A Business Process-Driven Approach for Requirements Dependency Analysis

Juan Li[1], Ross Jeffery[2,3], Kam Hay Fung[4], Liming Zhu[2,3], Qing Wang[1], He Zhang[2,3], and Xiwei Xu[2]

[1] Institute of Software, Chinese Academy of Sciences, China
[2] National ICT Australia, Australia
[3] School of Computer Science and Engineering, The University of New South Wales, Australia
[4] School of Information Systems, Technology and Management, The University of New South Wales, Australia

{lijuan,wq}@itechs.iscas.ac.cn,
{Ross.Jeffery,Liming.Zhu,He.Zhang,Xiwei.Xu}@nicta.com.au,
kamhayfung@ieee.org

Abstract. Dependencies among software artifacts are very useful for various software development and maintenance activities such as change impact analysis and effort estimation. In the past, the focus on artifact dependencies has been at the design and code level rather than at the requirements level. This is due to the difficulties in identifying dependencies in a text-based requirements specification. We observed that difficulties reside in the disconnection among itemized requirements and the lack of a more systematic approach to write text-based requirements. Business process models are an increasingly important part of a requirements specification. In this paper, we present a mapping between workflow patterns and dependency types to aid dependency identification and change impact analysis. Our real-world case study results show that some participants, with the help of the mapping, discovered more dependencies than other participants using text-based requirements only. Though many of these additional dependencies are highly difficult to spot from the text-based requirements, they are however very useful for change impact analysis.

Keywords: Business process modeling, workflow pattern, software development and maintenance, requirements dependency.

1 Introduction

In a volatile environment, software systems must evolve to adapt to the rapid changes of stakeholders' needs, technologies and the business [1]. A change can impact not only source code, but also other software artifacts, such as requirements, design and test cases [2]. To analyze the impact of a proposed software change, one should determine which parts of the software system may be affected by the change and ascertain their possible risks [3]. Bohner [3] proposed an impact analysis process that examines change requests to identify the Starting Impact Set (SIS) of software artifacts that could be affected by these requests. The SIS is then analyzed to identify other artifacts to be affected, which are then incorporated into SIS to form the Candidate Impact Set (CIS). His process's goal is to estimate a CIS that is as close as possible to the set of artifacts that is actually modified after all the changes are made.

A CIS can be determined starting from the requirements specifications affected by the changes to the source code level [4] through information about traceability [5]. A traceability link is "any relationship that exists between artifacts involved in the software-engineering life cycle" [6]. A link can be between artifacts in different models (e.g. requirements and code) or between artifacts within a model. Requirements dependency, an example link, characterizes the relationship between requirements within a requirement model and acts as a basis from which a CIS is analyzed. The CIS include not only requirements to be affected directly by the change requests but also requirements to change potentially. To improve the accuracy of a CIS, it is useful to associate semantics with traceability links [3], such as the use of requirement dependency types since they convey important information for change impact analysis. Requirements dependency discovery tends to require a significant effort especially when the set of requirements is large. Studies have explored ways to automate the discovery (e.g. [7]). However, the solutions to date are immature and human experts are still relied upon for a large set of requirements [8].

Traditionally text is the primary (or even only) means of documenting requirements specification. Our previous empirical evaluation [9] found that many dependencies, which are especially useful for change impact analysis, were not spotted during dependency discovery in text-based requirements alone. An explanation for this is that even when they were closely related to business processes and rules, they could not be represented explicitly in text-based requirements.

It is no doubt that a diagram is worth ten thousand words [10] and hence Business Process Modeling Notation (BPMN) based models [11] combined with text-based requirements should increase the likelihood of finding more dependencies than the latter alone in requirements dependency analysis. In this research, however, we are interested in *how* BPMN models can help requirements dependency analysis and change impact analysis in a more concrete fashion. In this spirit, we propose a mapping between workflow patterns in BPMN and dependency types to supplement text-based requirement dependency analysis by systematically and manually deriving dependencies from a typical business process model. We conducted a case study on a real-world industrial project to evaluate our approach and confirmed that practitioners using our approach did discover additional dependencies useful to change impact analysis.

2 Related Work

2.1 Dependency and Change Impact Analysis

Research in traceability is gaining attention in requirements engineering [12]. Correct traceability is the basis for change propagation analysis [13], important for all aspects of a software development project. In requirements engineering, requirements relationships are classified into dependency types based on the structural and semantic properties of requirements, to help practitioners identify these relationships (e.g. [14]). Requirements dependencies also play an important role in change impact analysis. Hassine et al. applied dependency analysis at the use case map level (rather than between requirements in natural languages) to identify the potential impact of requirement changes on the overall software system [12]. Yan et al. discussed the ripple-effect of requirements evolution based on requirements dependencies [15].

