Supporting Change Impact Analysis for Service Oriented Business Applications

Hua Xiao[1], Jin Guo[1] and Ying Zou[2]

School of Computing[1]
Queen's University
Kingston, Ontario, Canada
{huaxiao,guojin}@cs.queensu.ca

Department of Electrical and Computer Engineering[2]
Queen's University
Kingston, Ontario, Canada
ying.zou@queensu.ca

Abstract

Business applications encode various business processes within an organization. Business process specification languages such as BPEL (Business Process Execution Language) are commonly used to integrate various services in order to automate business processes within an organization. To remain competitive edge, managers frequently modify their processes. Determining the cost of modifying a business process is not trivial since the changes to the business process have to account for source code changes in various services. In this paper, we propose an approach to estimating the cost of a business process change in a service oriented business application. The approach applies change impact analysis techniques to business process specifications, and source code. The approach generates an initial change impact set from business process components. These components are then mapped to the corresponding source code entities. These code entities act as seeds for traditional source code impact analysis. Using code dependencies, such as call and inheritance relations, we derive a metric to capture the complexity of particular business process changes. Managers can then use this metric to gauge the cost and resources needed to implement changes in their business processes without having to study the code. We demonstrated the feasibility of our approach using an experiment on an open source service oriented business application.

1. Introduction

Business applications assist users in performing their daily work more efficiently and effectively. Business applications automate business processes. Business processes are a set of logically related tasks that are performed to achieve business objectives. For example, an on-line money order transferring business process consists of a sequence of tasks, such as, checking account balance, specifying transfer amount, and approving the transfer. Business process specification languages, such as BPEL4WS (Business Process Execution Language for Web Services) can be used to integrate various services, such as credit card authorization, into a service oriented business application.

Due to marketing strategies, organizations tend to reengineer their business processes in order to improve performance indices, such as cost and quality of services. When a business process is changed (e.g., adding a task, and removing a task), the source code that implements the business application needs to be updated in order to support the new business requirements. However, determining the impact of a business process change is not trivial and is challenging since business analysts are not experts in the source code and the spread of the changes across services may be too complex.

Business process changes usually result in code changes. These code changes may cause unexpected side-effects to other business processes. It is important for business analysts to evaluate the impact of business process changes especially when considering various alternatives. For example, a simple change to a business process such as the addition of a welcome screen or an order summary may seem trivial but may require a substantial amount of code changes. The code changes may be too costly in particular when the business value of such a summary page is considered. We would like to develop an approach that can give business analysts a rough and quick estimate of the impact and cost of changing a business process within a service oriented environment.

Most software engineering research focuses on change impact analysis at specific abstraction levels,

such as requirements, design or source code. For instance the impact of a requirement change is determined using other related requirements without mapping the requirement changes all the way to the code. In a business process modeling domain, an analyst can determine the impact of business process changes on other business processes. But the analyst cannot easily determine the overall cost at the code level without consulting with the development team.

In this paper, we propose an approach to support impact analysis and cost estimation activities performed by business analysts. The approach integrates results from two levels of abstraction: change impact analysis at the business process level and change impact analysis at the source code level. For a proposed change in a business process, the approach:

1) Identifies the business process components which must be changed and stores them in a *business component impact set*;
2) Performs impact analysis using data and control dependencies from the business process specification, in order to expand the *business component impact set*. The expanded impact set contains all business process components which may require modification;
3) Uses established links between business process components and source code entities [1] in order to map all the elements in *business component impact set* to their corresponding source code entities;
4) Stores the corresponding code entities in a *source code impact set*;
5) Analyzes the data and control dependencies of the source code entities in the *source code impact set* and generates propagation graphs for each source code entity in the impact set; and
6) Calculates a *change impact metric*. The metric is based on the generated propagation graphs. The metric attempts to capture the overall impact caused by a change in a business process.

The rest of the paper is organized as follows: Section 2 discusses steps (1-3) in our approach. In particular it introduces business processes and discusses the various types of changes that affect the functionality of business processes. Section 2 as well discusses how business process components are mapped to source code entities. Section 3 presents the other steps in our approach. It describes the analysis of the change impact in source code level and the calculation of the change impact metric. Section 4 demonstrates a case study. Section 5 discusses the related work. Finally, Section 6 concludes the paper and presents the future work.

