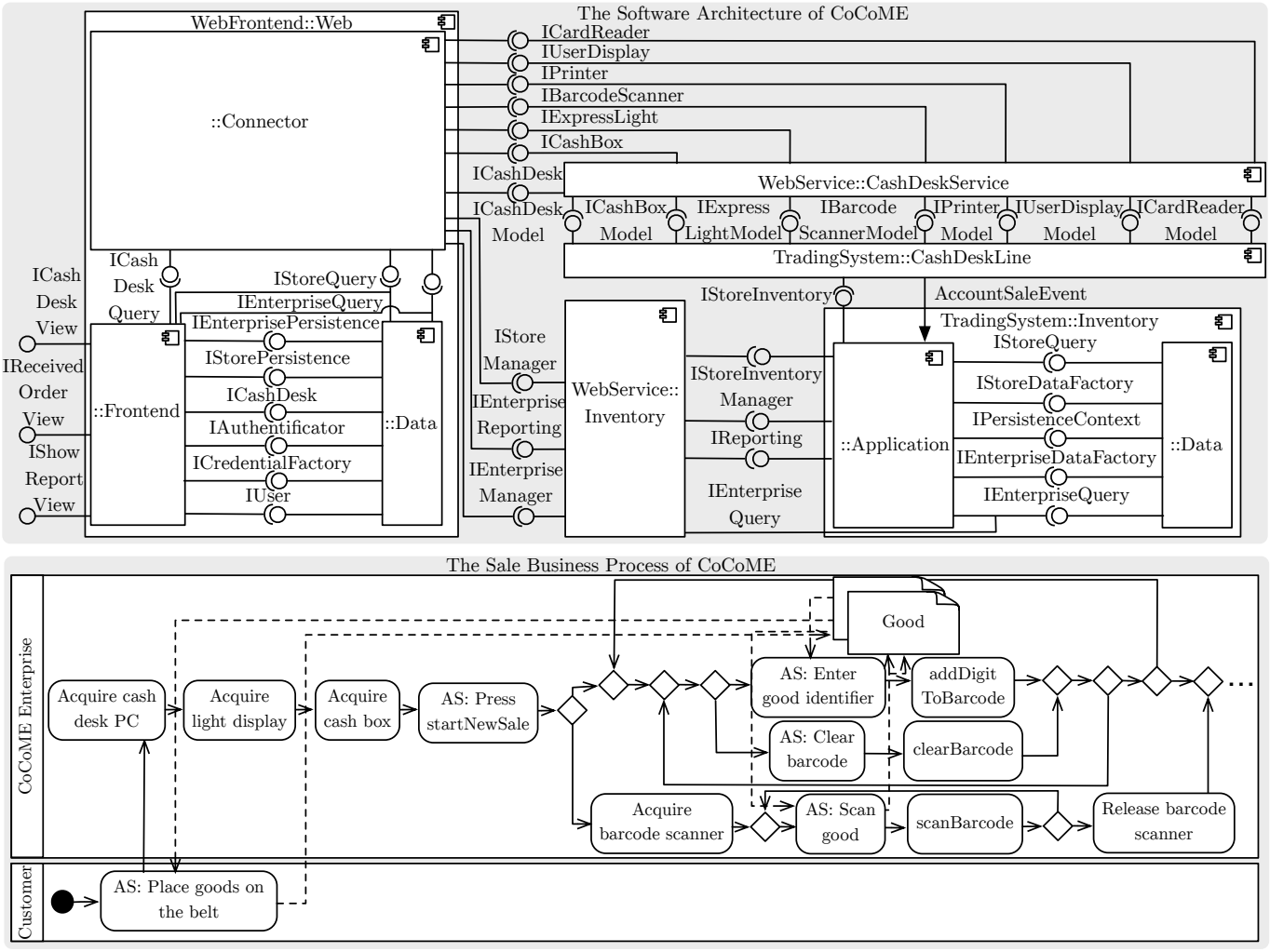

Fig. 2. Component diagram of CoCoME [19] and an excerpt from the adjacent sale business process

store has a store server controlling a cash desk line consisting a set of cash desks. A cash desk contains a computer and hardware resources (e.g., barcode scanner) connected to the computer. The store servers of all stores are connected to an enterprise store server [20], [19].

The IS architecture of the hybrid cloud-based CoCoME is a three-tier architecture and is composed of a composition of several composite components, namely `TradingSystem::Inventory`, `WebService::Inventory`, `TradingSystem::CashDeskLine`, `WebService::CashDeskService`, and `WebFrontend`. The `TradingSystem::Inventory` component comprises other composite components, namely `::Application` and `::Data`. They model the business logic and data layer of the three-tier architecture. The `WebFrontend` component provides a JavaServer-Faces-based user interface. The composite component `WebService::CashDeskService` provides the `WebFrontend` component access for the components of `TradingSystem::CashDeskLine`. The `TradingSystem::CashDeskLine` component is composed of `::CashDesk` components that in turn comprises all components at a cash desk such as `::BarcodeScanner` or `::CardReader`. The composite component `WebService::Inventory` provides `WebFrontend` access to enterprise components [19].

CoCoME supports several processes (e.g., sale or reporting). In the following, we use the main BP of CoCoME, namely the sale process. Fig. 2 illustrates an excerpt from the simplified sale process starting with placing goods on the belt of the cash desk. Actor steps are marked with `AS`. All other activities are system steps and represent the services of CoCoME. After a customer placed the goods on the checkout belt of the sale process, the cashier scans or types out the bar code. At the end, the customer pays by cash or by credit card. Finally, the stock will be updated due to successful sales [20].