2.2 Business Process Modeling in Requirements Engineering

Business process models (BPMs) are widely used in requirements engineering. To the best of our knowledge, however, no studies have considered applying BPMs in requirements dependency discovery to facilitate change impact analysis. For instance, to bridge business process modeling with requirements elicitation and analysis, de la Vara González and Díaz [16] described a business process-driven requirements engineering approach to derive requirements from organizational models that express business strategies and from business processes in BPMN. Cardoso et al. [17] used business process models to derive alternative sets of requirements for a process-oriented software system. These sets capture different decisions regarding the intended "level of automation" for various activities in a business process. Mathisen et al. [18] presented an approach for early detection of structural changes that have implications for the software architecture. Their approach hinges on using business process modeling to increase the level of understanding of the problem domain in early stages of a project.

3 Requirements Dependency Analysis Based on Business Process Models

Before delving into the details of our approach, we use a stimulating example to highlight its use. In Fig. 1, a simple BPM for processing home loan applications consists of four sequential processes as shown. Let us focus on the middle two processes, viz. "Check Credit" and "Approve Loan". The former performs credit checking on loan applicants and fulfills three requirements (UC1, UC2 and UC3). The latter lets a loan assessor approves loan applications and involves two requirements (UC4 and UC5). These two processes form a *sequence* workflow pattern in that "Check Credit" precedes "Approve Loan". The pattern logically *connects* requirements UC1, UC2 and UC3 to UC4 and UC5. In the context of dependency, UC1, UC2 and UC3 are regarded as *preconditions* for UC4 and UC5. More generally, the power of BPMs connects individual text-based requirements into a manageable and well-scoped visual structure. The connections are often useful guidance for implementation-related dependency identification. Sometimes it is easy to spot preconditions based on the textual descriptions of the requirements without the help of BPMs. To calculate how much an applicant can borrow (UC5), one first needs to know how much the person owes (UC2) and his/her income level (UC3). At other times, dependencies are non-trivial from textual descriptions whence BPMs may provide some hints (e.g. connecting UC1 and UC4).

The example just showcased that BPMs can be useful for dependency discovery. Now we present the formal definitions of the dependency type model plus the mapping between workflow patterns and dependency types.

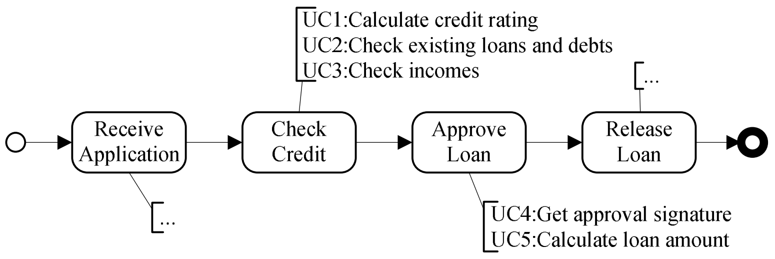

UC1:Calculate credit rating
UC2:Check existing loans and debts
UC3:Check incomes
...
Receive Application
Check Credit
Approve Loan
Release Loan
...
UC4:Get approval signature
UC5:Calculate loan amount

Fig. 1. BPM example

3.1 Requirements Dependency Model

Many dependency types have been proposed and they have different levels of abstraction and different criteria for categorization. Their complexity and diversity gives rise to a steep learning curve, which contributes to the difficulty in comprehending, using and evaluating them. Thus, in earlier work [9], we surveyed dependency types from the literature, consolidated them into a requirements dependency model and empirically evaluated its applicability in dependency identification and change impact analysis in a real-world industry project. Here we divide its twenty-three dependency types into three categories:

- *Document-related*: This category of dependency types is embedded in the structure, content and version relationships in the requirements representation. For instance, a requirement can have a "formalizes" dependency on another requirement, which means the former is defined more formally (using computational logic, business rules, constraints, etc.) than the latter.
- *Value-related*: This category is concerned with the relation between the realization of one requirement to the value that a customer/user perceives the realization of another requirement will provide [9]. For example, the adoption of a complex user-interface style can increase the usability of the user interface for a web-based application (i.e. "increases_cost_of" dependency type). This category can be useful for selecting the set of requirements to be fulfilled in release planning.
- *Implementation-related*: In this category, the realization of a requirement relates to one or more requirements. For instance, the requirement "withdraw cash" calls for the requirement "calculate account balance" to be realized to determine the withdrawal limit for a bank account. Dependency types in this category often indicate change propagations among low level models such as the detailed design and the code, which are important for change impact analysis.

The dependency types in each category are listed in Table 1. In this study, we concentrate on the implementation-related dependency types because they are the most useful and relevant to software design and development. Due to space limitations, we do not elaborate further on other dependency types. Interested readers may refer to [9] for the in-depth discussions. The implementation-related dependency types are:

- *Constraints*: A requirement can relate to another by being a constraint to the latter. For instance, the requirement "cash withdrawal is limited to $2000 daily" is a constraint for the requirement "withdraw cash".
- *Refines*: One requirement can be a refinement of another requirement, providing more detailed descriptions for the latter.
- *Precondition*: Only after one function prescribed by a requirement is finished, or one condition described by the requirement is satisfied, that another function prescribed by a requirement can be performed. Usually precondition reflects the business rules or the sequence relationship between (sub-)processes. For instance, the requirement "a customer successfully logged in to the application" is a precondition for "a customer withdraws cash from bank account".
- *Satisfies*: This type expresses that if one requirement is implemented in an application, the implementation will also satisfy another requirement. For example, the requirement "when a user logs out, all opened documents will be automatically saved" satisfies the requirement "no unsaved work will be allowed before the editor terminates".
- *Similar_to*: The prescription of one requirement (e.g. "display account balance") is similar to or overlapping with one or more requirements (e.g. "display amount available for withdrawal").