2. Impact Analysis at Business Process Level

A workflow provides a computerized specification for a business process. More specifically, a workflow consists of five components:

- Tasks describe the steps needed to achieve business objectives. A task can be described by a set of internal properties (e.g., time to execute, automatic task, and resource requirements) and external properties (e.g., input data and output data). Tasks are usually implemented by services in a SOA environment;
- Control flows determine the execution path of tasks. More specifically, a set of tasks that can be executed in different orders, such as sequence, parallel, alternation and repetition;
- Data flows describe the input/output of a task;
- Roles perform tasks; and
- Resources are necessary conditions for executing tasks.

Since a business analyst is familiar with workflows and workflow components, then the analyst is asked to specify business process changes at the business process level. We have identified several primitive changes in business processes. Table 1 lists these primitive changes. If the internal properties of a task, such as execution time, are modified, then this change is limited to that particular task and the external properties of the task are not affected. Therefore the *business component impact set* of such a change contains only the modified task. If the input data of a task is modified, then the change impacts the modified task and propagates to the prior tasks. The propagation of the change indicates that output data of prior tasks may need to be modified in order to match the modified input data of the changed task. The *business component impact set* includes the modified task and the prior tasks. When the output data of a task is modified, then the change impacts the modified task and all following tasks. In this case, the output data of the task fails to match the input of its following tasks. In the case that a new task is added and the external properties (e.g., input and output) of the new task are compatible with its neighbors, such a change will likely not affect other tasks. When a new task mismatches the external properties of its neighbors, such a change impacts the neighbors of the task. Hence the impact set will contain its neighbors. Task deletion has the similar situation as task addition. Other

complex changes can be composed by multiple primitive changes.

Table 1. Summary of primitive changes in workflow

Primitive change	Description	Business component impact set
Inner property modification	Modify the implementation of a task, without changing the interfaces	The modified task
Input data modification	Modify the input interface of a task	The modified task and its prior tasks
Output data modification	Modify the output interface of a task	The modified task and its following tasks
Task addition with matched interfaces	Add a new task. The input/output interfaces match the interfaces of the neighbors.	The added task
Task addition with mismatched interfaces	Add a new task. The input/output interfaces of the task mismatch the interfaces of its neighbors.	The added task and the tasks with mismatched interface
Task deletion with matched interfaces	Delete an existing task. The input/output interfaces of the task match the interfaces of the neighbors.	None
Task deletion with mismatched interfaces	Delete an existing task. The input/output interfaces of its neighbors are not matched after the deletion	The neighbors of the deleted task with mismatched interfaces

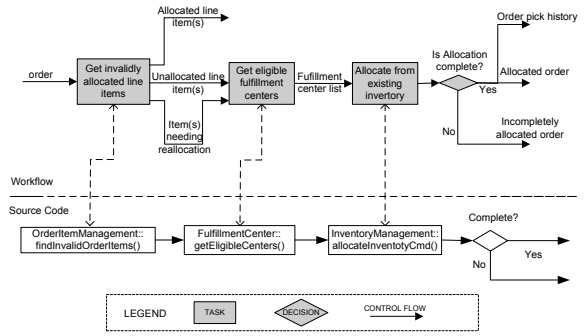

Figure 1. Mapping between workflow and source code for allocate inventory

Once the *business component impact set* is identified and reviewed by the business analyst. We must then map the changes to the code entities. In prior work we developed a technique to automatically map workflow components (e.g. tasks) to source code entities [1]. Figure 1 maps a workflow specification (at the top) to methods (at the bottom) which implement the corresponding tasks. The mapping is indicated using dashed arrows.

To demonstrate our approach that can propagate the business process changes to source code level, we present a simple example. Consider the task "*Get eligible fulfillment centers"*, shown in Figure 1. The task is performed by a customer. We would like to modify it so it can be performed by a sale representative. Such a change is an inner property change which is limited to the task and would not propagate to other tasks. The *business component impact set* would contain the "*Get eligible fulfillment centers*" task. From the mappings between a workflow to the code, we can identify a set of methods that is associated with elements in the business component impact set. In this case, the method, *FulfillmentCenter::getEligibleCenters()*, should be modified. Hence, this method is stored in the *source code impact set*.

Alternatively if the input data of the task "*Get eligible fulfillment centers"* is modified, such a change may have influence on the prior task (i.e., "*get invalidly allocated line items*"). Therefore both tasks will be added to the *business component impact set*. Following the mapping relations between both tasks and source code, the methods *FulfillmentCenter::getEligibleCenters()* and *OrderItemManagement::findInvalidOrderItems()* are stored in the *source code impact set*.

