# Version Control on Database Schema and Test Cases from Functional Requirements' Input Changes

Parichat Kiatphao and Taratip Suwannasart

*Abstract*— **In software engineering, version control is an important activity for change management. Functional requirements are essential parts to communicate software behaviors, which may include inputs of a corresponding function. Functional requirements may be changed during the software development process. Consequently, inputs in a functional requirement might relate to other parts of software artifacts such as database schema, and test cases. If changes are occurred more frequently, they lead to difficulties in maintaining data consistency. Moreover, the data will possibly be lost if the system is built without a version control, so it is hard to revert to the previous state. This paper proposes an approach for version control on database schema and test cases from the functional requirements' input changes. The proposed approach applies a successive versioning method with the backward versioning strategy. Thus, the proposed approach can be adopted in implementing a tool for effective version control.**

*Index Terms*— **Version Control, Functional Requirements' Input Changes, Database Schema, Test Case**

## I. Introduction

CHANGES could be occurred at any phases in the software development life cycle. When changes occur, their effect may possibly impact on many parts of software artifacts, e.g. Functional Requirements (FR), and test case. Basically, functional requirements are essential part of software artifacts that define and communicate one or more specific behaviors or functions that a software must perform [1]. A function that is enclosed within the functional requirements, is described as a set of inputs in order to produce expected outputs. The input in the functional requirement is related to the database schema in which the instance data of the input are stored. In general, Database Schema takes the responsibility to define characteristics of data in the database. Therefore, if users change the characteristics of the inputs, such as data type or length, it is possible that the database schema is affected by the changes.

Unfortunately, database schema is not the only affected artifact. Test cases are also affected because they are used to validate a software function in conformance with predefined functional requirements. If a set of the functional requirement's inputs is changed, test cases might be added, updated or deleted. Furthermore, since the relationship between functional requirements and test cases can be traced by the Requirement Traceability Matrix (RTM), the RTM is also required to update following the changes of test cases and the functional requirements.

If changes are occurred more frequently, they lead to the difficulty in maintaining data consistency because of the increasing number of data versions. In this case, it is necessary to control the evolution of the software and provide an ability to revert the state of data if there is a mistake. From the purpose of dealing with multiple changes, version control is needed because it is the task of keeping software system (or artifacts) consisting of many versions and configurations well-organized [2].

Previously, there is a research that proposed an approach for analyzing the impact from changes to the inputs of functional requirements [3], which also focuses on updating database schema and test cases to make software works properly as usual. However, their update procedure is to replace older version without storing the historical data. According to the previous research, if there is any mistake on the change, it is hard to revert to the previous version.

In this paper, we present an approach for version control on database schema, test cases and RTM where one or more inputs of functional requirements are changed. We have designed an approach for keeping versions by applying the concept of backward versioning strategy which supports reverting the previous version if the latest change is mistaken.

We have organized the rest of this paper as follows: Section 2 describes related work. Section 3 describes the necessary background knowledge. Section 4 presents the proposed approach for version control. Finally, conclusion and future work are discussed in section 5.

## II. Related Work

To investigate the feasibility, we first studied the alternative method for version control. E. J. Choi et al., presented a method for the version control by using a tree data structure [4]. They presents their method named *HiP (History in Parent)* which uses concept of backward versioning strategy. Their method maintains the history node as a sequence of pointers to each node which applied the "*Save_History*" algorithm. This algorithm is started by storing the beginning of the node list in head pointer which points to the current node (or current version). The current

Manuscript received January 07, 2017; revised January 23, 2017.

P. Kiatphao and T. Suwannasart are with the Software Engineering Lab, Center of Excellence in Software Engineering, Department of Computer Engineering, Faculty of Engineering, Chulalongkorn University, Bangkok, Thailand, E-mail: Parichat.Kia@student.chula.ac.th,Taratip.S@chula.ac.th.

node contains a pointer to the previous node, and so on. Finally, the last node (or first version) has its pointer points to NULL as a mark of the end of the list. This algorithm shows that the change history of a node will be maintained at its parent node explicitly. However, their method limits the application of version control in term of the program source code only. This paper applies their method for version control, especially the part of *Save_History* algorithm in order to perform the version control on database schema and test cases.

