An Approach for Estimating the Time Needed to Perform Code Changes in Business Applications

Brian Chan, King Chun Foo

Department of Electrical and Computer Engineering

Queen's University, Kingston, Ontario, Canada

e-mail: {2byc, 3kcdf}@queensu.ca

Lionel Marks

School of Computing

Queen's University, Kingston, Ontario, Canada

e-mail: lmarks@cs.queensu.ca

Ying Zou*

Department of Electrical and Computer Engineering

Queen's University, Kingston, Ontario, Canada

e-mail: {ying.zou}@queensu.ca

*contact author

Abstract Business applications automate various business processes within an organization. Business analysts frequently modify business processes to maintain a competitive edge. Estimating the time needed to modify a business process is not a trivial task, since changes to the business process result in changes to the source code of which the business analyst has limited knowledge. We propose a change impact metric which estimates the code to be modified as a result of a business process change. We demonstrate the effectiveness of our proposed change impact metric through a case study, using seven large business applications from the OFBiz open source project.

Keywords: Impact Analysis, Code Changes, Business Application, Business Processes.

---

1 Introduction

Most organizations rely on business applications to automate their daily operations and processes, such as planning enterprise resources, managing customer relationships and conducting sales. A business process describes how a set of logically related tasks are executed, ordered and managed by following business rules to achieve business objectives. For instance, a bookstore's website is a business application that follows a business process and executes a set of tasks, such as searching for books, adding the selected books into a shopping cart if the books are in stock, and validating the buyer's credit card.

Organizations must adapt their business processes and evolve their supporting business applications to cope with rapid changes to business requirements. Business analysts constantly reengineer the business processes in order to improve organizational performance measures such as reducing cost, increasing revenue and enhancing the quality of services. When a business analyst changes a business process by adding a task, or removing a task, the source code which implements the business processes must be updated to reflect the new business requirements. These code changes may cause unexpected side-effects to other business processes. A simple change to a business process such as the addition of a welcome screen or an order summary page may seem trivial but may require a substantial amount of code changes. A code change might be too complex when considering its business value (e.g., the value of a summary page might be very low). It is important for business analysts to evaluate the impact of business process changes on the implementation, especially when considering various alternatives. The complexity of the changed code increases the maintenance costs. Rough estimates for the changed code range are also critical when customizing a

vendor-specific business application to suit the needs of an organization during a consulting engagement.

Impact analysis attempts to quantify the amount of time needed to implement a specific change. However, determining the impact of a business process change is not trivial since business analysts are not experts in understanding source code and the scattering of the changes across various classes and packages may be too complex. In this paper, we develop an approach that gives business analysts a rough and quick estimate on the time needed to propagate business process changes to the code. Our approach integrates change impact analysis from two levels of abstractions: the business process and the source code levels. To quantify the time needed to change the code impacted by a proposed business process change, we build a change propagation graph that describes data and call dependencies of potentially affected source code entities. The propagation graph is built by first locating the code entities directly related to the changed business process components, then propagating the change in the source code entities while filtering code entities based on their business relevance. To provide an accurate time estimate, we consider a decay model which reduces the probability of changing a code entity based on its distance in the propagation graph from the initial affected code entity. As an extension to our work in [39], this paper improves the case study by analyzing the predictive power of the proposed metric that can estimate the efforts for a developer to make changes in the code.

The rest of the paper is organized as follows. Section 2 presents a simple example to cover the needed background and motivate our work. Section 3 discusses the impact analysis at the business process level. Section 4 discusses the impact analysis at the source code level. We propose a change impact metric for estimating the needed time. Section 5 presents a case study to demonstrate the effectiveness of our proposed metric. Section 6 gives an overview of the related work. Finally, Section 7 concludes the paper and discusses future work.

2 A Motivating Example

Business analysts create, visualize, and analyze business processes using business process modeling tools, such as IBM WebSphere Business Modeler (WBM)[30]. Business process definitions are often described in proprietary formats. Business processes can also be specified using standard formats, such as XPDL (XML Process Definition Language) [32]. A business process consists of three major components:

- Tasks describe the steps needed to achieve a business objective. 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). For example, the *Purchase Order* process as shown in Fig. 1 handles the checkout process and validates the customer payment information. *Checkout Cart, Enter Shipping Address,* and *Enter Payment Information* tasks are performed by a customer by entering the relevant information into the website. Consequently, the payment information is automatically validated by a credit card processing system.
- Control flows determine the execution path of tasks. A set of tasks can be executed in different orders, such as sequence, parallel, decision and repetition. For example, the tasks depicted in Fig. 1 are executed in sequential order.
- Data flows describe the input/output of a task. For example as shown in Fig. 1, *Enter Shipping Address* task accepts *Shopping Cart* as input data and generates *Order* as output data.

We developed a business process explorer tool which automatically recovers business processes from business applications and establishes the links between the business process components (e.g., tasks, data and control flows) and the source code entities (e.g., methods, and variables)[16]. In a business application, a variable keeps the data fetched from the database. Such a variable is used as an input for a task. Moreover, the content of a variable is saved into the database as the output of a task. The input and output of a task in a business process are called business data. Such data conveys high-level business concepts, such as *Order* and *Shopping Cart,* and is stored in a database. Our tool automatically identifies business data from local variables by checking the direct or transitive data dependencies on the database accesses. The techniques for recovering business processes from business applications, identifying business data, and maintaining a mapping between (i.e., linking) business process components and source code entities are detailed in [16, 20, 33]. We have applied these techniques successfully on several large open source and commercial software applications. Fig. 1 illustrates the linked code entities corresponding to the tasks in the recovered business process.

A business analyst could examine the process diagram and analyze the impact of a change using a business process modeling tool. For example to provide customers with more secure on-line payment options, a business analyst may add *PayPal* as an additional payment method. This decision results in changing the *Validate Payment Info* task in the *Purchase Order* process. The corresponding method

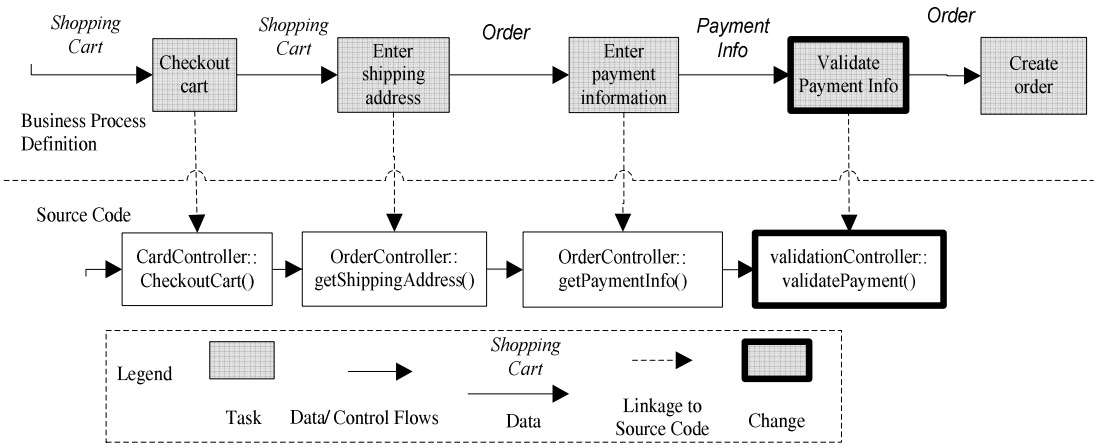

Fig. 1. A Purchase Order Business Process with a Change in the *Validation Payment Info* Task to Support PayPal.