3. Impact Analysis at Source Code Level

To analyze the change impact for a given method in the *source code impact set*, we construct a propagation graph for each entity (e.g., method) in the *source code impact set*. We construct a propagation graph by incorporating all method invocations and (directly and indirectly) inheritance relations for an entity in the *source code impact set*. We then derive a mathematical formula to give a quantitative estimation of the effort needed in order to implement a particular business process change. In the following subsections, we elaborate on impact analysis at the source code level.

3.1 Analysis of Class Hierarchy

A class hierarchy represents the inheritance relationship among classes and their methods. When a method in a class is changed, the impact of changes can be propagated to methods defined in its children classes through the inheritance relation. To create a propagation graph for a particular method in the *source code impact set*, we traverse the inheritance relations among class declarations, and record the relations between methods defined in a class and its direct

children and indirect children (e.g., grandchildren). For example as illustrated in Figure 2, class *AirExpressCenters* and class *GroundShippingCenters* are inherited from class *FullfillmentCenters*. Method *getEligibleCenters()* is declared in class *FullfillmentCenters*. The method *getEligibleCenters()* in class *AirExpressCenters* overwrites the method *getEligibleCenters()* defined in class *FullfillmentCenters*. The changes in method *getEligibleCenters()* in class *FulfillmentCenters* can affect the methods defined in *GroundShippingCenters()* for the reason that the method *getElgibleCenters()* in *FulfillmentCenters* is reused in class *GroundShippingCenters*. However, the changes have no impact on the method *getEligibleCenters()* in class *AirExperessCenters*.

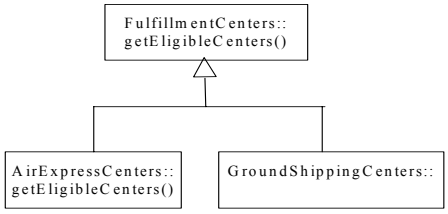

Figure 2. An Example Class Hierarchy

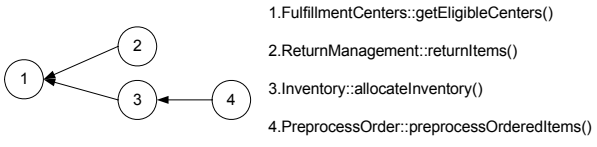

Figure 3. An Example Call Graph

3.2 Analysis of Call Graph

A call graph is a directed graph which illustrates the calling relationship among methods. In a call graph, a node represent an individual method, and edges represent call sites. Figure 3 depicts an example of a call graph. When a method is changed, methods which directly or indirectly call the changed method may be impacted. For instance as depicted in Figure 3, method *FulfillmentCenters::getEligibleCenters()* is invoked by method *Inventory::allocateInventory()*, which, in turn, is called by *PrepprocessOrder::preprocessOrderItems()*. When method *FulfillmentCenters::getEligibleCenters()* is changed, the method *Inventory::allocateInventory()* may be directly affected by this change. Moreover, *PrepprocessOrder::preprocessOrderItems()* may be indirectly affected by this change.

3.3 Creation of Propagation Graph

To estimate the cost of performing a source code change, we construct a propagation graph for each method that needs to be changed. We illustrate the construction of the propagation graph using an example, as shown in Figure 4. We begin from a changed method (node 1 - method *FullfillmentCenters::getEligibleCenters()*). Node 1 forms the initial node in the propagation graph. We expand the propagation graph by adding node 6 representing method *GroundShippingCenters::getEligibleCenters()* which directly inherits from *FulfillmentCenters::getEligibleCenters()*. From the Call Graph, shown in Figure 3, method 2 and 3 directly call method 1 and method 4 calls method 3. The expansion of the propagation graph stops when no additional methods can be added. We do not add library methods.