B. Change Scenario

In order to illustrate how our approach automatically supports the change propagation between a BP and the corresponding IS, let us assume, that we have to speed up the scanning process according to a new requirement. Thus, we decide to modify

the business process by exchanging the machine-readable data representation of goods from barcode tags to RFID tags.

IV. Approach

Our approach calculates the change propagation in the BP design and the corresponding IS model caused by an initial change request to these domains. To this end, it considers the mutual dependencies between a BP and the corresponding IS.

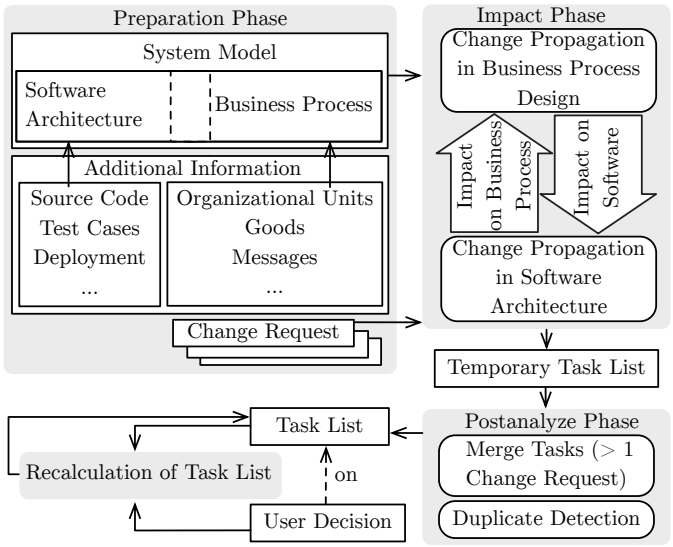

Fig. 3. Approach Overview

Fig. 3 gives an overview of KAMP4BP. KAMP4BP is composed of three phases: i) preparation phase, ii) impact phase, and iii) postanalyze phase. In the preparation phase, we model the design of the BP and the IS architecture. The impact phase automatically calculates the change propagation in the IS architecture model and in the model of BP design, and between both models. This phase results in a temporary task list representing an impact set to implement the change request. The postanalyze phase eliminates the duplicates and merges the task lists, if more than one change request was selected. Additionally, the approach allows users to reject single tasks referring to elements, that are not affected by the change. Based on decisions of the user KAMP4BP recalculates the task list. In the following, we refer to CoCoME to demonstrate the phases and the algorithms of KAMP4BP.

A. Preparation Phase

In the preparation phase, the design of the BP and the architecture of the corresponding IS are modeled. Model-based approaches can lead to a higher effort in the early development stages. However, it is expected, that this higher effort reduces the rework in the subsequent stages [22]. We can either model the architecture of the IS manually or automatically using a reverse engineering approach to reduce the effort (e.g., SOftware MOdel eXtractor (SoMoX) [23]). The BP design can be modeled using the extended usage profile of PCM as a set of actor steps and system steps [4]. Additionally, we annotate the model of the IS architecture or BP design with additional information regarding IS (e.g., test cases) or BP (e.g., organizational units). KAMP uses annotations related to IS to calculate the impact resulted by changing other elements than code (e.g., test cases) [6]. Our approach uses additional information in a BP to give its user feedback about elements, that are possibly affected (e.g., organizational units) or about the implicit impact due to changing some activities (e.g., training the employees).

In the next step, we select the initial change request(s) in the model of an IS architecture and/or the corresponding BP design. To avoid overestimation of the derived task list, only a minimal set of the initial change requests has to be selected. For example, if only one signature was modified initially, we have to select only the signature and not the whole interface. Further, deciding which element has to be initially modified in the IS architecture model and/or in the BP model due to an initial change request is not a trivial task and can be defined in different ways depending on the change scenario. Additionally, a change request can result in more than one initial changing element in the IS architecture model and/or the BP design model (hereafter referred to as "seed modification"). Further, the granularity of the model is an important factor for selecting the seed modification in the models.

Application to CoCoME: In the first step, we model the IS architecture and the design of the sale BP of CoCoME. In our change scenario, replacing barcode with RFID can be implemented in different ways. One possible seed modification can be replacing the barcode of all products in the BP with RFID tags and using RFID scanners instead of barcode scanners. However, replacing the barcode of all products can impact the model at different levels of granularity: i) If the physical `Barcode` has been modeled as a data object and the `Goods` have explicitly barcode tags, the `Barcode` data object is marked as modified. ii) If the `Barcode` is not explicitly represented in the BP model, maybe the data object `Goods` has to be marked as modified. For the application of our approach to CoCoME we explicitly modeled the physical `Barcode`. Therefore, the seed modification is the data object `Barcode` and the resource device `Barcode scanner`.

B. Impact Phase

In the impact phase, KAMP4BP automatically analyzes the propagation of the initial change requests in the BP and the corresponding IS. In particular, the change propagates from the modified elements of the model of the BP design and the architecture model of the IS to other elements in the models based on change propagation rules. When considering a BP, the corresponding IS, and their mutual dependencies change propagation analysis can be divided into three groups: i) Intra-domain change propagation between the elements of the BP (cf. Algorithm 1). ii) Inter-domain change propagation: from the IS to the BP and from the BP to the IS (cf. Algorithm 2) iii) Intra-domain change propagation between the elements of the IS (cf. Algorithm 3). In the following, we present the algorithms for the aforementioned groups. We describe the phases of each algorithm by the application to CoCoME.