Table 1. Dependency Type Model [9]

Category	Dependency types
Document-related	Compares, Contradicts, Conflicts, Example_for, Test_case_for, Purpose, Comments, Background, Replaces, Based_on, Elaborates, Generation, Changes_to, Formalizes
Value-related	Increase_cost_of, Decreases_cost_of, Increase_value_of, Decreases_value_of
Implementation-related	Constraints, Refines, Precondition, Satisfies, Similar_to

3.2 Business Process Driven Dependency Identification

In recent years, BPMs are increasingly used in requirements analysis to define system goals and requirements. We believe BPMs can also be used to discover missing and ambiguous requirements in requirements specifications. In our approach, we adopted BPMN as the business process modeling language because it is a widely used business process modeling language and covers most of the workflow patterns compared to other modeling languages.

A well-known collection of Workflow Patterns was proposed in [19, 20]. This work provides a comprehensive examination of the various perspectives (control flow, data, resource, and exception handling) of workflows. Each workflow pattern precisely defines recurring semantics between some process elements. Using the evaluation of BPMN version 1.0 against the workflow control-flow patterns [21], we map all relevant workflow patterns supported by BPMN models to all the implementation-related dependency types. The mapping is an informal one with no mathematic rigor as it is targeted for business analysts and requirements engineers to identify informal dependencies. We derived the mappings largely from the expert opinions and our observation in the previous case study [9]:

- Certain dependency types are often associated with certain workflow patterns. For example, we observed empirically that Precondition dependency types are often associated with workflow patterns such as the sequence workflow pattern.
- Some workflow patterns are similar to each other in terms of their associations with the dependency types. For example, sequence, merge/split-related workflow patterns are too often pointing to the Precondition dependency type. Thus, we created more generic definitions for these similar patterns to further reduce the complexity of the mapping.

The mapping only acts as an informal guidance and is highly indicative. There are many-to-many relationships between the workflow patterns and the dependency types. It is up to practitioners to confirm each instance of dependency identification in a particular project. We acknowledge the limitation and imprecision of the mapping but consider it helpful in the task as demonstrated by the later evaluation. We also excluded the workflow patterns on data, resource and exceptional handling in this study due to the scope. However, we believe they will also be helpful in identifying requirements dependencies and we plan to investigate them in future work. The mapping results are summarized in Table 2.

Table 2. Mapping between workflow patterns and dependency types

Workflow Patterns [19, 20]	Relevant Generic Pattern	Dependencies
sequence, split-related, merge-related, triggers	Generic Sequence	Precondition
merge-related	Generic Merge	Similar_to
split-related	Generic Split	Similar_to
cancelling/termination-related, interleaved routing, thread merge	Generic Crosscut	Constraints
Individual workflow patterns that provide more details to the decision points logic, sub-process	Generic Detailing	Refines, Satisfies

Generic Sequence

A (sub-)process is enabled after the completion of a preceding (sub-)process. For example, a bank allocates a valuation task to a valuer after a client has submitted the valuation request. Many workflow patterns, such as split-related, merge-related and triggers [19, 20], indicate a precedence-successor relationship and are therefore grouped under it. Due to the large number of workflow patterns involved, we do not list all of them here. Although this is one of the most obvious patterns, the textual descriptions of this pattern's instances are often scattered around or only implied in text-based requirements, resulting in the dependencies not being found. This is a basic pattern which may be included in the other workflow patterns such as split-related and merge-related patterns. The corresponding dependency type for Generic Sequence is "Precondition" (e.g. A is a precondition of B in Fig. 2(a)).

Generic Split

This represents the divergence of a (sub-)process into two or more (sub-)processes (i.e. branches). For example, after a client has proposed property valuation requests, one or more of the "desktop valuation", "curbside valuation" or "full valuation" can be performed. This generic pattern includes all the workflow patterns for choices and splits, such as parallel split, synchronization, exclusive choice, multi-choice, thread split [19, 20]. In some of the splitting instances, the branches serve a similar purpose (Similar_to each other) and only one or more of them are selected. However, the textual descriptions may look very different and are again scattered within requirements. Our case study described later on is one such example. Practitioners missed some of the similar relationship among the text-based requirements but they were easily identified in the BPMN models. The corresponding dependency type for Generic Split is "Similar_to" (e.g. B and C are similar to each other in Fig. 2(b)).