Table 1. Summary of Primitive Changes in a Business Process

Case	Name	Primitive Business Process Change	Description	Business Component Impact Set
1	Inner Property Modification	Task 1 → Task 2 (chg) → Task 3 $Output(Task_1) = Input(Task_2)$ && $Output(Task_2) = Input(Task_3)$	Modify the implementation of a task, without changing the interfaces.	$S_{input}=\{\ \emptyset\ \}$ $S_{output}=\{\ \emptyset\ \}$ $S_{logical}=\{Task_2\}$
2	Input data modification	Task 1 ⇢ Task 2 (chg) → Task 3 $Output(Task_1) \neq Input(Task_2)$	Modify the input interface of a task	$S_{input}=\{Task_2\}$ $S_{output}=\{Task_1\}$ $S_{logical}=\{\emptyset\}$
3	Output data modification	Task 1 → Task 2 (add) ⇢ Task 3 $Output(Task_2) \neq Input(Task_3)$	Modify the output interface of a task.	$S_{input}=\{Task_3\}$ $S_{output}=\{Task_2\}$ $S_{logical}=\{\emptyset\}$
4	Task addition with matched interfaces	Task 1 → Task 2 (add) → Task 3 $Output(Task_1) = Input(Task_2)$ && $Output(Task_2) = Input(Task_3)$	The input/output of the added task match the interfaces of the neighbors.	$S_{input}=\{\ \emptyset\ \}$ $S_{output}=\{\ \emptyset\ \}$ $S_{logical}=\{\emptyset\}$
5	Task addition with mismatched interfaces	Task 1 ⇢ Task 2 (add) ⇢ Task 3 $Output(Task_1) \neq Input(Task_2)$ ‖ $Output(Task_2) \neq Input(Task_3)$	Add a new task. The input/output interfaces do not match the interfaces of the neighbors.	$S_{input}=\{Task_3\}$ $S_{output}=\{Task_1\}$ $S_{logical}=\{\emptyset\}$
6	Task deletion with mismatched interfaces	Task 1 → Task 2 (del) → Task 3 $Output(Task_1) \neq Input(Task_3)$	Delete an existing task. The input/output interfaces of its neighbors are not matched after the deletion.	$S_{input}=\{Task_3\}$ $S_{output}=\{Task_1\}$ $S_{logical}=\{\emptyset\}$
7	Task deletion with matched neighbors	Task 1 → Task 2 (del) → Task 3 $Output(Task_1) = Input(Task_3)$	Delete an existing task. The input/output interfaces of its neighbors are matched after the deletion.	$S_{input}=\{\ \emptyset\ \}$ $S_{output}=\{\ \emptyset\ \}$ $S_{logical}=\{\emptyset\}$
	Task — Data/ Control Flow — Changed Task — Changed Data/Control Flow — Deleted Task — Deleted Data/Control Flow	(chg) = Change (add) = Add (del) = Delete		

validationController::validatePayment(), must be changed to handle the *PayPal* payment. The changed task and its corresponding code are shown in bold in Fig. 1. Consequently, changes to *validationController::validatePayment()* may have an impact on the prior or subsequent methods in the code, which in turn affect the properties of the corresponding tasks in the business process. For example, the method *OrderController::getPaymentInfo()* that implements *Enter Payment Information* task may require restructuring to produce the input data required by the *validationController::validatePayment()*. However, changes to the return type of the method would cause changes in the output of the corresponding task (i.e., the *Enter Payment Information* task). This change would in-turn produce a ripple effect as it propagates to other methods, classes or

packages in the code. The change may affect other tasks in the same process or other processes in the application.

We propose a change impact metric which estimates the time needed to make this change. The metric quantifies the impact of a business process change at the source code level. Using this metric, a business analyst can perform a cost-benefit analysis with limited knowledge about the source code which implements the changed business processes.

3 Impact Analysis for Business Processes

Business analysts are familiar with business processes, so they specify changes to processes at the business process level. We identify seven primitive changes in a business process [31]. Complex changes can be treated as a combination of one or more primitive changes in a business process.

3.1 Primitive Changes in a Process

In a business process, a primitive change is limited to the granularity of a task. Addition, removal or modification of a task is examples of primitive changes in a process. For each primitive change listed in Table 1, we analyze the propagation of the change, and identify a set of changed components in a business process. More specifically, a primitive change of a task may lead to changes in the interfaces of its neighbors. For example, as shown in Case 2 of Table 1, the input data of a task (i.e., Task 2 in Case 2 of Table 1) is modified, then the change propagates to the prior tasks (i.e., Task 1 in Case 2). The propagation of the change indicates that the output data of prior tasks may need to be modified in order to match the modified input data of the changed task (i.e., Task 2 in Case 2). The changed components caused by a primitive change consist of either a change in the input of a task, the output of a task, or the internal logic of a task. For each primitive change, we collect the *business component impact set* by identifying three subsets: S_{input} , S_{output} and $S_{logical}$:

1. S_{input} and S_{output} refer to the sets of tasks that have an interface mismatch in their input or output respectively. For example, in Case 2 listed in Table 1, the *business component impact set* includes the modified task caused by the change of the input (i.e., S_{input}=$\{Task_2\}$) and the prior tasks caused by the change of the output (i.e., S_{output}=$\{Task_1\}$).
2. $S_{logical}$ refers to the set of tasks where the change modifies the internal logic of a task without affecting the input and output of the task. For example, in Case 1 listed in Table 1, the internal properties of a task, such as its execution time, are modified. This change is limited to that particular task. Therefore the *logical set* (i.e., $S_{logical}$ = $\{Task_2\}$) of such a change contains the changed task.