1) Impact Analysis in BP: Algorithm 1 presents the analysis of the change propagation in the model of a BP design. The seed modification in a BP model can be device resources, roles, actor steps, or data objects. Device resources represent devices or machines needed for an activity or a sequence of activities (e.g., barcode scanner) [4]. In order to model a physical data object in a BP, we extend the PCM to include the data object. The data object concept is inspired by Business Process Model and Notation (BPMN) [24] but reflects only the physical elements which actors need as input or output to perform their actor steps. In our scenario, `Goods` is an example of a data object and represents the goods the customers bought. A physical data object can correspond to one or many data types and vice versa. For example a physical invoice can map to a table or to an object in a database. Further, a data object can be a composed data object or a collection data object. A collection data object can only include several data objects of the same type (i.e., collection of different invoices can only include invoices and no other types, such as goods.). Thus, a data object can be mapped either to a composed data type or to a collection data type. Each phase of Algorithm 1 describes how a change propagates from an element in a BP model to other elements in the BP model.

Application to CoCoME: If we apply the first phase of the Algorithm 1 to the sale BP model of CoCoME, a change to the data object `Barcode` propagates to the composite data object `Good` that contains the `Barcode`. After the composite data object `Good` was modified, the collection of `Goods` representing the purchases or the goods in the stock has also to be modified. Applying the second phase leads to changing the actor step `scan items` performed by the cashier, since this actor step uses `Goods` as input. By applying the final phase the change propagates from the device resource `Barcode scanner` to all related activities, such as `acquire barcode scanner` and `release barcode scanner`, depicted in Fig. 2.

2) Impact Analysis between BP and IS: Change propagation between the BP and the corresponding IS is calculated using Algorithm 2. The change between both domains can propagate due to the data objects or data types dependencies and system steps. If a data type is changed, the change can propagate to the corresponding data object and vice versa. As the system steps in the BP model are calls to the provided roles, a change to a signature in a provided role propagates to the system step of the BP model.

Application to CoCoME: As the data object `Barcode` and `Goods` were modified in the previous step, the second phase of Algorithm 2 marks the corresponding data types in the IS architecture model `Barcode` and `Products` as modified. Another example is the signature `addDigitToBarcode` of `ICashDeskView` that represents a system step (i.e., entry level system call). If the algorithm marks this signature as modified, the change would propagate from the IS architecture model to the corresponding model of the BP design.

3) Impact Analysis in IS: Algorithm 3 extends the KAMP algorithms for change propagation in an IS and refines the existing steps to enable the change propagation from the architecture model of an IS to the model of the corresponding BP design. The seed modifications in an IS architecture model are interfaces and their signatures, components, data types, event types, and event groups. KAMP analyzes the change propagation at the granularity of interfaces. However, to support the change propagation to and from the domain of the BP we extend KAMP to signature-based change propagation within a component and between two components (cf. the first phase of Algorithm 3 "Apply modified KAMP"). Additionally,

Algorithm 1 Algorithm for the Change Propagation Analysis in a BP Model

Require: Model of BP design, change request

1: Calculate the change propagation from a data object to the other data objects

while There is at least one new modified data object **do**
  Determine composed data objects that contain the modified data object. Mark them as modified.
  Determine collection data objects that contain the modified data object. Mark them as modified.
end while

2: Calculate the change propagation from data objects to actor steps using the modified data objects as input

3: Calculate the change propagation from a role to the corresponding actor steps

4: Calculate the change propagation from a modified actor step to the directly following actor step

while There is at least one new modified actor step A **do**
  Determine the actor step B directly following the modified actor step A.
  if The same data object serves as input of B and output of A **then**
    Mark B as modified.
  end if
end while

5: Calculate the change propagation from a modified actor step to the directly following system step

6: Calculate change propagation from modified device resources

Identify modified device resources.
for all Acquire-/ReleaseDeviceResource activities using these device resources **do**
  Mark Acquire-/ReleaseDeviceResource activities as modified.
  for all activities between two corresponding AcquireDeviceResource and ReleaseDeviceResource **do**
    Mark the activities between as modified.
  end for
end for

Algorithm 2 Algorithm for the Change Propagation Analysis between the Model of IS Architecture and BP and vice versa

Require: model of IS architecture and BP design, change request

1: Calculate the change propagation from data types to the corresponding data objects

2: Calculate the change propagation from data objects to the corresponding data types

3: Calculate the change propagation from modified signatures to the system steps (entry level system calls in the usage model)

4: Calculate the change propagation from modified system steps (entry level system calls in the usage model) to the signatures

KAMP4BP supports change propagation in the case of event-based communication between components, as in an event-based communication, the components can fire (i.e., source role) or handle (i.e., sink role) event groups, which are composed of single events (i.e., event types) [17]. Further, the algorithm supports change propagation analysis from a data type to the composite data type, collection data type, and event types.