Generic Merge

This represents the convergence of two or more (sub-)processes (i.e. branches) into a single subsequent (sub-)process. For example, "prepare invoice" and "send for approval" happen simultaneously and join before "send report". This generic pattern covers all the workflow patterns for synchronization, merge and join and discriminators, such as structured synchronizing merge, multi-merge, thread merge [20]. The rationale behind this pattern is similar to the Generic Split pattern. For example, task "prepare valuation reports" can be started if and only if all the valuation results are returned. An example of change request is checking the quality of the reports before allowing an administrator to print them. This may result in checking all the valuation results before they are returned. The corresponding dependency type for Generic Merge is "Similar_to" (e.g. A and B are similar to each other in Fig. 2(c))

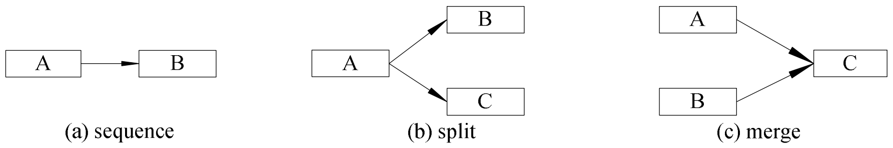


Fig. 2. Workflow patterns used in the mapping

Generic Crosscut

The workflow patterns grouped under this category are cancelling, thread merge and interleaved routing [20]. Cancelling-related workflow patterns (cancelling discriminator, cancelling partial join, etc.) abort the execution of other (sub-)processes and so crosscut many of these (sub-)processes. BPMs help identifying cross-cutting impacts of exceptional events such as Canceling. In BPMN models, for example, all the preceding (sub-)processes of a cancellation point can act as a source of dependency candidates narrowing down the scope of possibly affected requirements. These dependencies are difficult to identify across text-based requirements. Thread merge and interleaved routing workflow patterns concern the control of execution threads (i.e. multiple (sub-)processes) at runtime. The information they specify crosscuts multiple requirements and is often not captured in text-based requirements. The corresponding dependency type for Generic Crosscut is "Constraints".

Generic Detailing

All decision points in BPMN models have precise meanings. For example, a split can mean running in parallel, an exclusive choice or a multiple choice. In text-based requirements, such precise meanings are often elaborated separately and it is often difficult to spot the dependency between the elaboration and its original requirements. All such decision points related patterns also help tremendously in terms of identifying missing and/or ambiguous requirements. The corresponding dependency types are "Refines" and "Satisfies". The sub-process notations in BPMN also have similar meanings with respect to and are obvious helpers for identifying these two dependency types.

The same type of dependencies identified in BPMs often has different nature from those identified in text-based requirements. For example, Similar_to mostly refers to task similarity in BPMs but data similarity is often identified in text-based requirements. In BPMs, Similar_to can exist in the generic merge pattern where several (sub-)processes join to proceed to the next (sub-)process. These (sub-)processes may have some similarity in talking to the next activity. But in text-based requirements, Similar_to can exist between many functional requirements dealing with the same data information. Another example is Constraints. It indicates constraints that non-functional requirements, such as security-related non-functional requirements, have on functional requirements. These new dependencies are usually not identified in text-based requirements but very useful to impact analysis.

3.3 Usage Guidelines

While the mapping is useful for identifying dependencies in requirements, it is expected to be used in a larger framework. For instance, one can define a mini-process for requirements elicitation for business-process oriented applications as below:

1. Develop text-based requirements and supplement them with BPMs. They cover business-level activities, business goals and business rules.
2. Identify dependencies among requirements. This makes use of our mapping between workflow patterns and dependency types to help analyze requirements dependencies.
3. Perform coverage checking between BPMs and text-based requirements so that missing and ambiguous text-based requirements can be discovered and resolved along with dependency analysis.

Our work presented in this paper contributes to Step 2 above, relying on availability of BPMs. Steps 1 and 3 have their own complexities and we do not elaborate them further because of space limitations. Note that the mapping involves overheads, which includes learning the dependency types and the mapping, eliciting detailed BPMs and applying the mapping. However, these can be offset with the productivity gained from identifying dependencies and a more accurate CIS (since more impacted requirements are found), both of which exemplified by the case study in the next section.

Note that while the mapping can be useful for identifying the dependencies of and hence linking text-based requirements, mapping text-based requirements onto BPM structures and vice versa is an additional problem in itself. For instance, a security requirement can cross-cut several parts of an application and may be linked to many parts of a BPM. Due to space limitations, this issue is deemed out of scope.

4 Case Study

4.1 Questions to Evaluate and Case Selection

To evaluate our approach, we undertook a case study to answer these questions:

Q1: Can more dependencies be found by using both BPM and text-based requirements than using text-based requirements alone?

Q2: Are additional dependencies found in Question 1 actually useful in change impact analysis and how?

The kinds of data linked to research questions and collected through a questionnaire are shown in Table 3.