The *Business Component Impact Set* (BCIS) for each primitive change is listed in Table 1. Each entry represents a singular change to a business process. However there are cases where more than one primitive change is made. To find the impact of multiple changes, we must factor in all those primitive changes together. Multiple business process changes are defined as the union of individual primitive changes to the business process as shown in Equation (1). The impact domain of a compound change is composed of its primitive changes. Such changes are the combination of Case 1 (i.e., $S_{logical}$=$\{Task_2\}$) and Case 2 (i.e., S_{input}=$\{Task_2\}$ and S_{output}=$\{Task_1\}$). The BCIS for this case is $\{Task_2, Task_1\}$.

$$BCIS = \cup_i (S^i_{input} \cup S^i_{output} \cup S^i_{logical}) \quad (1)$$

BCIS refers to a business component impact set. i refers to cases listed in Table 1, i∈[1,7]. S^i_{input} represents a set of tasks with changes in the input in Case i. S^i_{output} represents a set of tasks with changes in the output in Case i. $S^i_{logical}$ represents a set of tasks which has logical changes in Case i. The symbol $\cup_i$ is the union over all the primitive changes of Case i.

It is important to identify the complete set of primitive changes that can be performed on a given task. Different primitive changes can affect similar code entities to produce synergistic effects. We can avoid counting the same affected code entity multiple times resulted from different primitive changes for a task in the impact analysis.

3.2 Tracing a Primitive Change to Code

Following the links between tasks and code entities, we collect methods which implement the tasks in the *business component impact set* and create a *source code impact set*. Most cases as listed in Table 1 involve changes in the input or output of a task in a business process. Furthermore, the inputs of a task are mapped to the incoming business data of the linked method. For example, the *Order getShippingAddress(ShoppingCart cart)* method is linked to the *Enter Shipping Address* task as shown in Fig. 1. The *Shopping Cart*, as an input to the task, is

linked to the *cart* variable which is defined as a parameter for the *getShippingAddress(ShoppingCart cart*) method. The output of a task is associated with the variables as outgoing business data from a method. For example, *Order* is the return type of the *Order getShippingAddress(ShoppingCart cart)* method. The return variable of the method is associated with the output of the *Enter Shipping Address* task shown in Fig. 1.

For Case 1 as listed in Table 1, the changes for the inner properties affect the business logic and rules in the code without modifying the signatures of the method. The business logic is composed of code blocks which operate on business data and deliver tasks specified in a process. A business rule makes decisions according to the state of the business data in the code. A business rule is often implemented using control statements such as if statements and for or while loops. Changes in the business logic and rules are governed by the business data passed as incoming or outgoing data of the method (e.g., using parameters or return value). For example, as shown in Fig. 1, the payment information is business data. The state of the payment information determines if the product should be delivered. Adding *Paypal* as a new payment method would change the implementation of the corresponding business rules and logic that operate on the payment business data. Although no interface change is needed in Case 1, it is important to trace the uses of the business data as incoming or outgoing data for the linked method. Therefore, the input and output of a task with internal logical changes are mapped to the parameters and return variables of the linked method in the code.

4 Impact Analysis at the Code Level

Every task in the BCIS is linked to a corresponding source code entity. For example shown in Fig. 1, the method *validationController::validatePayment()* has various input variables and is linked to the task *ValidatePaymentInfo*. We perform a static analysis on the Abstract Syntax Tree (AST) of the linked method and trace method invocations starting from the initial method linked to a changed task. To estimate the impact of a change on the source code, we propose a propagation graph that traces the dependencies of the business data as parameters or return variables derived from the initial changed method. We derive a mathematical formula to give a quantitative estimation of the time needed to fulfill a changed task in the *business impact component set*. In the following subsections, we elaborate on the impact analysis at the source code level.

4.1 Creating a Propagation Graph

When an initial method linked to a task in a business process is changed, methods which are directly or indirectly invoked by the initial method may be impacted. A call graph captures the control flow relation among methods. In a call graph, a node represents an individual method, and edges represent call sites. To estimate the impact of performing changes in the source code, our early work [31] expands the call graph by including all the method invocations initiated from the method linked to a changed task. Fig. 3 depicts an example of a call graph of the program listed in Fig. 2. In the example call graph, *m1()* calls *m2()*, and *m3()* and *m2()* calls *m4()*.

In a complex situation, a call graph for a method may enclose all the methods defined in a program. However, not all the methods in the call graph contain business-relevant code. In particular, the call graph includes utility methods that are used as building blocks to implement business logic. When a task is changed, the likelihood for modifying utility methods which do not directly deliver business tasks is low. To provide a more accurate estimate of the needed time, we filter the methods that are not relevant to high-level business tasks. We include the methods that use business data and their derived variables. The heuristics and rules used to determine the business relevance of a code and data entities are detailed in our prior work [16, 20, 33].The main intuition behind our prior work is the tracing of the flow of business data from user inputs and databases. Fig. 4 illustrates a propagation graph for the program shown earlier in Fig. 2.

The propagation of the incoming and outgoing business data is presented as dependency paths in the built propagation graph. We trace the uses of the incoming business data passed from the initial method, and check if the local variables are dependent on the business data in each method along the call paths. We identify the business relevant local variables by checking the assignment statements, which assign the information directly or indirectly obtained from the business data to a local variable. For example, as shown in Fig. 2, *m1()* is the initial method mapped to a business task in a business process. The parameter *v0* is linked to the incoming business data as the input of the changed task. We trace the uses of *v0* within the method body of *m1()* to derive local variables dependent on *v0*. In this example, the local variables, *v1*, *v2*, and *v3*, are derived as business data due to their dependence on *v0*. Dependencies are established using assignment statements in lines 3 to 5.

Similar to tracing the propagation of the incoming business data, the propagation of the outgoing business data is traced from the definition of the outgoing business data in the method. We locate the definition of an outgoing business data in a method and analyze the uses of the outgoing business data in defining other local variables. For example in Fig. 2, *v5* defined in line 10 and we assume that *v5* is a return variable (i.e., an outgoing business data). The uses of *v5* include lines 11 and 12 in Fig. 2.

The business data can be passed through parameters in method invocations. We expand a dependency path when business data is passed as parameters to an invoked method. We examine if the local variables defined in an invoked method are business relevant using assignment statements. For example as shown in Fig. 4, *v4* is business data since *v2* (i.e., business data) is passed to *v4*. Similarly, *v5* is derived as business data due to the data dependence on *v4* (i.e., business data). The expansion of a propagation graph stops under the following conditions:

1) an invoked method is linked to another task in the *business component impact set*;
2) the method calls itself; and
3) the end of the program.

In the propagation graph, we omit local variables that are not business relevant, since such local variables are implementation specific and are not likely impacted by changes to a task.