Algorithm 3 Algorithm of the Change Propagation Analysis in IS Architecture Model

Require: IS architecture model, change request

1: Apply modified KAMP at the level of signatures
2: Calculate the change propagation from a modified data type to the other data types

while There is at least one new modified data type **do**
Determine composed data types that contain the modified data type. Mark them as modified.
Determine collection data types that contain the modified data type. Mark them as modified.
end while

3: Calculate the change propagation from a modified data type to the corresponding event types, that contain this data type
4: Calculate the change propagation from a modified event type to the corresponding event groups, that contain this event type
5: Calculate the change propagation from a modified event group to the corresponding components, that handle and fire this event group
6: Calculate the change propagation from a modified data type to the corresponding signatures, which use this data type in the input parameter or in the return type
7: Calculate the change propagation from signature to interface

if The signature is marked as modified **then**
Determine the interface that includes this signature. Mark it as modified.
end if

8: Calculate change propagation from interface to its signatures

if The whole interface is initially marked as modified **then**
Determine its signatures. Mark them as modified.
end if

9: Calculate the change propagation from a source role and/or a required role of a component to the corresponding provided and/or sink role of the same component
10: Calculate the change propagation from a sink role of a component to the corresponding source role of another component

Application to CoCoME: Using KAMP, the change to the data type `Barcode` and `Products` first propagates to provided and required roles of interfaces that use these data types. The rest of the Algorithm 3 calculates the change propagation due to interfaces, signatures, data types, or events to signatures, interfaces, events, or data types that use or relate to the changing elements. For example, the data type `ProductTO` has to be changed, as it composed of other changing data types, in particular the data type `Barcode` (cf. the second phase of Algorithm 3). Following the third phase of Algorithm 3, the event type `ProductBarcodeScannedEvent` has to be modified, since it uses the data type `Barcode`. Changing the event type `ProductBarcodeScannedEvent` requires to change the event group `handleProductBarcodeScannedEvent` of the `CashDeskEventHandler` component (cf. the fourth phase of Algorithm 3). Thus, according to the fifth phase, the component `CashDeskEventHandler` has to be changed in order to apply the changes in its interfaces. An example of the application of the sixth phase of Algorithm 3 is the change propagation to the `getBarcode` signature of the provided `ICashDesk` interface of `Web::Data::CashDeskData` component. `getBarcode` is modified due to the data dependency to the `Barcode` data type. `ICashDesk` contains at least one changing signature `getBarcode`. According to the seventh phase of the Algorithm 3, we need to change the interface `ICashDesk`. The `addDigitToBarcode` signature of `ICashDeskView` interface calls `getBarcode`. Thus, the refinement of KAMP results in changing `addDigitToBarCode` (cf. the first phase of Algorithm 3). The last three steps are not used for the change propagation in this example, but in the change scenarios in the evaluation.

The Algorithms 1, 2, and 3 are repeated as long as the algorithms result in new changes. The change propagation terminates if there are no new changes in the previous iteration. Thus, this phase of the approach predicts all elements of the IS architecture model and the BP design model which are potentially affected by the change request. The impact phase results in a temporary task list.

Application to CoCoME: Tab. I illustrates the output of KAMP4BP as a task list required to implement the initial change requests based on the CoCoME models.

C. Postanalyze Phase

If we select more than one change request as modified in the preparation phase, KAMP4BP merges the tasks of the temporary task list and removes duplicates in the post-analyze phase. Further, a task can refer to an element that cannot be modified (e.g., if a component is a third party component). As the element referred by the task can be the cause for the change propagation in other elements, the user has the possibility to mark this task as rejected. The approach can reanalyze the task list based on the user decisions and calculates a modified change propagation task list according to these decisions. The result of this phase is a merged task list.

V. Evaluation

KAMP4BP analyzes how a change request affects the model of a BP design and the model of the IS architecture. To this end, we consider the IS architecture and the BP design and the mutual dependencies between both domains. Using our approach, software architects and process designers can compare the design alternatives and the ways in which a change can be implemented. For the change impact analysis, first, the user has to model the BP design and the corresponding IS architecture and select the seed modifications in the models. Then, KAMP4BP analyzes the potentially affected elements of the models due to the seed modifications. The main contribution of our approach is the automatic identification of tasks required to implement a change request regarding the complex mutual dependencies between BP and IS. Additionally, a tool-supported approach allows the analysis of the change propagation in a

Initial Change Request 1: Device `barcode scanner` **Initial Change Request 2: Data Object `barcode`**
Propagation due to Data Dependencies
≡ Modify Interface `ICashDeskDAO`
≡ ≡ Modify Signature `enterBarcode`
≡ Modify Interface `ICashDesk`
≡ ≡ Modify Signature `getBarcode`
≡ ≡ Modify Signature `setBarcode`
≡ Modify Interface `ICashDeskQuery`
≡ ≡ Modify Signature `enterBarcode`
...
≡ Modify Data Type `Barcode`
≡ Modify Data Type `Product`
≡ Modify Data Type `ProductTO`
...
≡ Modify Data Object `Goods`
...
≡ Modify Actor Step `Scan Items`
...
Inter Business Process Propagation
≡ Modify `acquire barcode scanner`
≡ Modify `release barcode scanner`
≡ Modify System Step `addDigitToBarcode`
...
Change propagation due to Interface Dependencies
≡ Modify `CashDesk::CashDeskEventHandler`
≡ ≡ Modify sink `handleProductBarcodeScannedEvent`
≡ ≡ Modify required role `ICashDeskModelLocal`
≡ Modify `Web::Data::CashDeskData`
≡ ≡ Modify provided role `ICashDeskDAO`
≡ ≡ Modify provided role `ICashDesk`
≡ ≡ Modify required role `ICashDeskQuery`
...
Intracomponent Propagation
≡ Modify `Web::FrontEnd::CashDesk`
≡ ≡ Modify provided role `ICashDeskView`
≡ ≡ ≡ Modify signature `addDigitToBarcode`
≡ ≡ ≡ ...
≡ ≡ Modify required role `ICashDesk`
≡ ≡ ≡ Modify signature `getBarcode`
≡ ≡ ≡ ...
≡ ≡ Modify required role `ICashDeskDAO`
≡ ≡ ≡ Modify signature `enterBarcode`
≡ ≡ ≡ ...
...
...