Table 3. Research question and data to collect

Question	Data to collect
Q1	Number of dependencies found through BPMs
	Number of dependencies found through text-based requirements
	Time spent on dependency discovery
Q2	Impacted requirements found through dependencies in BPMs
	Impacted requirements found through dependencies in text-based requirements
	Dependency types used in change impact analysis

The case we selected is a property valuation system (PVS) developed by NICTA (National ICT Australia) for a company employing the LIXI (Lending Industry XML Initiative) standards for the format and exchange of lending-related data using XML. LIXI is an independent non-profit organization established to remove data exchange barriers within the Australian lending industry. Through the work of LIXI, member organizations - including major banks, mortgage originators and brokers, mortgage insurers, property valuers, settlement agents, trustees and information technology providers - offer services to customers more efficiently and at lower costs. PVS had gone through changes with V2 being the latest. It included a "Pocket Valuer" sub-system on Personal Digital Assistant (PDA) for capturing property valuation data onsite, a desktop sub-system for managing information and a web-based business process system for managing the valuation workflows. The company planned to migrate all the desktop sub-system functions to a web-based system and implement new functions for PVS. In this paper, this whole new system is called PVS V3. This study was conducted as part of the PVS V3 requirements development and system design project. Our study used requirements in PVS V2 for dependency identification and change propagation analysis triggered by new requirements in PVS V3.

4.2 Case Study Procedures

Four practitioners, all experienced in requirements engineering, participated in this case study. They were evenly split into two groups: the Text group and the BPMN+Text group. The former used only text-based requirements specifications and the latter used both a BPMN process model and a text-based requirements specification. The BPMN+Text group were also familiar with the BPMN language. The case study was conducted as follows. Participants in the Text group were firstly given:

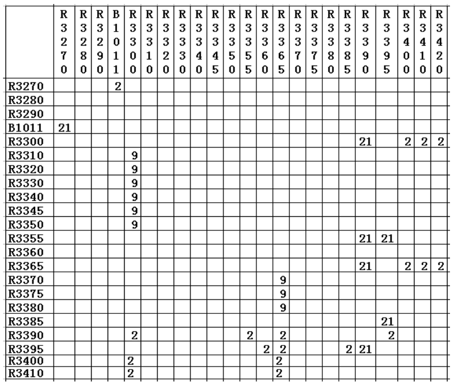
- thirty three change requests from PVS V3 (e.g. new functions);
- the dependency types and their definitions; and
- the PVS V2 requirements document written in natural language, consisting of 144 requirements organized into nineteen modules. Snippets of example modules "Valuation Requests" and "Valuation Bookings" are shown in Table 4:

Table 4. An example of text-based requirements for desktop application specification

Module	ID	Specification
1 - Valuation Requests	R3000	The ability to create valuation requests
	…..	…..
2 - Valuation Bookings	R3200	Requests should be able to be assigned to a valuer and booked for a specific date and time
	…..	…..

Participants in the Text group individually learned the dependency types, identified dependencies from the text-based requirements of PVS V2 and recorded them in a dependency matrix (example snippet in Fig. 3). Next, they analyzed the requirements estimated to be changed because of the change requests from PVS V3. Fig. 5(a) summarizes this procedure.

For the BPMN+Text group, the participants were handed the documentation to the Text Group plus the BPMN models to accompany PVS V2 and V3 requirements, a set of workflow patterns and the mapping between these patterns. After learning dependency types, the workflow patterns and the mapping, this group identified dependencies in the PVS V2 text-based requirements and the BPMN model, during which they identified the workflow patterns in the BPMN model and noted the dependencies in patterns in the BPMN model. Dependencies were recorded in a dependency matrix (e.g. Fig. 3) and annotated on the BPMN model (e.g. Fig. 4). Finally, they were asked to identify change propagation paths and dependency types used in change impact analysis and to analyze the requirements estimated to be changed as triggered by change requests in PVS V3. Fig. 5(b) depicts this procedure.



	R3270	R3280	R3290	B1011	R3300	R3310	R3320	R3330	R3340	R3345	R3350	R3355	R3360	R3365	R3370	R3375	R3380	R3385	R3390	R3395	R3400	R3410	R3420
R3270				2																			
R3280																							
R3290																							
B1011	21																						
R3300																			21		2	2	2
R3310					9																		
R3320					9																		
R3330					9																		
R3340					9																		
R3345					9																		
R3350					9																		
R3355																			21	21			
R3360																							
R3365																			21		2	2	2
R3370														9									
R3375														9									
R3380														9									
R3385																				21			
R3390					2							2		2						2			
R3395													2	2				2	21				
R3400					2									2									
R3410					2									2									