As shown in Fig. 4, the propagation graph contains the business relevant methods associated by the parameter passing in call paths, and the dependency paths of business data. The incoming business data and the definition of the outgoing business data are the sources of dependency paths. For example, *v0* is the source of three dependency paths: (*v0*, *v1, v6, v7*), (*v0, v1, v2, v3*), and (*v0, v1, v2, v4, v5, …*). Whenever a change in the source occurs, it will affect other variables on the dependency paths. If more than one dependency path is possible to a variable, the longest path will be considered. For example, two paths: (*v0, v1, v2, v3*) and (*v0, v3*) both leave from *v0* and lead to *v3*. The variable, *v3*, can be affected through two possible paths. We consider the maximum impact caused by *v0*. Therefore, we take the maximum distance between *v0* and *v3* (i.e., 3).

```
1.    m1(Type0 v0)
2.    {
3.        Type1 v1= v0.getValue();
4.        Type2 v2 = v1.getValue() + 1;
5.        Type3 v3 = m2(v2);
6.        m3(v1);
7.    }
8.    Type3  m2(Type2 v4)
9.    {
10.       Type3 v5= v4.getValue();
11.       m4(v5);
12.       return v5;
13.   }
14.   void m3(Type1 v6)
15.   {
16.       Type4 v7= v6.getX();
17.   }
```

Fig. 2. Example Program.

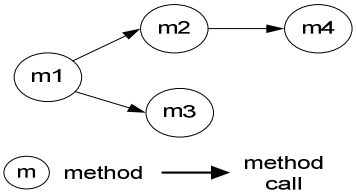

Fig. 3. An Example Call Graph.

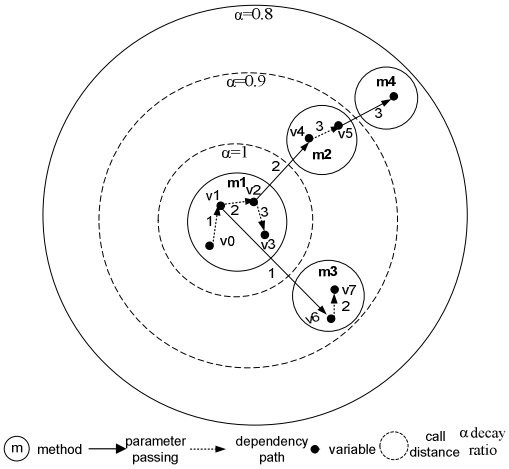

Fig. 4. An Example Propagation Graph.

4.2 Estimating Change Impact Value

The method in the center of a propagation graph represents the initial method that is linked to a changed task. As an invoked method is further away from the method in the center, the likelihood of such a method to be impacted is minimal. Similarly, when business data (i.e., variable) are further away from the source (i.e., the initial changed business data) along a dependency path, the potential for changing the code that uses the variable is decreased. Moreover, the complexity of a method also determines the effort

needed to understand and modify its code. To estimate the impact for changing business logic and rules, we measure the complexity of the control statements that involve business data.

$$ImpactValue = \sum_{business\ data} DependencyDistance * (Complexity + 1) \quad (2)$$

ImpactValue is an indicator for the change impact in the code. DependencyDistance refers to the distance of business data away from the source node of a dependency path. Complexity estimates the difficulty in changing business-relevant control statements.

We propose Equation (2) to calculate the impact value which estimates the change impact and indirectly measures the needed time. Using the business data path identified in a propagation graph, we quantify the impact by considering the dependency distance of each business data in a dependency path and the complexity of control statements involving each business data in the dependency paths. When business data are not used in control statements, the complexity value assigned for changing business logic and rules is set to zero.

Equation (2) is designed to reflect the synergy between *DependencyDistance* and *Complexity*. Business data that affect no conditions but appear in many lines of code have limited impact since the values of the business data have no impact on the conditional structures. Likewise, business data that appear rarely but are involved only in conditional structure can only affect the conditional expressions where they appear in. However business data that affect both the conditional structure and a large proportion of the code would have more significant effect. Specifically, the business data would not only affect the control flows of the code and the subsequent lines of code that contains the business data. The affected lines would potentially cascade into other conditional structures. Equation (2) reflects that no impact to the code will be large unless both the impacted lines of code (i.e., *DependencyDistance*) and the conditional structures (i.e., *Complexity*) are significant.

In the following subsections, we derive the measurement for dependency distance as defined in Equation (4). The complexity of the control statements involved with the business data is also derived later in Equation (5).

4.2.1 Measuring the Dependency Distance

To measure the propagation decay of a change to an initial method, we measure the call distance (i.e., *callDistance*) between the initial method and each invoked method in the propagation graph. For example, as illustrated in Fig. 4, the *m1()* in the center represents the initially changed method. For example, *m2()* and *m3()*) are one distance away from *m1()*. We introduce, α, as the propagation's decay ratio for a method in a call path in Equation (3). An example α with $k=10$ is illustrated in Fig. 4. The choice of the k value controls the amount of decay down a call path. The smaller the k value, the less impact a method with a large call distance produces.

$$\alpha = e^{-\frac{callDistance}{k}},\ k > 0,\ 0 < \alpha \le 1 \quad (3)$$

α is the decay ratio of an invoked method. callDistance is the number of method calls between the initial method and an invoked method. k is a positive real number that controls the impact of an invoked method in the call path.

Each method in a propagation graph has a collection of business data as described in various dependency paths (shown in Fig. 4). We count the dependency distance (i.e., *DependencyDistance* in Equation (1)) that each business data is away from the source of a dependency path. Business data with a greater dependency distance in a dependency path have less chance to be modified. The source business data mapped to the interface of a task (e.g., *v0* in Fig. 4) have a dependency distance of 0. For example as shown in Fig. 4, *v1* is defined data from *v0*. Then, *v1* is 1 distance away from *v0*. When a variable is passed to a parameter of a method, the dependency distance of the parameter of the invoked method retains the same dependency distance as the passing variable. For example as shown in Fig. 4, *v4* as a parameter of *m2()*, has a dependency distance of 3 same as *v3*, a variable in *m2()*.

Although business data in different methods may have the same dependency distance (not call distance (i.e., *callDistance*)), the business data in the central method (e.g., *m1()* in Fig. 4) are more likely to be changed than the business data in the methods further away from the center. For example, *v2* and *v7* have the distance of 2 in the dependency path. However, *v2* in *m1()* is more likely to be modified than *v7* in *m3()*. Therefore, we integrate the decay ratio of a method, which contains the business data in a dependency path, for computing the dependency distance of the business data. Moreover, the same business data can be used multiple times in a method. The overall dependency distance of one business data counts the frequency of uses within a method. As a

summary, the dependency distance of business data is computed using Equation (4).

$$DependencyDistance = varCount \times \alpha \quad (4)$$

varCount counts the number of instances for business data within a method in a propagation graph. α is the decay ratio of the method that contains the business data.