TABLE I
Application to CoCoME: AN EXCERPT FROM THE TASK LIST FOR BARCODE

complex IS with a wide variety of elements, that is part of a large BP. Thus, KAMP4BP improves the traceability of modifications based on the IS architecture and BP design models.

We validate the prediction quality by applying our approach to the community case study CoCoME. To this end, we compare the results obtained from KAMP4BP to a reference task list.

A. Study Design

To evaluate KAMP4BP we follow the Goal Question Metric (GQM) plan. Our **Goal** is to evaluate the quality of the automatically derived task list compared to the reference task list to draw conclusions about the quality of the results. The reference task list is a manually created task list based on the code, the architecture model of hybrid cloud-based CoCoME and the model of the BP design. The reference task list contains only the elements which are actually affected by a concrete change request (i.e., the affected set). By contrast, the generated task list is the temporary task list without duplicates (i.e., an impact set). Further, we did not eliminate the elements in the postanalyze phase, which are not affected by the change request, in order to avoid to influence the results of the evaluation.

To evaluate the quality of the generated task list regarding false positives and false negatives, we define **Question 1**: How precise and complete are the automatically generated task lists? To answer this question in a measurable way we use **Metric 1** F_1 score (i.e., harmonic mean). F_1 score aggregates the recall and precision of the generated task list. After the comparison of the tasks in the generated task list with the tasks in the reference task list each task of the generated task list is classified into one of the following groups: i) A task of the generated task list is *true positive* when the task is actually changed by the change request and is contained in both task lists. ii) We call it a *false positive* if a task regarding a changed element is contained in the generated task list, but is missing in the reference task list (i.e., contained only in the impact set). iii) A task is classified as a *false negative* if a task regarding a changed element is contained in the reference task list, but is missing in the generated task list. Such a task is not contained in the impact set which the approach calculated. Given the number of true positives, t_p, the number of false negatives, f_n, and the number of false positives, f_p, we can calculate precision and recall as follows: $precision = \frac{t_p}{t_p+f_p}, \quad recall = \frac{t_p}{t_p+f_n}$. F_1 score as the harmonic mean of precision and recall can be calculated as follows: $F_1 = 2\,\frac{precision \times recall}{precision+recall}$.

To evaluate the coverage of the generated task lists, we formulate **Question 2**: Can our approach reduce the number of model elements software architects and process designers have to consider while analyzing the change propagation? To answer this question we define 2 metrics: **Metric 2.1** is defined as: The ratio, r_t, of the number of the actually changed model elements to the number of all model elements. Given the number of true positives (i.e., actually changed elements), t_p, and the number of all model elements, n, we can calculate Metric 2.1 as: $r_t = \frac{t_p}{n}$. **Metric 2.2** is defined as: The ratio, r_g, of the number of the elements referred in the generated task list to the number of all model elements. Given the number of elements in the generated task list, l, and the number of all model elements, n, we can calculate Metric 2.2 as follows: $r_g = \frac{l}{n}$.

B. Change Scenarios

As KAMP4BP is a scenario-based approach, we use different scenarios for validation. In the following, we discuss the quality of the generated task lists for six change scenarios for CoCoME. The change scenarios represent the equivalence classes referring to (PCM) metamodel elements of the IS architecture and BP design that are relevant for the change propagation. Further, the change scenarios were chosen with the focus of mutual dependencies between the BP and the corresponding IS. The change requests CR1 - CR5 involve the sale BP, whereas the

CR6 deals with the BP "show delivery reports" and "received ordered products". As mentioned in Sec. III-B mapping a changing request to a concrete element in the model may result in more than one initial change request in the system. Therefore, some of the following change scenarios involve more than one initial change request in the IS architecture model or the BP design model:

Change Scenario 1 (CR1): In the BP, barcode tags are replaced with RFID tags. Thus, the data object `Barcode` and the device resource `Barcode scanner` are initially affected in the model of the sale BP. This change request is already discussed in Sec. III.

Change request 2 (CR2): So far, the interfaces of CoCoME are designed in accordance to the predefined scenarios (e.g., "sale" process). In this scenario, we revise the implementation of the interface `ICashDeskView` and separate it to several interfaces due to different concerns. To this end, we need to change the interface `ICashDeskView`.

Change request 3 (CR3): The internal implementation of the `Tradingsystem::Inventory::Application::Reporting` component is replaced by a more efficient implementation without changing the interfaces.

Change request 4 (CR4): If the customers pay with credit card, they can sign for purchases instead of entering the pin code. The initial change request in the CoCoME architecture model is the event type `CreditCardPinEnteredEvent` and in the BP design model the actor step `Enter Pin`.

Change Request 5 (CR5): The cashier scans barcodes but does not need to enter the barcode manually. Therefore, we remove all relevant activities from the sale BP (i.e., system steps `addDigitToBarcode` and `clearBarcode` and actor steps `enterDigits` and `clearBarcode`).