Fig. 3. An example of the dependency matrix

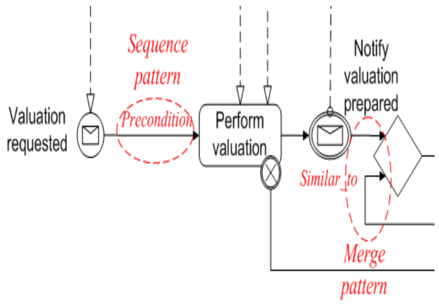


Fig. 4. An example of patterns and dependencies in the BPMN model

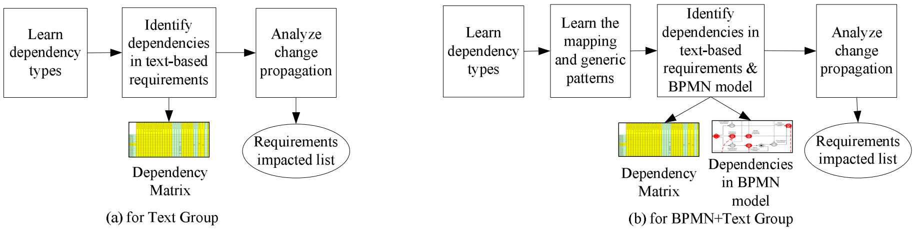

Learn dependency types
Identify dependencies in text-based requirements
Analyze change propagation
Dependency Matrix
Requirements impacted list
(a) for Text Group
Learn dependency types
Learn the mapping and generic patterns
Identify dependencies in text-based requirements & BPMN model
Analyze change propagation
Dependency Matrix
Dependencies in BPMN model
Requirements impacted list
(b) for BPMN+Text Group

Fig. 5. Case study procedures

4.3 Analysis Results

Q1. Can more dependencies be found by using both BPM and text-based requirements than using text-based requirements alone?

The BPMN+Text group found more dependencies with higher efficiency (i.e. higher number of dependencies found per hour) than the Text group (cf. Table 5). None found more than ten wrong dependencies because of their familiarity with PVS.

Table 5. Comparison of dependencies found

Category	Text Group	BPMN+Text Group
Number of implementation-related dependencies	125	231
Number of document-related and value-related dependencies	95	103
Total number of dependencies	220	334
Efficiency (number of dependencies identified per hour)	55	74.2

The numbers of document-related dependency found by two groups are similar, but there is 85% difference between the numbers of implementation-related dependencies found by both groups. To explain this difference, we analyzed the data, interviewed the participants and confirmed that the BPMN model and the mapping helped the BPMN+Text group to understand the text-based requirements and find more dependencies. This is because some dependencies are missing or not explicitly expressed in text-based requirements. This group also found twenty three requirements from the BPMN model that should be but were left out in the text-based requirements. This could be explained by the ambiguity of text-based requirements and a clearer relational view of business related requirements as provided by the BPMN model.

The implementation-related dependency distribution is shown in Fig. 6. The BPMN+Text group identified more instances of Precondition and Constraints dependencies than the Text group. There were many sequence patterns in the BPMN model corresponding to the Precondition dependency which were not explicitly represented in text-based requirements. Therefore, it was easier for the BPMN+Text group to discover Precondition dependencies. Additionally, there were many cancelling-related workflow patterns in the BPMN model which indicated the Constraints dependency. However, most of the canceling-related information was missing in text-based requirements as only normal event flows were described. The BPMN model, on the other hand, captured these exceptional events comprehensively and helped the BPMN+Text group to find more Constraints dependencies.

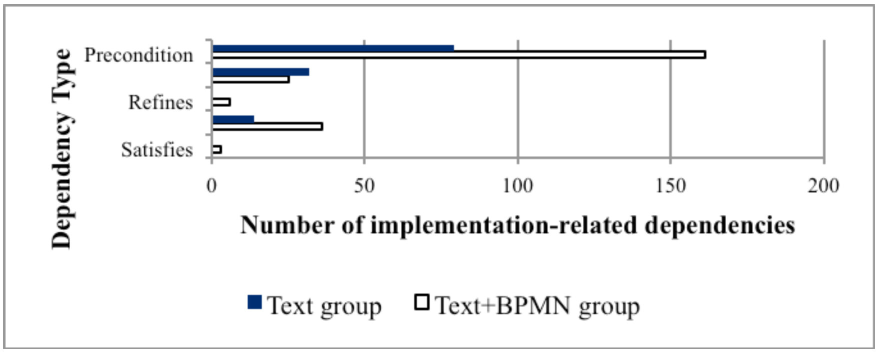


Fig. 6. Implementation-related dependency distribution

Q2. Are additional dependencies found in Question Q1 actually useful in change impact analysis and how?

We grouped thirty three change requests from PVS V3 into nine categories according to their intent and descriptions and compared the number of requirements impacted by the nine categories as reported by each of the participant groups. The number of impacted requirements for two request categories was the same for both groups and they were discarded from further analysis. The numbers of impacted requirements found for the remaining seven categories are shown in Fig. 7, The BPMN+Text group scored higher in each of the seven categories than the Text group since they found more dependencies and potentially more change impacts.

Although both groups identified all five implementation-related dependency types, the BPMN+Text group discovered 106 more implementation-related dependencies. Subsequently, this group used these extra dependencies to find forty seven impacted requirements, a significant portion of all the impacted requirements found (47%), suggesting that these dependencies were useful for change impact analysis.