4.2.2 Measuring the Complexity of a Change

We propose Equation (5) to measure the complexity of a change associated with control statements, which examine business data collected in a propagation graph. Business data may be used in the conditional expression of different control statements in the code. To this effect, we compute the complexity of a single control statement, which examines the number of variables used in the conditional expressions, the nested condition depth, and the depth of the business data in the control statement.

$$Complexity = \sum_{statements} (NumVar + ConditionDepth - OccuringDepth) \quad (5)$$

NumVar refers to the total number of variables used in a control statement. ConditionDepth refers to the total number of nested conditions within the control statement. OccuringDepth is the shallowest depth of business data used in conditional expressions.

Traditional complexity metrics, such as the Cyclomatic complexity, evaluate complexity of a given code segment by counting the number of the control structures and execution paths. It does not take into the account of the business data used in the conditional expression. In particular, a conditional expression becomes complex when more business data are used to control the execution path. To measure the impact of complexity resulted from the changes in business tasks, a new metric is defined in Equation (5) to consider the effect of business data in the complexity of the code. As shown in Equation (5), *NumVar* counts the number of variables used in the expressions of a control statement. This includes variables that are not directly related to business data. For example, the *NumVar* for *if(a<b)* has is 2 regardless of how *a* and *b* are created as long as the control statement uses business data in a conditional expression. As stated in the McCabe complexity metric [25], the larger the number of nested conditions in a control statement, the more complicated of the control logic. *ConditionDepth* is the total number of nested conditional structures within a control statement. For example as shown in Fig. 6, the *ConditionDepth* is 4. *OccuringDepth* is the level at which the business data exist in the nested conditional structure. Business data at a shallower nested condition are more likely to change than more deeply nested conditions; therefore we consider the shallowest depth. In Fig. 5, the *OccuringDepth* of the business data corresponds to 3.

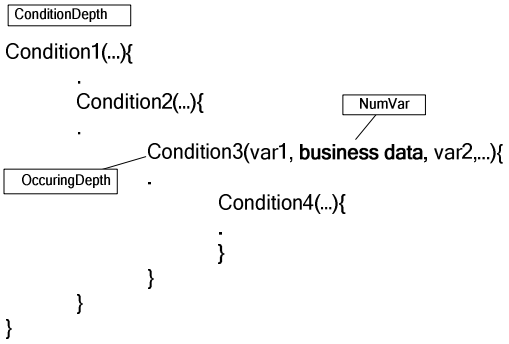

Fig. 5. Example Conditional Structure.

5 Case Study

We conducted a case study to examine the effectiveness of our proposed change impact metric in accurately estimating the time needed to change code based on a business level change. In this section, we describe the setup of the case study, and the criteria for evaluating the results. We also discuss the hypotheses, analyze results and elaborate on the threats to their validity.

5.1 Setup

The Apache Open For Business Project (OFBiz) [3] is a large open source project that offers a suite of business applications for ERP (Enterprise Resource Planning), CRM (Customer Relationship Management) and E-Commerce. The Business Process Explorer Tool [16] is used to recover the business processes implemented in OFBiz and establishes the links between the business process components and source code entities in the OFBiz project. In the recovered processes from the OFBiz project, the tasks are implemented using HTML and Java.

To assist a business analyst in reviewing and analyzing the business processes, we visualize the recovered processes using a commercial business process modeling tool, IBM WebSphere Business Modeler (WBM) [30]. Changes in the business processes are captured and traced into the linked source code entities.

To assess the impact of a proposed change in a process and in turn estimate the needed time to perform the change, we built a prototype impact calculator which implements the algorithm described in Section 4.

5.2 Criteria for Evaluating the Change Impact Metric

To measure the effectiveness of the proposed impact metric, we want to examine how well the change impact metric can be used to predict the amount of time a subject needs to make code changes as a result of particular change in a business process. We assume that the subjects take more time to complete their code analysis for changes that have a greater impact on the source code. The prediction of the change impact metric should relate to the amount of time a subject takes. For example if the impact metric returns a small value, it should take a subject a short time to make the changes in the code. For large impact values, a subject would spend more time.

Prior research shows that most complexity metrics correlate well with LOC [23]. Therefore, we use LOC as a base metric to compare our performance against. To measure the LOC, we sum the LOC of each method in a propagation graph. Intuitively the larger the affected LOC, the more time a subject will need to spend in understanding and modifying the code.

5.3 Data Collection

We recruited six subjects (graduate students) to implement specific business changes. The six subjects have a similar level of proficiency in Java, the implementation language of the OFBiz project, but they have different level of familiarity with the source code of the OFBiz project.

In our study, we focus on predicting the change impact on the Java source code. We randomly picked 102 business processes recovered from 7 applications in OFBiz project. The majority of business processes chosen relate to purchase applications similar to an online bookstore. Common processes include adding desired products to a cart or processing items through the check-out.

We randomly select 102 primitive changes, which covers approximately 14 to 15 changes for each type of primitive changes in business processes as listed in Table 1. We conduct one primitive change in each business processes. Each subject is randomly assigned 17 primitive changes. We measure the actual time for each subject to locate the impacted source code and perform, debug, and test the change. The summary of the data collection is listed in Table 2.

Table 2. Summary of Data Collection

# App.	# Business Processes	# Primitive Changes	# Subjects
7	102	102	6

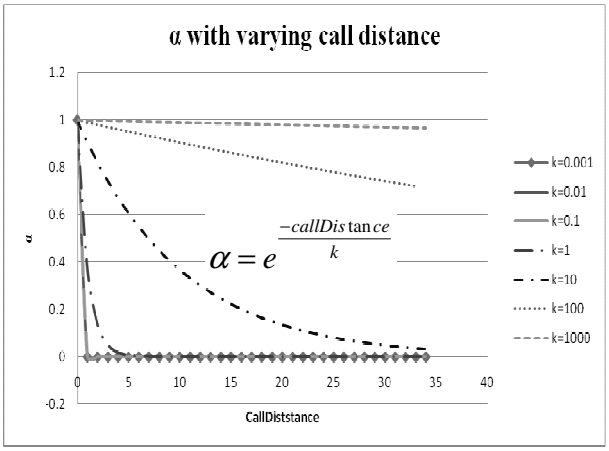

Fig. 6: Effect of *k* Values of Decay Ratio, α.