Another related research was proposed by A. Kampeera et al. [3], in which they presented an approach to analyze the impact of functional requirements' input changes to database schema and test cases. They also generated a SQL command for updating schema directly. Moreover, a related test case, which is identified by the RTM, will be updated. However, their approach used the update operation by replacing an old version of functional requirements and test cases with a new version directly. The aforementioned operation may consider as a drawback of their approach because it leads to difficulty in reverting to any previous state of the data if the change is mistaken. Hence, our approach offers a solution to these problems by collecting a variety of data version as a change history automatically. The collection of change history will allow users for tracking and reverting changes more effectively.

## III. Background

### A. Version Control

Version control is a challenging task or activity for software development and maintenance in software engineering [2]. Version control helps managing changes for maintaining software (or software artifact) versions in order to provide an ability of tracking and reverting changes to the specified version. Version control also refers to an activity for keeping the old version of a software (or software artifact) when it is changed, and a new version is created [4].

Version control can be performed in one of two methods: the *Full Copy* and the *Delta*. The Full Copy method maintains versioned data entirely in each version. The modification is done by duplicating as a new file with same name but different version numbers. The Delta method maintains only one complete version and reconstructs other versions from the difference between each version. There are two strategies used in the Delta method [5]. The *forward versioning strategy* which maintains the oldest version as a complete version and keeps the version of the difference between each version and the oldest one. The *backward versioning strategy* which maintains the current version as a complete version and keeps the version from the difference between each version and the current one.

## IV. Proposed Approach

In this section, we will describe our proposed approach for version control on database schema and test cases when functional requirements' inputs are changed. The framework for constructing version control environment following our proposed approach consists of six steps:

1) Initiate the Corresponding Data
2) Analyze and Store Change Data
3) Analyze and Update the Impact of Change
4) Store Change History and Control Version
5) Cancel the Latest Change
6) Display Result

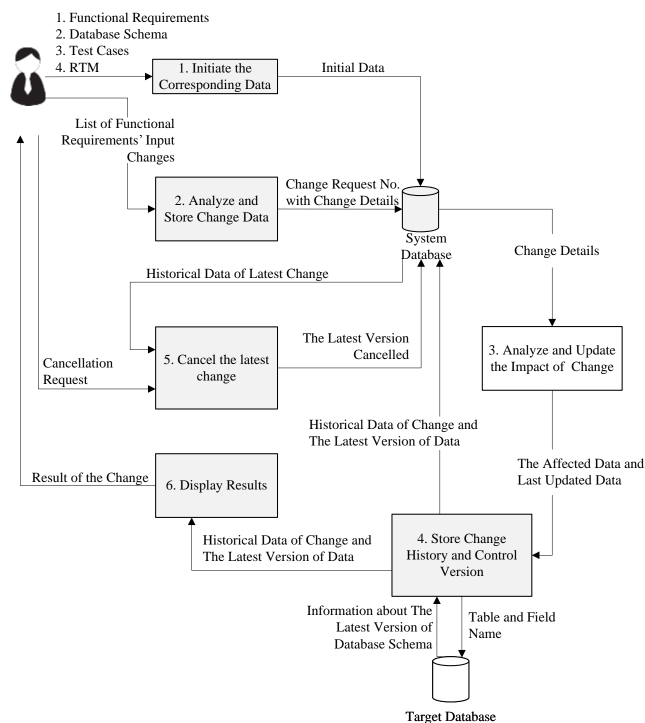


Fig. 1. Conceptual Framework for Version Control on Database Schema and Test Case from Functional Requirements' Input Changes

Fig. 1 shows the behavioral representation of our proposed approach. Each step is described and given the example respectively in the following subsections.

### A. Initiate the Corresponding Data

First, a user initiates the corresponding data for the system, including (1) Functional Requirements which consist of a functional requirement ID (FR Id), a description (FR Description), a version (FR Version), and a set of inputs. The inputs are specified by name, data type, data length, constraints, and its relation to database schema. (2) Database schema which consists of table names, field names as well as their specifications including data types, data lengths, and constraints. (3) Test cases which consist of a test case ID, a related functional requirement ID, a test case version, an expected result, and inputs with associated test data. (4) A requirement traceability matrix or RTM which consists of the relations between functional requirements and test cases.

### B. Analyze and Store Change

In this step, we analyze the effect on functional requirements' input(s) from a user request. This paper focuses on three types of change:

1) Add new inputs
2) Delete inputs
3) Update inputs, including input name, data type, data length, and constraints.

Assume that we have an original version of functional requirements as shown in Table I. If the change is occurred

as a change request shown in Table II, we will analyze the change for storing the detail of the changed input into a system database and generating a new unique identifier (Change Request No.) to the change request.