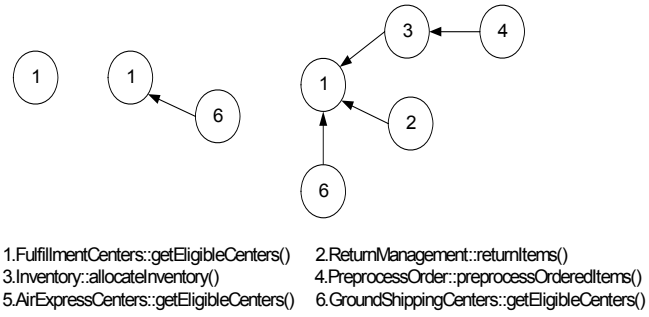

Figure 4. Different steps in constructing the impact propagation graph (from left to right)

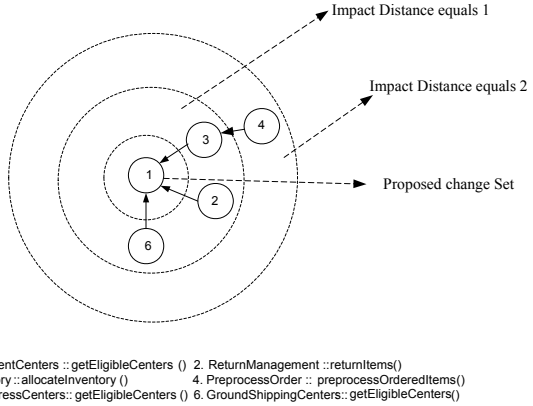

Figure 5. Calculating Impact Distances

To evaluate the impact of changes in source code, we calculate the distance between a changed method and each affected method in its propagation graph. For example as illustrated in Figure 5, the node 1 in the center represents a method to be modified. The affected methods (e.g., nodes 2, 3, and 6) are one distance away from node 1. The nodes with distance 1 are likely to be directly impacted by changes in node 1. Node 4 is two distance away from node 1. Node 4 is less likely to be affected by the changes to node 1.

$$I(M) = \sum_{Mj \in S} \frac{1}{D(M_j)} \quad (1)$$

Set $S = \{M_j | M_j$ is the methods in the propagation graph of method $M\}$. $D(M_j)$ measures the shortest distance from method M to method M_j in the propagation graph.

To quantify the overall impact caused by changing a method, we propose a preliminary formula to estimate the change effort. As shown in Formula 1, S contains a set of methods (e.g., $M_j \in S$) that are affected by a method to be changed (i.e., M). The distance between method M and method M_j is represented by $D(M_j)$. Essentially, when a method, M_j, is further away from the method to be changed, the method, M_j, is less likely to be affected by the changed method, M. Therefore, the likelihood of being impacted is evaluated as $1/D(M_j)$. $I(M)$ denotes the overall impact caused by a changed method, M. As shown in Formula 1, $I(M)$ is a sum by adding all the likelihoods of being impacted for each node in the propagation graph. When a method is invoked following different invocation paths, the same method can appear in a propagation graph with different distances. In this case, we consider the minimal distance from the method to be changed (i.e., M) and the method ($M_j \in S$) with multiple occurrences, since the minimal distance shows the strongest impact.

As a change in a business process can result in a set of methods, which are mapped to the business components, to be changed, we calculate the impact value for each method in the *source code impact set* and take the summation of the impact value for each method. In the future we plan to refine Formula 1 to consider more complex situations. For example, the propagation graph for a method may enclose all the methods defined in a program. We plan to identify heuristic to filter the irrelevant methods in the propagation graph.

4. Experiment

To demonstrate the preliminary result of our approach, we apply our approach on an open-source system, namely Open For Business Project (OFBiz). The OFBiz project provides a service oriented framework which enables organizations to perform various functionalities for Enterprise Resource Planning (ERP), Customer Relationship Management (CRM) and E-Commerce [2]. In the OFBiz project, applications are implemented in Java and a proprietary scripting language, called Mini-Language, which can be used to integrate various service applications. In this case study, we select one of the variants, called Sequoia Open Source ERP [3], from the OFBiz project.

Table 2. Case studies for OFBiz

Features	Process 1	Process 2
Task #	3	4
Class #	4	3
Methods #	7	6
Impact Value	2	5/2
Time to Change (minutes)	32	41

We utilize two business processes recovered from the source code. As shown in Table 2, Process 1 handles the requests from users, and consists of three tasks, including parsing requests, logging the requests, and sending the response. The source code that implements these three tasks involves seven methods in four different classes. If we wanted to modify the task *parsing request*, a method, called *parseRequest()*, has the direct mapping to the task. Moreover, two other methods utilize *parseRequest()* and therefore may be impacted by the change of the method *parseRequest()*. The distance of both methods is 1. Therefore, the impact involved in modifying *parseRequest()* is 2, using Formula 1. We further measure the time takes a developer to modify the source code. The time to change row as shown in Table 2 includes the time for understanding the code and the time for identifying the location in the source code to implement the code. We exclude the time for compiling, debugging and testing.