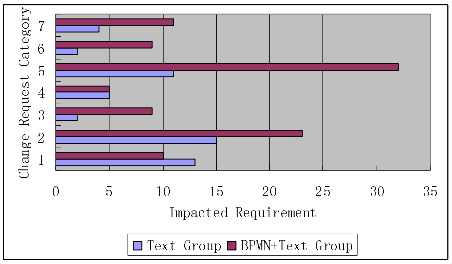


Fig. 7. Number of requirements impacted by change requests

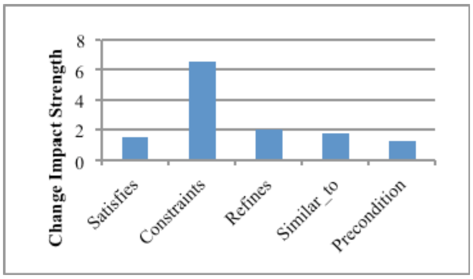


Fig. 8. Change impact strengths of dependency types

To show how the five dependency types are useful to change impact analysis, we used the concept of change impact strength – which we define as the average number of requirements impacted by one change request - to measure the capability of a dependency type to propagate changes between requirements. Fig. 8 shows the change impact strengths of the dependency types. “Constraints” has the highest change impact strength because this dependency type indicates that one requirement may interact with many other requirements to an extent like a crosscutting relation in aspect-oriented software development. Please note, from Fig. 8, that the change impact strengths of dependency types are different, and therefore it is important to pay attention to dependency types, which tend to have higher change impact strengths. Our findings provide a sweet spot for practitioners to analyze change impacts.

5 Discussions

Our case study emphasized realism and integrity of commercial confidentiality rather than the sample size. All our chosen participants were and/or had been involved in the development of the application at some stage of its lifecycle. However, this recruitment opened potential threats to the credibility of the case study. Thus, we investigated whether or not any relevant factors could lead to the bias of its results. In terms of the extent of domain knowledge about the application, one particular participant had been involved in the development of the project since inception. However, the other participant in the same group who had less experience with the application found more dependencies. The lack of domain knowledge did not seem to influence the results. We also observed that having specific knowledge and experience in requirements engineering or development determined greatly on what dependency types participants could find. One participant in the Text group, who has more requirements analysis experience, found more document-related dependencies while the other, who has more development experience, found more implementation-related dependencies. Both found similar numbers of dependencies. That means experience could influence the dependency types the participants focus on. On the other hand, we do believe that experience did influence a participant in learning the dependency types and later on the number of dependencies he/she found but the case study lacks statistical evidence to support it (i.e. only four participants).

There was no evidence of learning effect in this case study because both groups discovered dependency in one round. The Text group found dependencies in text-based requirements and the BPMN+Text group identified dependencies using both text-based requirements and the BPMN model together at the same time.

For external validity, we have only evaluated our approach on one project and acknowledge it as a limitation. Thus our evaluation results were constrained by the project. However, PVS appeared to be fairly complex and representative of a real application. The project offered us an opportunity to conduct an in-depth case analysis as an initial evaluation for our approach.

With regard to reliability, the participants reported that certain definitions of dependency types were vague when they applied the types. This might result in different understanding of the dependency types and affect the dependencies found. To assure the correctness and consistency of dependencies found, we conducted a follow-up interview with participants, discussed about disputed dependency types and achieved an agreement on the understanding of those disputed dependency types.

In related work, System Modeling Language (SysML) defines a number of requirement relationships for systems-of-systems and enterprise modeling [22]: derive, copy, verify, refine, satisfy and trace. The "trace" relationship is an abstract class for all the other relationships. "Refine" and "satisfy" are equivalent to our "Refines" and "Satisfies" relationships. The "verify" relationship links a test case to a requirement and it is not applicable to our dependency model which it is limited to the requirement stage and BPMN. Being software-centric, our current dependency model does not consider or explore SysML's "copy" and "derive" relationships. The former refers to one requirement's text property being a read-only copy of another requirement's for requirement reuse. The "derive" relationship relates one requirement to its derived requirements, which are at the next level of the system hierarchy. A future extension to our model and mapping is to incorporate all of SysML's requirement relationships to support systems-of-systems and enterprise architecture.

6 Conclusion and Future Work

From the requirements perspective, change impact analysis can be efficiently supported by dependency information. To facilitate dependency discovery in business process oriented applications, we proposed a mapping between a set of frequently occurring workflow patterns typical in BPMN, and a set of dependency types: Similar_to, Constraints, Precondition, Satisfies and Refines. Through this mapping, instances of these dependency types can be systematically derived from a typical BPM by identifying those workflow patterns from the BPMN-based model.

We conducted a case study in a real-world project to compare BPMN-and-text with text-only dependency discovery. In the former, case study participants discovered new dependencies in the requirements that were highly difficult to spot from the text-based requirements, suggesting an increased scope of change impacts. Through this case study, we also found the dependency types discovered through the mapping and BPMs very useful for change impact analysis and the change impact strengths of dependency types were different. These findings suggest that it is important to consider the nature of dependency during change impact analysis and inspire us in future work to explore the change impact abilities of different dependency types which may help improve the accuracy of change impact analysis.

Our study provides insights into applicability of business process modeling in requirement dependency discovery and change impact analysis for both research and practice. In the future, we will apply our approach to more industrial projects to validate its applicability.