TABLE I
THE FUNCTIONAL REQUIREMENT FR-01 (VERSION 1.0)

| **FR Id** | FR-01 | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **FR Description** | Insert student information | | | | | | | | |
| **FR Version** | 1.0 | | | | | | | | |
| **List of Inputs** | | | | | | | | **Relation** | |
| | | | **Constraints** | | | | | | |
| **Input Name** | **Data Type** | **Length** | Unique | Default | Null | Max | Min | **Table Name** | **Field Name** |
| Student Id | varchar | 10 | Y | - | N | - | - | student | studentId |
| First Name | nvarchar | 45 | - | - | N | - | - | student | firstName |
| Last Name | nvarchar | 45 | - | - | N | - | - | student | lastName |
| Faculty | varchar | 2 | - | - | N | - | - | faculty | facNo |
| Department | varchar | 2 | - | - | N | - | - | departme nt | deptNo |
| Year | int | 1 | - | 1 | N | 1 | 4 | student | year |
| Mobile | varchar | 15 | - | - | Y | - | - | student | mobile |

Table II shows that three inputs of a functional requirement is changed due to the change request 'CR-01' as described as follows: First, ‘Student Id’ is changed its data type from VARCHAR to CHAR and its length from 10 to 9 characters. Second, a ‘Mobile’ input field is deleted entirely. Finally, a new input named ‘Email’ is added to the functional requirements, with initiation of its attributes and constraints.

TABLE II
AN EXAMPLE OF LIST OF INPUT CHANGES

| **Change Request No.** | | CR-01 | |
|---|---|---|---|
| **FR Id** | | FR-01 | |
| **No** | **Input Name** | **Changes** | |
| | | **Type** | **Description** |
| 1 | Student Id | Edit | Data Type (‘varchar’, ‘char’), Data Length (‘10’, ‘9’) |
| 2 | Mobile | Delete | Delete a field |
| 3 | Email | Add | Data Type (‘’, ‘varchar’) Data Length (‘’, ‘50’), Unique (‘’, ‘Y’), Default (‘’, ‘’), Null (‘’, ‘N’), Max (‘’, ‘’), Min (‘’, ‘’), Table (‘student’), Field (‘email’) |

### C. *Analyze and Update the Impact of Change*

In this paper, we assume that the change impact analysis will be done by a stub service which behaves as same as the actual service proposed in [3]. Thus, we use the detail of the change request from step B to analyze the change impact and update associated data. In this moment, the schema of the target database will be updated to the current state. The results of impact analysis are as follows:

1) Affected Functional Requirements
2) Affected Test Cases
3) Affected tables and fields in the target database.
4) Affected Requirements Traceability Matrix

### D. *Store Change History Data and Version Control*

In this step, the change history is recorded after a stub service returned its result. The analysis result presents a relationship of the corresponding data between each version as shown in Table III and IV, and the result is stored into the system database as well.

TABLE III
HISTORICAL CHANGE DATA OF THE AFFECTED FUNCTIONAL REQUIREMENT

| **Change Request No.** | CR-01 |
|---|---|
| **FR Id** | FR-01 |
| **Old Function Version** | 1.0 |
| **New Function Version** | 2.0 |

Table III shows an example of historical data belonging to the change request ‘CR-01’ which indicates that the functional requirement ‘FR-01’ has affected by this change and a new version of ‘FR-01’ was created. Therefore, the version of ‘FR-01’ is changed from 1.0 to 2.0.

TABLE IV
THE AFFECTED FUNCTIONAL REQUIREMENT IN DETAILS

| **Change Request No.** | | CR-01 | | |
|---|---|---|---|---|
| **FR Id** | | FR-01 | | |
| **Input Name** | | Student Id | Mobile | Email |
| **Change Type** | | Edit | Delete | Add |
| **Data Type** | Old | VARCHAR | VARCHAR | |
| | New | CHAR | | VARCHAR |
| **Data Length** | Old | 10 | 15 | NULL |
| | New | 9 | NULL | 50 |
| **Unique** | Old | | | NULL |
| | New | | | Y |
| **Null** | Old | | Y | NULL |
| | New | | NULL | N |
| **Default** | Old | | | |
| | New | | | |
| **Max** | Old | | | |
| | New | | | |
| **Min** | Old | | | |
| | New | | | |

* The blank cells represent data that does not change anything.

Table IV shows an example of the change details for the functional requirement ‘FR-01’. The change details show that the functional requirement ‘FR-01’ has been changed in three inputs from the change request ‘CR-01’ and each input is compared between old and new version. For instances, ‘Student Id’ field in the functional requirement ‘FR-01’ has reduced its length from 10 to 9 characters.