Change Request 6 (CR6): The final change scenario considers changing of roles in the BP. To do so, we consider two further processes of CoCoME: the "received ordered products" process and the "show delivery reports" process. In the "received ordered products" BP the "stock manager" examines the new deliveries for correctness and updates the inventory. In the "show delivery reports" BP the "enterprise manager" creates reports. In this scenario, we merge both roles to one single role that is concerned with both responsibilities. Thus, the seed modification changes the roles of the "stock manager" and of the "enterprise manager".

C. Results

Tab. II summarizes the results of the evaluation. The comparison has been executed at a fine-grained level (e.g., at the level of signatures instead of interfaces).

	CR1	CR2	CR3	CR4	CR5	CR6
Recall	1.000	1.000	1.000	1.000	1.000	1.000
Precision	0.351	1.000	1.000	1.000	0.857	1.000
F_1 score	0.519	1.000	1.000	1.000	0.923	1.000
r_t	0.145	0.032	0.001	0.020	0.009	0.008
r_g	0.414	0.032	0.001	0.020	0.010	0.008

TABLE II
PRECISION, RECALL, AND F_1 SCORE FOR EACH CHANGE SCENARIO

In **CR1**, the change propagates to almost every element in the IS and BP. Thus, the precision and recall results are 0.351 and 1, respectively. The F_1 score is 0.519. This is caused by the number of false positives in the generated task list. However, in general, approaches to change impact analysis overestimate the impact set by generating more false positives. The generated task list does not contain any false negatives, which is most important. Further, this change scenario presents a fundamental change to the system, as many elements depend on the initially changing elements. Consequently, if a change to an element, such as a data type, results in several false positives in one of the first iterations, these false positives result in further false positives in the next iterations. To avoid the propagation of false positives, KAMP4BP allows software architects and process designers to select the false positives in the first iterations as "elements that are not changed" in the postanalyze phase. Based on the decision of the user KAMP4BP then recalculates the task list. For this evaluation, though, we used the temporary list in each domain without duplicates and did not change the results to evaluate the quality of the generated task list. Another important aspect is the ratio of the number of the elements that the user should consider to implement a change: The ratio of the number of the actually changed model elements, r_t, is 0.145, whereas the ratio of the number of the elements in the task list, r_g, is 0.414. Thus, the results show, even though the generated task list for this change scenario contains a high number of false positives, the generated task list reduce the number of model elements, which process designers and software architects have to consider while analyzing this change scenario, to only 41% of all model elements. In **CR2** - **CR6**, the ratio of the number of the actually changed model elements, r_t, shows, that the changes propagate to only a few number of elements. In these scenarios, KAMP4BP calculates false positives only in the CR5. Thus, the generated task lists for these scenarios are more precise. Similar to the first scenario, the task list does not include any false negatives.

Overall, the evaluation shows how the automated change impact analysis takes into account the mutual dependencies between both domains. The case study indicates that the approach tends to overestimate the impact set depending on the scenario under study. However, it did not provide any false negatives in the above scenarios. Further, the approach significantly reduces the number of elements which software architects and process designers have to consider in such change scenarios (cf. r_t and r_g in Tab. II). This in turn leads to a faster and a more realistic change impact analysis than manually impact analysis or in the case of the consideration of only one domain in isolation.

D. Threats to Validity

We considered four classes of validity [25]:

Internal validity: Our approach calculates changes based on PCM models. The accuracy and the granularity of such models can potentially affect the analysis results. Therefore, we compared the task lists generated from the models to reference task lists assembled from the source code to eliminate different

accuracies and granularities of the conducted models by interpreting the results. Consequently, parts of the models that are not implemented in code (e.g., connectors between interfaces and components) are omitted from the quality analysis of the task list. Further, a change request can be implemented in different ways. For our evaluation we used a reference task list for the implementation. Using another reference task list may lead to different results. However, different task lists for a change scenario involve similar elements.

External validity: In contrast to an experiment, a case study allows a better understanding of the phenomenon under study (i.e., the mutual dependencies between IS and BP regarding the maintainability) using a community case study (i.e., CoCoME). Thus, the results in our case study might not be transferable to other systems. However, we applied KAMP4BP to the hybrid cloud-based CoCoME, since this is a common system for case study applications and ensures high comparability of approaches [19]. Additionally, the case study allows for demonstrating the importance of considering the mutual dependencies between BP and IS.

Construct validity: KAMP4BP is based on models. Thus, the user needs to specify the activities of the BP model (e.g., actor steps or system steps) and the interfaces and signatures in the IS architecture model. However, re-engineering approaches (e.g., SoMoX) support a software architect to construct a model that can then be used in KAMP4BP. The change scenarios in our evaluation represent equivalence classes of all elements of the metamodels of the BP design and the IS architecture that are relevant for the change propagation. The goal of the evaluation is assessing the quality of the results of the change scenarios that leads to the change propagation between the IS architecture model and the BP design model. Thus, we selected change scenarios with the focus on the mutual impact on the IS architecture and the BP design models.

Conclusion validity: To exclude the interpretation effects by a specific researcher, we used statistical metrics to evaluate the quality of the results.

