Computer Science and Information Systems 18(1):129–154 https://doi.org/10.2298/CSIS200221032J

A method of assessing rework for implementing software requirements changes

Shalinka Jayatilleke[1] and Richard Lai[2]

[1] Department of Management, Sports and Tourism
La Trobe University, Victoria. 3086, Australia
s.jayatilleke@latrobe.edu.au
[2] Department of Computer Science and Computer Engineering
La Trobe University, Victoria. 3086, Australia
R.Lai@latrobe.edu.au

Abstract - Software development is often affected by user/system requirements changes. To implement requirements changes, a system which is being developed needs to be reworked. However, the term "Rework" has not been clearly defined in the literature. Depending on the complexity of the changes, the amount of rework required varies from some software module modifications to a non-trivial alteration to the software design of a system. The effort associated with such a rework obviously will vary too. To date, there has been scant research on rework assessment, and the relationship between it and change effort estimation is hardly understood. In this paper, we present a definition for rework, and describe a method of assessing rework for implementing software requirements changes. Our method consists of three stages: namely (i) change identification; (ii) change analysis; and (iii) rework assessment. To demonstrate the practicality that it enables developers to compare the rework between the different options available for implementing a requirements change and to identify the one which is less invasive and requires lesser amount of modifications to the software system design, we explain our concept with the use of a running example.

Keywords - Rework, Rework assessment, Requirements changes, Requirements change management, Software System Design Document

1. Introduction

Software development is often affected by changes in user/system requirements. Rapid changes in requirements are found to be one of the main cost drivers [1]; and they have a significant impact on development efforts and project duration [2, 3]. To implement requirements changes, a system in design phase or later (but not yet deployed) needs to be reworked on. However in the literature, the term "*Rework*" has not been uniformly and clearly defined as past practitioners and researchers considered terms like "reconsideration", "re-instantiation", "redoing" and "revision" as synonymous with rework, while the Oxford Dictionary defines "*Rework*" as "making changes to the original version of something". Depending on the complexity of the changes, the amount of rework required varies from some software module modifications to a non-trivial alteration to software design of a system. The cost associated with such a rework obviously will vary too.

According to our systematic review on Requirements Change Management (RCM) [4], there are three main components in managing requirements changes: change identification, change impact analysis and change cost/effort estimation. Effort estimation is about calculating and predicting the effort of a set of activities before they are actually performed [5, 6]; and effort is a value usually expressed in terms of time and/or dollars. Subsequently, change effort estimations are to predict the cost and time required for implementing a change. Such estimations are important as underestimation can result in budget overrun, poor quality and delay in project completion; whereas overestimation may result in the allocation of too many resources which will cause inefficiency [6]. Accurate estimation can also help assess the feasibility of implementing a change, prioritize the implementation of the requested changes and determine the cost of the implementation of a change.

Prior to conducting a change effort estimation, we need to have a better understanding of the extent to which and how a system would be reworked as it is possible to have more than one option for implementing a change and different options require different amounts of rework to be made to a system. In such a situation, estimation might need to be done for each option in order to determine its suitability. It should be noted that with the complexity of the changes requested and the number of implementation options available, change effort estimation can be a tedious and time-consuming task. It would therefore be beneficial to have a method which can identify the implementation option which involves the lesser amount of rework, before any estimation is carried out; and a lot of time will be saved by not having to conduct the unnecessary estimations. Given the importance of rework for estimation, the relationship between them is hardly understood.

In our systematic review [4], we explained how existing estimation methods and models can be applied to effort estimation related to implementing requirements changes and pointed out the fact that general effort estimation models may not be suitable for estimating the effort of implementing a requirements change. There are a few models that deal specifically with requirements change effort/cost estimation as discussed in the related work section of our systematic review paper [4]. Most existing methods use expert judgment which is based on the experience of the estimator which is not a consistent component and expert judgement also relies on past project data which may not be applicable to a particular case where there is no historical data. It is therefore important that the interrelations and dependencies between systems functions are identified for estimating the effort/cost of changes as the dependencies will have an impact on an implementation. An inherent drawback in most existing estimation methods is that these dependencies are not well understood [6].