Furthermore, from the change request ‘CR-01’, test cases that are related to the functional requirement ‘FR-01’ are also affected, e.g. the test case ‘TC-01’ contains test data with length of 10 characters. We illustrate the change history for the affected test case as shown in Table V and VI.

TABLE V
CHANGE HISTORY OF THE AFFECTED TEST CASE

| **Change Request No.** | CR-01 |
|---|---|
| **Test Case No.** | TC-01 |
| **Change Type** | Edit |
| **Old Test Case Version** | 1.0 |
| **New Test Case Version** | 2.0 |

Table V shows an example of historical data belonging to the change request ‘CR-01’ which indicates that the test case

'TC-01' has affected and changed its version from 1.0 to 2.0. In additional, our approach supports three test case change types, including *Addition, Deletion*, and *Editing* of an affected test case.

Since the length of 'Student Id' field of the functional requirement 'FR-01' have been changed, Table VI shows that the test input 'Student Id' and its corresponding test data are affected and makes this test case inapplicable. Consequently, the test input and its test data need to be changed. Moreover, there are two more test inputs that are also affected by this change and need to be handled as well.

TABLE VI
THE AFFECTED TEST CASE IN DETAILS

| **Change Request No.** | | CR-01 | | |
|---|---|---|---|---|
| **Test Case No** | | TC-01 | | |
| **Input Name** | | Student Id | Mobile | Email |
| **Change Type** | | Edit | Delete | Add |
| **Test Data** | Old | 5870947021 | 02-333999 | |
| | New | 876987893 | | Test@a.com |

Other than that the change probably affects to the corresponding database schema and it needs to be updated. The change history of the database schema is also recorded. We illustrate the change history for the impact on the corresponding database schema as shown in Table VII.

TABLE VII
CHANGE HISTORY OF THE AFFECTED DATABASE SCHEMA

| **Change Request No.** | | CR-01 | | |
|---|---|---|---|---|
| **Schema Table Name** | | student | student | student |
| **Schema Field Name** | | studentId | mobile | email |
| **Change Type** | | Edit | Delete | Add |
| **Old Schema Version** | | 1.0 | 1.0 | |
| **New Schema Version** | | 2.0 | | 1.0 |
| **Data Type** | Old | VARCHAR | VARCHAR | |
| | New | CHAR | | VARCHAR |
| **Data Length** | Old | 10 | 15 | |
| | New | 9 | | 50 |
| **Unique** | Old | | | |
| | New | | | Y |
| **Null** | Old | | Y | |
| | New | | | N |
| **Default** | Old | | | |
| | New | | | |
| **Max** | Old | | | |
| | New | | | |
| **Min** | Old | | | |
| | New | | | |

Table VII shows that the database schema of table 'student' need to be changed with respect to the functional requirement 'FR-01'. Thus, the change history keeps the difference between each version, e.g. the database field 'studentId' in the database table 'student' has changed its data type from VARCHAR to CHAR and the length from 10 to 9 characters as same as the corresponding functional requirement 'FR-01'. In addition, the new version of the current database schema can be kept directly from the target database by using a SQL command after a stub service has updated.

From the aforementioned example of the versioning step, we can classify our strategy used as the backward versioning strategy for version control because a new or current version is created after completing an update and treat as a complete version. Other than that the new or current version is linked to the previous one before disabling it. We follow the backward versioning strategy as it can be represented in the same way as shown in Fig. 2.

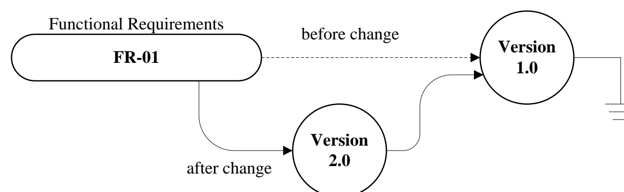


Fig. 2. Example of the new version creation using backward versioning strategy

In this paper, we apply the backward versioning strategy into an atomic level of each data, i.e. each test case, each functional requirement, and each field of database schema.

Finally, the updated functional requirement 'FR-01' can be represented as a new version as shown in table VIII. The current version is stored in the system database as a complete version.