E. Assumption and Limitation

KAMP4BP derives a task list involving all elements in the IS architecture and BP design models that are potentially affected by an initial change request (i.e., impact set). However, only a subset of the impact set (e.g., depending on the implementation of the IS) needs to be modified (i.e., affected set). Additionally, there is a trade-off between the coverage of change propagation rules and the quality of the results. If the rules are more general and cover many cases, in which changes propagate in specific circumstances, more false positives can be expected. In contrast, if we have a small set of rules that covers only a few cases, in which changes probably propagate, the results potentially contain more false negatives. If the results contain more false negatives, users have to examine the whole BP and the relating IS, as they cannot predict an accurate impact set using only the approach. Thus, to avoid false negatives, KAMP4BP tends to overestimate the change propagation, too. Overestimation leads to more false positives. In general, approaches to change propagation analysis overestimate the impact set.

VI. Related Work

Related work to our approach can be divided into three categories: i) change propagation in IS, ii) change propagation in BP, and iii) co-evolution of IS and BP.

A. Architecture-based or Scenario-based Software Evolution

The approach of Carbon [5] analyzes software evolution based on IS architectures. However, it does not consider automated change impact analysis automatically. Approaches, such as Software Architecture Analysis Method (SAAM) [26], Architecture-Level Modifiability Analysis (ALMA) [27], Architecture-Centric Project Management (ACPM) [28] or KAMP [6] analyses the change propagation in the software architecture and architecture evolution based on scenarios. However, all these approaches support only the architecture of the IS and do not consider the change propagation in the BP and the mutual dependencies between the BP and the IS.

B. Evolution of Business Processes

In the domain of BP, Weber et al. [29] identified a set of fine-grained change patterns. There are several approaches addressing the problem of change propagation from the metamodel of a BP to its instances [8], [9], [30], [31], [32]. However, they do not consider a scenario-based change propagation in a BP model. Approaches to change propagation in collaborative BP (cf. [11], [10], [33], [34], [35]) study the impact of a change to a BP on the related BP. In contrast to the previous approaches, Weidmann et al. [12] analyze the change propagation between a set of BP models at different abstraction levels but referring to the same BP. The approach of Lezoche et al. [36] consider inconsistencies in models of a BP, after the BP rules changed. But, all of these approaches lack on an impact analysis within each BP model based on an initial change request. Works on change impact in the BP, however, do not consider the IS architecture and its mutual dependency to the BP design.

C. Co-evolution of Software Systems and Business Processes

There are several approaches (cf. [1], [13]) describing the importance of consideration of the mutual dependencies between the BP and the IS while analyzing the change propagation, without considering the change impact analysis in the domains. Warboys et al. [2] present two related metamodels as the solution of the problem of co-evolution of the BP and the IS. However, the approaches do not support an automated analysis of change propagation. Vanderfeesten et al. [37] consider the similarities between the BP and the IS, however neglect the impact propagation analysis. Aysolmaz and Demirors [38] presented an approach to extract the requirements and the software size based on the BP models. Jamshidi and Pahl [39] proposed fine-grained change patterns in the IS architecture resulting from changes in the BP using graphs, but they neglect the impact of changes in each models. Works on change impact analysis in enterprise architectures (cf. [14], [15], [40]) describe

a set of coarse-grained rules for change propagation using the relationship between elements of both domain. But, they either do not support estimating change effort or do not offer automated impact analysis or derivation of change activities.

VII. Conclusion and Future Work

In this paper, we presented KAMP4BP to automatically analyze the change propagation between the model of a BP design and the corresponding IS architecture model based on the mutual dependencies between both domains. Our approach comprises three phases: i) In the preparation phase, the design of a BP and the architecture of the corresponding IS can be modeled and annotated with context information. ii) The impact phase automatically analyzes the change propagation within the model of the IS architecture, the BP design, and between both models. iii) In the postanalyse phase, our approach eliminates the duplications and considers user decisions. The result of KAMP4BP is a task list needed to implement a change request. Process designers and software architects can use this task list to analyze the impact of changes from an IS to the corresponding BP and vice versa. This prediction can then be used to estimate the maintenance cost of a change request. That leads to a better comparison of different design decisions.

Based on CoCoME, we presented how KAMP4BP helps calculating the change propagation with regard to the mutual dependencies between the domain of IS and the domain of BP. The results showed that our approach provides precise task list representing the holistic impact of a change request. Additionally, KAMP4BP improves the traceability of modifications in a complex IS, that is part of a large BP.

As future work, we plan to support change propagation from requirements and design decisions to the domain of IS and BP. Additionally, we will extend the approach to other domains (e.g., manufacturing systems). Further, the idea of the approach could be extended to a cost estimation approach.

Acknowledgment

This work was partially supported by the DFG (German Research Foundation) under the Priority Programme SPP1593 and the MWK (Ministry of Science, Research and the Arts Baden-Württemberg) in the funding line Research Seed Capital (RiSC). We would like to thank Jakob Bach for inspiring discussions and support during conception and tool development.

References

[1] K. Liu, *et al.*, "Co-design of business and it systems - introduction by guest editors." *ISF*, vol. 4, no. 3, pp. 251–256, 2002.

[2] B. Warboys *et al.*, *Systems Engineering for Business Process Change*. Springer, 2000, ch. Modelling the Co-Evolution of Business Processes and IT Systems, pp. 10–23.

[3] R. Winter and R. Fischer, "Essential layers, artifacts, and dependencies of enterprise architecture," in *EDOCW'06*. IEEE, 2006, pp. 30–.

[4] R. Heinrich *et al.*, "Integrating business process simulation and information system simulation for performance prediction," *SoSyM*, 2015.