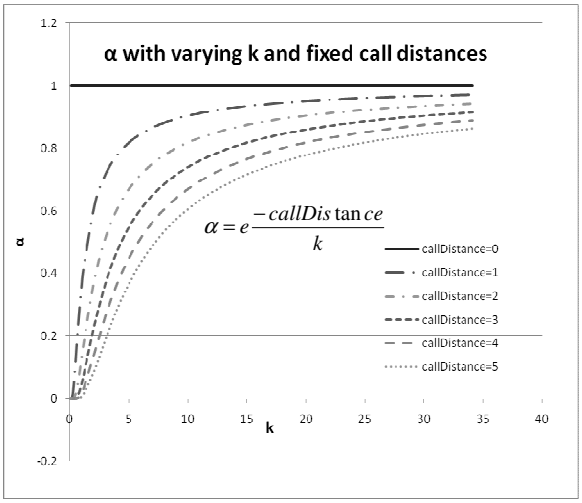

Fig. 7: Effect of CallDistance of Decay Ratio, α.

Given a primitive change, we use the impact calculator to count the LOC for the methods in the path of change propagation. We also use the impact calculator to compute our proposed impact value for each change. For the 102 primitive changes, we generate 102 impact values and 102 results for LOC metrics.

5.4 Selection of k values for the Impact Metric

As shown in Equations (3) and (4), the impact value is affected by the selection of *k*, which controls the

diminishing effect of the decay ratio, α. Fig. 6 illustrates the relation of decay ratio and call distance for different *k* values. Each line represents a fixed *k* value. As shown in Fig. 6, the decay ratio decreases as the call distance increases. The *k* values controls the steepness of the trend line. For example as shown in Fig. 6, when *k* is 0.001 or 0.01, the trend line is very steep. A small increase in call distance causes a large drop in the decay ratio. When the call distance is greater than 1, the decay ratio turns into close to 0, which indicates that the methods with a call distance greater than 1 are assigned a negligible weight in the change propagation. As *k* value grows, the steepness of the trend line decreases. For example when *k*=1000, the trend line is close to *y=1*. By observing different trend lines of *k* values, *k* of 10 has a smooth decay ratio when the call distance value increases. Fig. 7 illustrates the trend line between *k* values and decay ratio, α, with a fixed call distance. As shown in the Fig. 7, as *k* value increases, the differences in decay ratio, α, of different call distances is minimized.

We wish to select an appropriate *k* value such that the decay ratio, α, doesn't drop to close to 0 when the impact of the called methods with a larger call distance is considered. We select a *k* value of 10 which produce a smooth decrease in the decay ratio, α, when all the call distances are less than 3. For other larger systems a larger *k* value might be considered.

5.5 Design of Experiments

We build several linear regression models to test various aspects of our study. We perform the following four experiments.

1. ***Change Impact Metric vs. LOC.*** We study whether the impact metric correlates better than the base metric, LOC, with the time needed to perform a change.
2. ***Change Impact Metric vs. the Interaction of Subjects.*** Each subject has their own capability due to their familiarity with the code and their working styles. We explore if factoring in the subjects into our models will improve the correlations reported in our first experiment.
3. ***Change Impact Metric vs. the Interaction of LOC.*** Continuing our correlation experiments, we study the performance of a model which combines in all factors (our impact metric, LOC, and subjects).
4. ***Predictive Power of the Impact Metric vs. Predictive Power of LOC.*** We study the predictive power of our best correlation model using the impact metric and our best model using the LOC.

For the first three experiments, we build models which correlate the time needed to perform a change with various metrics such as LOC, and our impact metric. For the fourth experiment, we test the predictive power of our built models. For each experiment, we repeat the model building ten times using a ten-fold validation approach. For each repetition, we use 90% of the data to train the models (training data). We measure and compare the R^2 statistic for each model. R^2 measures the quality of the fit for each model. A value of zero for R^2 indicates a very low correlation and a value of one for R^2 indicates a very strong correlation. In our fourth experiment we study the predictive power of the models; therefore we use the 10% of the data (testing data) that were not used to build the model in each iteration (i.e., fold). For the fourth experiment, we measure and compare the prediction error of each model.

At the end of each experiment, we compare the means of either the R^2 statistic or the prediction error for the ten iterations. To ensure the statistical validity of our comparisons, we measure the statistical significance of the means using a Wilcoxon signed rank test are at a level of significance of 5 percent (that is, $p \leq 0.05$).

5.5.1 Change Impact Metric vs. LOC

To perform this experiment, we build two linear regression models for the impact metric and LOC metric shown as below:

$M1: time = \beta_1 * LOC + \beta_2$
$M2: time = \beta_3 * ImpactValue + \beta_4$

For each model, time represents the time needed to perform the change. M1 uses LOC and M2 relies on our impact metric. As shown in Table 3, the R^2 mean for M2 is greater than 5.68 of the R^2 mean for M1. The difference in means is statistically significant at $p < 0.005$. Therefore, our proposed impact metric correlates better than LOC with the time needed to perform a change.

Table 3. Statistical Results for Experiment 1

$\mu(R^2_{M1})$	$\mu(R^2_{M2})$	$\mu(R^2_{M1}) - \mu(R^2_{M2})$	$\frac{\mu(R^2_{M2})}{\mu(R^2_{M1})}$
0.091	0.519	-0.429	5.68

5.5.2 Change Impact Metric vs. Interaction of Subjects

The goal of this experiment is to determine whether our findings in experiment 1 hold true when we consider the capabilities of the subjects in the built linear regression models. We build two linear regression models, M3 and M4 which factor in the subjects as defined below.

M3: time = β_5 LOC + β_6*Subjects + β_7*
*M4: time = β_8*ImpactValue + β_9*Subjects + β_{10}*

Table 4. Statistical Results for Experiment 2

$\mu(R^2_{M3})$	$\mu(R^2_{M4})$	$\mu(R^2_{M3}) - \mu(R^2_{M4})$	$\frac{\mu(R^2_{M4})}{\mu(R^2_{M3})}$
0.569	0.869	-0.3	1.53

The Subjects variable encodes the six participants. As shown in Table 4, the R^2 mean for M4 is greater than the R^2 mean for M3. The difference in means is statistically significant at $p < 0.005$. Again our proposed impact metric correlates better than LOC with the time needed to perform a change. The R^2 when factoring in the Subject improves considerably relative to our first experiment.

5.5.3 Change Impact Metric vs. Interaction of LOC and Subjects

The goal of this experiment is to study if the correlation can be improved if we add the interaction of LOC to the best correlation model, i.e., M4, which uses the impact values and subjects as predictors. We build a linear regression model, M5 as shown below.