TABLE VIII
THE LATEST VERSION OF FUNCTIONAL REQUIREMENT FR-01

| **FR Id** | | | FR-01 | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **FR Description** | | | Insert student information | | | | | | |
| **FR Version** | | | 2.0 | | | | | | |
| **List of Inputs** | | | | | | | | **Relation** | |
| **Input Name** | **Type** | **Length** | **Constraints** | | | | | **Table Name** | **Field Name** |
| | | | Unique | Default | Null | Max | Min | | |
| Student Id | char | 9 | Y | - | N | - | - | student | studentId |
| First Name | nvarchar | 45 | - | - | N | - | - | student | firstName |
| Last Name | nvarchar | 45 | - | - | N | - | - | student | lastName |
| Faculty | varchar | 2 | - | - | N | - | - | faculty | facNo |
| Department | varchar | 2 | - | - | N | - | - | department | deptNo |
| Year | int | 1 | - | 1 | N | 1 | 4 | student | year |
| Email | varchar | 50 | Y | - | N | - | - | student | email |

Also, the updated version of other affected data is stored in the system database as well. For instances, the complete or current version of the affected test case 'TC-01' in the system database can be shown in Table IX.

TABLE IX
THE LATEST VERSION OF TEST CASE TC-01

| **Test Case Id** | TC-01 |
|---|---|
| **Test Case Version** | 2.0 |
| **FR Id** | FR-01 |
| **Expected Output** | Valid |
| **List of Inputs** | |
| Input Name | Test Data Value |
| Student Id | 876987893 |
| First Name | Julie |
| Last Name | Ann |
| Faculty | 05 |
| Department | 10 |
| Year | 1 |
| Email | Test@a.com |

### *E. Cancel the Latest Change*

In real-life situation, the change can possibly be cancelled

ISBN: 978-988-14047-3-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

by any mistaken after the change has committed. In this case, our approach supports the change cancellation of the latest change only. In other word, our approach provides an ability to revert the version but our approach does not allow cancelling any version in the middle of the version sequence.

The cancelled change will be analyzed to find if the data version corresponds to the cancelled change, e.g. the change from change request 'CR-03' is cancelled. From the analysis, the result appears that this change affects the functional requirement 'FR-02' and 'FR-03' from version 2.0 to 3.0 and 1.0 to 2.0 respectively. Thus, if the change request 'CR-03' is cancelled, the functional requirement 'FR-02' and 'FR-03' will be reversed to version 2.0 and 1.0 respectively by removing the cancelled version of data. The linkage between versions can be found explicitly in the change history we stored in step D.

### *F. Display Result*

After all operation is done, the result of version control will display with respect to the change (or change cancellation), to inform the user about the following information:

1) The change request information – this information describes about who request the change, the date of request, and which functional requirements' input the request intend to change.
2) Details of the effect on functional requirements, test cases, RTM and database schema from the change request.
3) The latest version of the corresponding data that are changed from the change request.

## V. Conclusion

We have proposed our approach for version control on database schema and test cases from functional requirements' input changes. Using our approach, the record of all changed data are kept as a historical data (or change history). Then, the current state of data will be updated as a new or current version. The creation of a new or current version are handled by using the concept of backward versioning strategy and apply 'Save_History' algorithm. Moreover, by following our approach, the latest change is allowed to be cancelled by removing the current version of the data in order to reverse to the previous version. In the future, we will develop a tool from our proposed approach. Subsequently, the developed tool will be evaluated by measuring the data accuracy.

## References

[1] *IEEE Standard for Application and Management of the System Engineering Process*, IEEE Standard 1220, 2005.

[2] W. F. Tichy, "RCS—a system for version control," *Softw. Pract. Exper*., vol. 15, no. 7, pp. 637-654, July. 1985.

[3] A. Kampeera and T. Suwannasart, "Impact Analysis to Database Schema and Test Case from Inputs of Functional Requirements Changes," in *Lecture Notes in Engineering and Computer Science: Proceedings of The International MultiConference of Engineers and Computer Scientists 2016*, pp. 449-453.

[4] E. J. Choi and Y. R. Kwon, "An Efficient Method for Version Control of a Tree Data Structure," *Softw. Pract. Exper*., vol. 27, no. 7, pp. 797-811, July. 1997.

[5] P. Dadam, V. Y. Lum and H. –D. Werner, "Integration of Time Versions into a Relational Database System," in *Proc. 10th Int. Conf. on Very Large Data Bases,* San Francisco, 1984, pp. 509-522.J. Wang, "Fundamentals of erbium-doped fiber amplifiers arrays (Periodical style—Submitted for publication)," *IAENG International Journal of Applied Mathematics*, submitted for publication.