[5] R. Carbon, "Architecture-centric software producibility analysis," Ph.D. dissertation, 2012.

[6] K. Rostami *et al.*, "Architecture-based assessment and planning of change requests," in *QoSA'15*. ACM, 2015, pp. 21–30.

[7] N. Chapin *et al.*, "Types of software evolution and software maintenance," *JSM*, vol. 13, no. 1, pp. 3–30, 2001.

[8] S. Yoo *et al.*, "Rule-based dynamic business process modification and adaptation." in *ICOIN*. IEEE, 2008, pp. 1–5.

[9] S. Sadiq and M. Orlowska, "Architectural considerations in systems supporting dynamic workflow modification," in *SABPM'99*, 1999.

[10] W. Fdhila *et al.*, "Dealing with change in process choreographies: Design and implementation of propagation algorithms," *Inf. Syst.*, vol. 49, 2015.

[11] S. Rinderle *et al.*, "On the controlled evolution of process choreographies," in *22nd Intern. Conference on Data Engineering*, p. 124.

[12] M. Weidmann *et al.*, "Business process change management based on process model synchronization of multiple abstraction levels," in *SOCA'11*, 2011, pp. 1–4.

[13] A. Aerts *et al.*, "Architectures in context: On the evolution of business, application software, and ict platform architectures," *IJIM*, vol. 41, no. 6, pp. 781–794, 2004.

[14] T. Bodhuin *et al.*, "Impact analysis for supporting the co-evolution of business processes and supporting software systems." in *CAiSE Workshops (2)*. FCSIT, Riga TU, 2004, pp. 146–150.

[15] S. Sunkle *et al.*, *Analyzing Enterprise Models Using Enterprise Architecture-Based Ontology*. Springer, 2013, pp. 622–638.

[16] S. A. Bohner, "Extending software change impact analysis into cots components," in *SEW-27'02*. IEEE Computer Society, 2002, pp. 175–.

[17] R. Reussner *et al.*, *Modeling and Simulating Software Architectures - The Palladio Approach*. MIT, 2016.

[18] M. Strittmatter *et al.*, "A modular reference structure for component-based architecture description languages," in *ModComp*. CEUR, 2015.

[19] R. Heinrich *et al.*, "The CoCoME platform for collaborative empirical research on information system evolution," KIT, Tech. Rep., 2016.

[20] S. Herold *et al.*, "The common component modeling example." Springer, 2008, ch. CoCoME-The Common Component Modeling Example.

[21] R. Heinrich et al., "The CoCoME platform: A research note on empirical studies in information system evolution," *SEKE*, vol. 25, no. 9-10, 2015.

[22] H. Koziolek, "Parameter dependencies for reusable performance specifications of software components," Ph.D. dissertation, University of Oldenburg, 2008.

[23] K. Krogmann, "Reconstruction of software component architectures and behaviour models using static and dynamic analysis," Ph.D. dissertation, KIT, 2010.

[24] Object Management Group, "Business Process Model and Notation (BPMN) Version 2.0," Tech. Rep., 2011.

[25] P. Runeson, M. Host, A. Rainer, and B. Regnell, *Case Study Research in Software Engineering: Guidelines and Examples*, 1st ed. Wiley, 2012.

[26] P. Clements *et al.*, *Evaluating Software Architectures: Methods and Case Studies*. AW, 2002.

[27] P. Bengtsson *et al.*, "Architecture-level modifiability analysis (ALMA)," *J. Syst. Softw.*, vol. 69, no. 1-2, pp. 129–147, 2004.

[28] D. J. Paulish and L. Bass, *Architecture-Centric Software Project Management: A Practical Guide*. AW, 2001.

[29] B. Weber *et al.*, "Change patterns and change support features - enhancing flexibility in process-aware information systems," *DKE*, vol. 66, no. 3, pp. 438–466, 2008.

[30] M. Reichert *et al.*, "Adaptive process management with adept2," in *21st Intern. Conference on Data Engineering*. IEEE, 2005, pp. 1113–1114.

[31] S. Sadiq *et al.*, "Managing Change and Time in Dynamic Workflow Processes," *IJCIS*, pp. 93–116, 2000.

[32] M. Kradolfer and A. Geppert, "Dynamic workflow schema evolution based on workflow type versioning and workflow migration," in *CoopIS'99*, 1999, pp. 104–114.

[33] T. A. Kurniawan *et al.*, "Relationship-preserving change propagation in process ecosystems," in *ICSOC'12*. Springer, 2012, pp. 63–78.

[34] S. Rinderle *et al.*, "Evolution of process choreographies in DYCHOR," in *OTM'06*, 2006, pp. 273–290.

[35] M. Weidlich *et al.*, "Change propagation in process models using behavioural profiles," in *IEEE SCC*, 2009.

[36] M. Lezoche *et al.*, "Business process evolution: a rule-based approach," in *BPMDS'08*, 2008.

[37] I. Vanderfeesten *et al.*, "Quality Metrics for Business Process Models," in *BPM and Workflow Handbook 2007*.

[38] B. Aysolmaz and O. Demirors, "Modeling business processes to generate artifacts for software development: A methodology," in *MiSE'14*. ACM.

[39] P. Jamshidi and C. Pahl, "Business process and software architecture model co-evolution patterns," in *MiSE'12*. IEEE, 2012, pp. 91–97.

[40] O. Avila and K. Garcés, *Change Management Contributions for Business-IT Alignment*. Springer, 2014, pp. 156–167.