The second business process creates a series of public or private keys. As shown in Table 2, this process consists of four tasks, including *reloading keys*, two logging tasks that handle logging using two different approaches) and *acknowledgement*. Six methods from three classes implement this business process. In this business process, we change the task, *reloading keys* in the cache. The impact of performing the changes in business process is 5/2.

As illustrated in Table 2, the impact value for Process 1 is less than the impact value of Process 2.

Moreover, it takes less time to modify the required changes in Process 1 than Process 2, by comparing results in "time to change" row in Table 2. The results of this experiment indicate that our framework could help to analyze the impact of a change at the business process level. In the future we plan to compare the value of our metric against the true cost of performing such changes.

5. Related Work

Many approaches and prototype tools are developed to conduct impact analysis in the level of source-code. [4] proposes an approach that compares the changes between two versions of a Java program, decomposes their differences into a set of atomic changes, and calculate inter-dependences of these changes. OMEGA [5] is an integrated environment tool for C++ program maintenance. OMEGA adapts program slicing to identify the propagation effects caused by code modification. McCabe [6] supports impact analysis using call graphs. Code Surfer [7] provides a browser that understands pointers, indirect function calls, and whole-program effects. [8] presents a technique for dynamic impact analysis. [9] proposes several heuristics to predict change propagation.

In the requirement and design level, [10] proposes an approach to support change impact analysis at the level of the requirements. The focal point of this approach is the use of probability to combining the predicted impact propagation paths. However, this approach focuses on the impacts caused by changes to requirements and can not reflect the impact among requirements and source code. [11] presents an approach to link the test cases with requirements, design and code. [12] proposes a UML model-based approach to performing impact analysis.

In our research, we trace the business requirement changes to the source code which implements these requirements. We aim to provide a business analyst with a qualitative assessment of the effort required to make changes in business requirements.

6. Conclusion and Future Work

This paper presents a change impact analysis approach that integrates the change impact analysis in business process and the change impact analysis in source code. We propose a formula to quantify the impact caused by changes made in business process level. Our work can assist the business analysts to estimate the cost for conducting the required changes in business requirements without having to study the code. We demonstrated the feasibility of our approach through an experiment on an open source business application. In the future, we plan to perform empirical case studies to evaluate our approach.

References

[1] Y. Zou, T. Lau, K. Kontogiannis, T. Tong and R. McKegney, "Model-Driven Business Process Recovery", In Proceedings of the 11th Working Conference on Reverse Engineering, 2004, pp. 224-233.

[2] Open for Business home, http://www.OFBiz.org

[3] Sequoia ERP. http://sourceforge.net/forum/forum.php?forum_id=517176

[4] X. Ren, G. Barbara, S. Maximilian and T. Frank, "Chianti: A Change Impact Analysis Tool for Java Programs", Proceedings of the 19th annual ACM SIGPLAN Conference on Object-oriented programming, systems, languages, andapplications, Vancouver, BC, Canada, October,2004

[5] X. Chen, W. Tsai, H. Huang, M. Poonawala, S. Rayadurgam and Y. Wang, "Omega-an integrated environment for C++ program maintenance", in Software Maintenance Proceedings, CA, Nov. 1996, Page(s):114 – 123

[6] http://www.mccabe.com

[7] http://www.grammatech.com/products/codesurfer/index.html

[8] J. Law, and G. Rothermel, "Whole program Path-Based dynamic impact analysis", Proceedings of the 25th International Conference on Software Engineering, Portland, Oregon, May 03-10, 2003

[9] A. Hassan, RC. Holt, "Predicting change propagation in software systems", Proceedings of the 20th IEEE international conference on software maintenance, Chicago, USA, pp 284–293, 2004

[10] S. Ibrahim, N.B. Idris, M. Munro and A. Deraman, "A Requirements Traceability to Support Change Impact Analysis", Asian Journal of Information Technology, Pakistan, vol. 4(4), pp. 345-355, 2005

[11] S. Lock and G. Kotonya, "An Integrated Framework for Requirement Change Impact Analysis", proceedings of the 4th Australian conf. on Requirements Engineering, September 1999

[12] L. C. Briand, Y. Labiche, L. O'Sullivan, "Impact Analysis and Change Management of UML Models", Proceedings of the International Conference on Software Maintenance, IEEE September 2003, pages 256-265,2003