Acknowledgement. This work is supported by National ICT Australia and funded through the Australian Government's Backing Australia's Ability initiative, in part through the Australian Research Council. This work is also supported by the Commonwealth of Australia under the Australia-China Science and Research Fund, the National Natural Science Foundation of China under grant No. 60803023, No. 61073044, National Science and Technology Major Project 2010ZX01036-001-002, 2010ZX01037-001-002.

References

1. Lehman, M.M., Ramil, J.F., Wernick, P.D., Perry, D.E., Turski, W.M.: Metrics and laws of software evolution-The nineties view. In: Proc. 4th Intl. Software Metrics Symp., pp. 20–32. IEEE Computer Society Press (1997)
2. Arnold, R.S., Bohner, S.A.: Software Change Impact Analysis. Wiley-IEEE Computer Society (1996)
3. Bohner, S.A.: Software change impacts-an evolving perspective. In: Int. Conf. Softw. Maintenance (ICSM 2002), pp. 263–272. IEEE Computer Society Press (2002)
4. Fasolino, A.R., Visaggio, G.: Improving software comprehension through an automated dependency tracer. In: Proc. 7th Int. Wkshp. Program Comprehension, pp. 58–65. IEEE Computer Society Press (1999)
5. De Lucia, A., Fasano, F., Oliveto, R.: Traceability management for impact analysis. In: Frontiers of Software Maintenance (FoSM 2008), pp. 21–30. IEEE Press (2008)
6. Aizenbud-Reshef, N., Nolan, B.T., Rubin, J., Shaham-Gafni, Y.: Model traceability. IBM Syst. J. 45(3), 515–526 (2006)
7. Dag, J., Regnell, B., Carlshamre, P., Andersson, M., Karlsson, J.: A feasibility study of automated natural language requirements analysis in market-driven development. Requirements Engineering 7, 20–33 (2002)
8. Pohl, K.: PRO-ART: Enabling requirements pretraceability. In: 2nd Int. Conf. Requirements Eng., pp. 76–84. IEEE Computer Society Press (1996)

9. Li, J., Zhu, L., Jeffery, R., Liu, Y., Zhang, H., Wang, Q., Li, M.: An initial evaluation of requirements dependency types in change propagation analysis. In: 16th Int. Conf. Evaluation & Assessment in Softw. Eng., EASE 2012 (2012)
10. Larkin, J.H., Simon, H.A.: Why a Diagram is (Sometimes) Worth Ten Thousand Words. Cognitive Science 11, 65–100 (1987)
11. Object Management Group: Business Process Modeling Notation Specification v1.2 (2009)
12. Hassine, J., Rilling, J., Hewitt, J., Dssouli, R.: Change impact analysis for requirement evolution using use case maps. In: 8th Int. Wkshp. Principles of Software Evolution, pp. 81–90. IEEE Computer Society Press (2005)
13. von Knethen, A., Grund, M.: QuaTrace: a tool environment for (semi-) automatic impact analysis based on traces. In: Int. Conf. Softw. Maintenance (ICSM 2003), pp. 246–255. IEEE Computer Society Press (2003)
14. Dahlstedt, Å., Persson, A.: Requirements Interdependencies: State of the Art and Future Challenges. In: Engineering and Managing Software Requirements, pp. 95–116. Springer, Berlin (2005)
15. Yan, Y.-Q., Li, S.-X., Liu, X.-M.: Quantitative Analysis for Requirements Evolution's Ripple-Effect. In: Int. Asia Conference on Informatics in Control, Automation and Robotics (CAR 2009), pp. 423–427. IEEE Computer Society Press (2009)
16. de la Vara González, J.L., Díaz, J.S.: Business process-driven requirements engineering: a goal-based approach. In: Pernici, B., Gulla, J.A. (eds.) Proc. CAiSE 2006 Workshops (vol. 1) - 8th Wkshp. Business Process Modeling, Development, and Support (BPMDS 2007), Trondheim, Norway (2007)
17. Cardoso, E.C.S., Almeida, J.P.A., Guizzardi, G.: Requirements engineering based on business process models: A case study. In: 13th Enterprise Distributed Object Computing Conference Workshops (EDOCW 2009), pp. 320–327. IEEE Computer Society Press (2009)
18. Mathisen, E., Ellingsen, K., Fallmyr, T.: Using business process modelling to reduce the effects of requirements changes in software projects. In: 2nd Int. Conf. Adaptive Science & Technology (ICAST 2009), pp. 14–19. IEEE Computer Society Press (2009)
19. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distributed and Parallel Databases 14, 5–51 (2003)
20. Wohed, P., Russell, N., ter Hofstede, A.H.M., Andersson, B., van der Aalst, W.M.P.: Patterns-based evaluation of open source BPM systems: The cases of jBPM, OpenWFE, and Enhydra Shark. Information and Software Technology 51, 1187–1216 (2009)
21. Kous, K.: Comparative analysis versions of BPMN and its support with Control-flow patterns. In: MIPRO, 2010 Proc. 33rd Int. Convention, pp. 315–319. IEEE Computer Society Press (2010)
22. Object Management Group: OMG Systems Modeling Language (OMG SysML^{TM}) v1.2 (2011)