*M5: time =β_{11}*ImpactValue+β_{12}*Subjects+β_{13}*LOC+β_{14}*

Table 5. ANOVA Significance Values for Experiment 3

Factor	**Significance**
Impact Value	<0.05
Subject	<0.05
LOC	0.47

We perform an ANONA test to determine the statistical significance of the LOC parameter on the model, M5. As listed in Table 5, the results show that the LOC has no statistical significance to the model M5, and can be dropped from M5. Therefore, M4 is the best correlation model.

5.5.4 Predictive Power for the Impact Metric

Based on the last three experiments, we know that models M3 and M4 are the best two correlation models. However, we didn't test the predictive power of these models. In general, M3 and M4 can be expressed in a consistent form:

$$\text{time} = \gamma_0 * x + \gamma_1 * s + \gamma_2,$$

where x represents either LOC or impact value; and s describes a subject.

For each fold (i.e., repetition), we estimate γ_0, γ_1 and γ_2 using the training data of the fold then we measure the predication error using the testing data for that fold. Mathematically for every model with γ_0, γ_1 and γ_2 as parameters, we get a $\overline{time_i}$ for every x_i and s_i, where $\overline{time_i}$ is the predicted time need to make a code change. We define the absolute prediction error as $e_i = \left|\overline{time_i} - time_i\right|$, where $time_i$ is the actual time for performing one of the code changes in the testing data. Thus the total predictor error of a model is: $E = \sum_{i=1}^{n} e_i^2$, for all n in the testing data.

To evaluate the predictive power of both models, we compare the errors for M3 and M4. We perform a Wilcoxon signed rank test, similar to the previous experiments, at a level of significance of 5 percent (that is, $p \leq 0.05$). In particular, we formulate the following test hypotheses:

$$H_0 : \mu(e_{A,i} - e_{B,i}) = 0$$
$$H_A : \mu(e_{A,i} - e_{B,i}) \neq 0$$

Where $\mu(e_{A,i} - e_{B,i})$ is the population mean of the difference between the absolute errors for both models. If the null hypothesis H_0 holds (i.e., the derived p-value > 0.05), then the difference of two models is not significant and is likely due to the variability of the data. If H_0 is rejected with a high probability (that is, the derived P value ≤ 0.05), then we are confident about the prediction power of M4 relative to M3. Comparing the means of the errors for both models, we observe that the predictions by the M4 model, based on the impact metric, are on average 15% better than the predictions by the M3 model, based on the LOC. These findings are statistically significant with a p-value < 0.01.

5.5 Threats to Validity

In the case study, we evaluated the change impact metric using business applications contained in the OFBiz project. To ensure the generality of our findings, we should evaluate our work in more projects different from the OFBiz project. In particular, we should study the applications with longer call distances to investigate whether the different selections of k will affect the predictive power of models using the impact metric. The value of k which we calculated in Section 5 is obtained by analyzing OFBiz. Similar to other business applications, OFBiz are built using the MVC (Model-View-Controller) design paradigm. All applications that can be authored by different developers must follow a strict convention of MVC design pattern. In the future, we plan to investigate the choice of k value in other applications which are not built on top of MVC design patterns.

Six subjects were recruited for performing the changes to the code. Five out of six subjects are not familiar with the code of the applications. We also plan to conduct a larger case study with more participants with varying the knowledge of the code.

Our metric is based on static analysis of the code, therefore when propagating a change we may miss a change due to dynamic dependencies. Our metric could make use of dynamic dependencies however such dependencies are more time consuming to calculate. It would be interesting to measure the additional improvement in the accuracy of our metric estimates and to compare the improvement against the additional overhead.

Our analysis depends on the availability of source code to trace the propagation of changes in business tasks. The propagation stops when a third party component, such as database management systems, is encountered. Therefore, our proposed metric has limitation to evaluate the business changes involved in the third party components of the application when the source code for the third party components are not available.

6 Related Work

Software metrics are often used to predict software quality attributes, such as maintainability, reusability and modifiability [6, 7, 12, 13, 14, 18, 21, 35, 36]. For example, Bilal and Black [6, 7] present a ripple-effect algorithm that measures the complexity of an object-oriented software. Similar to our work, the ripple-effect algorithm traces the data flow of a variable in the program [35, 36]. In our research, we consider the diminishing effect of a variable on a change based on the distance of that variable from the initial change.

Other researchers have proposed techniques to perform impact analysis based on higher level abstraction than code [8, 10, 11, 29]. Ajila [1] investigates the change impact among four phases in the software development: requirements, specification, design and implementation. The changes performed in the earlier phases are analyzed to understand their impact on later phases. Instead of using business processes recovered from existing systems, de Boer et al. [8] study the ripple effect on software architecture description by modifying the relations in a software architectural design. In [10, 11], impact analysis is conducted using UML models. Soffer [29] provides techniques to analyze the change impact on business processes without considering the code that implements the business processes. In our work, we propagate the changes from a higher level of abstraction (i.e., business processes) to the lower level (i.e., implementation code).

Several research efforts perform impact analysis on the source code. Bohner et al. [9] focus on the source code changes that would affect the security aspect of the system. Hassan and Holt [17] conducted an empirical study on how changes to source code entities propagate across the entire software system. Hoffman [19] predicts modules to be affected based on maintenance changes. Lee [22] investigates the impact on legacy software and the challenges of modifying older code. Mendes et al. [26] provide impact analysis techniques for web applications. In [28], dynamic analysis is used to evaluate the impact of changes in software systems. The static analysis used in our approach cannot identify dynamic method invocation. Sneed proposed various models for estimating costs for performing maintenance tasks [37, 38]. Our work focuses on estimating the impact on the code. The impact on the code is one of the many factors affecting the overall cost of performing a maintenance task.

Traceability of code to the business process level was needed for our work. We use the techniques developed in [18, 31]. We refer the readers to these publications for additional detailed about the techniques to obtain the business processes. Other research on traceability focuses on tracing techniques from code to documentation [2, 5, 24, 27].

7 Conclusion and Future Work

We present an approach which propagates business process changes to the linked code entities. We propose a change impact metric which quantifies the time needed to perform a code change in response to

changes at the business process level. Our work can assist business analysts in determining the needed time for performing business process changes without having to study the source code. However, a modification in a business process might also impact other components different from the software system: human resources, technological resources and logistic components [34]. In our research, we only focus on the technical source code impact which provides a rough estimate and can be used as input to other processes to gain a complete picture of the overall impact. Our rough estimates are particularly important when customizing a vendor-specific business application to suit various needs of an organization during consulting engagements.

In the future, we plan to perform a more in-depth empirical case study to evaluate our approach on more business applications. This includes further justification of variables such as the propagation decay ratio. We want to investigate if developers with different level of knowledge of an application have an effect on the performance of our proposed change impact metric. We will refine the impact metric by examining more complex changes in the business processes, such as tasks changes combined with control flow changes.

References

1. S. Ajila "Software Maintenance: Approach to Impact Analysis of Objects Change", *Software-Practice and Experience*, Vol. 25(10), 1995, pp. 1155-1181.
2. G. Antoniol, G. Canfora, A. De Lucia, "Maintaining Traceability During Object-Oriented Software Evolution: a Case Study", *International Conference on Software Maintenance (ICSM),* 1999.
3. The Apache Open for Business Project, http://www.OFBiz.org/.
4. M. Asmild, J. C. Paradi and A. Kulkarni, "Using Data Envelopment Analysis in Software Development Productivity Measurement", *Software Process Improvement and Practice*, 2006, pp. 561-572.
5. G. Antoniol, G. Canfora, G. Casazza, A. D. Lucia, E. Merlo, "Recovering Traceability Links between Code and Documentation", *IEEE Transactions on Software Engineering*, Vol. 28(102002), pp. 970-983.
6. H. Bilal, S. Black, "Ripple Effect: A Complexity Measure for Object Oriented Software", *European Conference on Object-Oriented Programming*, 2007.
7. S. Black, "Computing the Ripple Effect" in *Journal of Software Maintenance and Evolution: Research and Practice*, 2001, pp. 263-279.
8. F.S. de Boer, M.M. Bonsangue, L.P.J Groenewegen, A.W.Stam, S.Stevens and L. Van Der Torre, "Change Impact Analysis of Enterprise Architectures", *Information Reuse and Integration Conference*, 2005.
9. S. Bohner and D. Gracanin, C. Dong, B. George, J. Ying and Y. Zhou, "Software Security Impact Analysis", *Goddard Space Flight Center - Advanced Architectures and Automation Branch*, 2002.
10. L. C. Briand Y. Labiche G. Soccar, "Automating Impact Analysis and Regression Test Selection. Based on UML Designs", *ICSM*, 2002.
11. L. C. Briand, Y. Labiche, L. O'Sullivan, "Impact Analysis and Change Management of UML Models", *ICSM 2003.*
12. S. R. Chidamber and C. F. Kemerer. "A Metrics Suite for Object Oriented Design", *IEEE Transactions on Software Engineering*, Volume 20(6), June 1994, pp. 476-493.
13. J. Daly, A. Brooks, J. Miller, M. Roper, and M. Wood, "The effect of inheritance on the maintainability of object oriented software: An empirical study", *ICSM* 1995.
14. S. L. Fleeger, S. A. Bohner, "A Framework for Software Maintenance Metrics", ICSM 1990.
15. T. Graves, A. Karr, J. Marron, and H. Siy, "Predicting Fault Incidence Using Software Change History", IEEE Transactions on Software Engineering, 25(7), July 2000.
16. J. Guo, K. Foo, L. Barbour, Y. Zou, "A Business Process Explorer: Recovering and Visualizing E-Commerce Business Processes", International Conference on Software Engineering, 2008.
17. A. E. Hassan and R. C. Holt, "Predicting Change in Propagation in Software Systems", *ICSM*, 2004.
18. S. Henry and D. Kafura, "The Evaluation of Systems' Structure using Quantitative Metrics", *McGraw-Hill International Series*, 1993, pp.99-111.
19. M. A. Hoffman, "Automated Impact Analysis of Object-Oriented Software Systems", *International Conference on Object-Oriented Programming, Systems, Languages, and Application*s, 2003, pp.72-73.
20. M. Huang, Y. Zou. "Recovering Workflows from Multi Tiered E-commerce Systems", *International Conference on Program Comprehension,* 2007, pp.198-207.
21. H. Kagdi and J. I. Maletic, "Combining Single Version and Evolutionary Dependencies for

Software Change Prediction", *International Conference on Software Engineering*, 2007.
22. M. Lee, "Change Impact Analysis of Object Oriented Software", *George Mason University,* 1998.
23. M. Lipow, Number of Faults per Line of Code, IEEE Transactions Volume SE-8, Issue 4, July 1982, Pages 437-439.
24. A. Marcus, JI. Maletic, Recovering Documentation-to-Source-Code Traceability Links using Latent Semantic Indexing. ICSM, 2003.
25. T. McCabe, and C. Butler, "Design Complexity Measurement and Testing", Communications of the ACM 32, 12 (December 1989), pp. 1415-1425.
26. E. Mendes, N. Mosley and S. Counsell, "Web Metrics – Estimating Design and Authoring Effort", *IEEE Multimedia*, 2001, pp. 50-57.
27. A. De Lucia, R. Oliveto, G. Tortora, "Recovering Traceability Links using Information Retrieval: a Controlled Experiment", *International Symposium on Grand Challenges in Traceability*, 2007, pp. 46-55.
28. A. Orso, T. Apiwattanapong, J. Law, G. Rothermel, and M. J. Harrold, "An Empirical Comparison of Dynamic Impact Analysis Algorithms", *International Conference of Software Engineering*, Scotland, UK, 2004.
29. P. Soffer, "Scope Analysis – Identifying Impact of Changes in Business Process Models", *Software Process Improvement and Practice*, 2005, pp.393-402.
30. WebSphere Business Modeler. http://www-306.ibm.com/software/integration/wbimodeler/index.html. June 2009.
31. H. Xiao, J. Guo, Y. Zou, "Supporting Change Impact Analysis for Service Oriented Business Applications", *International Conference on Software Engineering Workshops*, 2007.
32. XML Process Definition Language (XPDL), http://www.wfmc.org/standards/docs.htm#XPDL_Spec_Final. April 2008.
33. Y. Zou, T. C. Lau, K. Kontogiannis, T. Tong, R. McKegney. "Model Driven Business Process Recovery", *Working Conference on Reverse Engineering,* 2004.
34. L. Aversano, T. Bodhuin, and M. Tortorella, "Assessment and Impact Analysis for Aligning Business Processes and Software Systems", Proc. of ACM Symposium on Applied Computing, SAC 2005.
35. S. Barros, T. Bodhuin, A. Escudie, J.P. Queille, J.F. Voidrot, "Supporting Impact Analysis: a Semi-Automated Technique and Associated Tool", ICSM, 1995.
36. J. O'Neal, D. Carver, "Analyzing the Impact of Changing Requirements", ICSM, 2001.
37. H. Sneed, "Estimating the Costs of software maintenance tasks", ICSM, 1995.
38. H. Sneed, "Impact Analysis of Maintenance Tasks for a Distributed Object-oriented System", ICSM, 2001.
39. B. Chan, L. Marks, Y. Zou, "An Approach for Estimating Code Changes in E-Commerce Appllications", International Symposium on Web Systems Evolution, 2008.