To date, there is limited research on the concept of rework for software development and assessing rework for implementing requirements changes. In this paper, we first present our definition of rework and then describe a method of assessing rework for implementing software requirements changes in the context of its definition. Our method consists of three stages: (i) identification of the change; (ii) identification of the activities within the software system design which are affected by the change; and (iii) assessing the rework required. Stages 1 and 2 are based on the concepts and ideas described in our two previously published papers and the results of applying Stages 1 and 2 enable Stage 3 to be carried out. Stage 3 involves the computations of: (i) Interaction Comparison (IC); (ii) Interaction Weight (IW); and (iii) Rework which is based on IC and IW. To demonstrate the practicality that it enables developers to

compare the rework between the different options available for implementing a requirements change and to identify the one which is less invasive and requires lesser amount of modifications to the software system design, we have applied our method to a running example for a reader's better comprehension of it.

2. The concept of rework and our proposed definition

The concept of rework exists in fields outside software development. In the field of medicine, doctors may need to rework treatment plans for patients who have developed unexpected reactions; in the building industry, civil engineers may need to rework plans for the load bearing of a bridge depending on future traffic conditions; academics will need to rework course and/or subject material depending on assessment outcomes or feedback by students. Several studies in civil engineering defined rework as "the unnecessary effort of redoing a process or activity that was incorrectly implemented the first time" [7, 8].

Rework is common in software development due to changes emanating from clients, development environment, and laws of the government and society. We discussed the causes of these changes extensively in our systematic review [4]. A key activity in RCM is to identify the amount of rework required for the proposed changes, as this will have a significant impact on the time and cost of a project. Studies show that normally rework leads to additional effort and cost [9-14] of a project. However, a clear relationship between rework and effort estimation has not been understood/established. Some studies proposed methods for reducing the amount of rework [10, 15], yet the fact remains that there will still be a considerable amount of rework to deal with. In the agile software development environment, it encourages rework instead of attempting to eliminate it [4, 16]. Rework is often unavoidable as the understanding of a problem and its possible solutions evolve over time.

Rework is a central activity in the development of software. The cost of rework is said to reach or even exceed 50% of the total project cost [9-11, 17]. These costs are one of the main concerns in software development since it is an important parameter defining the success of software projects [12, 13]. According to Charrette [14], software developers spend 40-50% of their time on rework activities. Based on the above facts, rework is generally considered as an important software development activity. In software development, Zhao and Osterweil [15] define rework as "the re-instantiation of tasks previously carried out in earlier development phases in a richer context that is provided by the activities and artifacts that had been performed and created during subsequent phases". In a simpler manner, Ghezzi et al. [17] suggest that rework consists of "going back to a previous phase" of software development to redo decisions made or work carried out in that previous phase". It is clear that the concept of rework has been subject to different interpretations. In short, rework has not been uniformly and well defined [10, 15, 18].

Based on the discussion above and our systematic review findings, we are of the opinion that the concept of rework needs to be more narrowly focussed on the following items:

- Requirements changes are the reasons for doing it;
- Instead of being broadly considered as a software development activity, it is one which falls in the area of RCM; and
- It is closely related to change cost/effort estimation which is also a RCM activity.

Our proposed definition of *rework* is therefore as follows:

"Rework in the field of software engineering is an activity within the area of Requirements Change Management (RCM), which makes modifications/alternations to a system which has a software design document and is being developed (pre-delivery) for implementing certain requirements changes, with the alternations/modifications normally introducing extra work and increasing the total amount of cost/effort for completing the software project; and assessing rework, a preliminary step to change cost/effort estimation which is another RCM activity, is about studying how a system needs to be modified/altered for implementing the changes."

According to this definition, we establish that rework is an activity conducted prior to the delivery of a system. Given that a software design document is necessary, rework assessment can be applied to any stage of software development as long as a software system design document is available; and it is independent of the type of software development methodology (be it waterfall or agile). With Agile Software Development, a design document becomes available as the development progresses and therefore rework assessment becomes plausible.

Another noteworthy point is that there is a key difference between our definition of rework and maintenance. According to IEEE standard 1219, software maintenance is defined as "the process of modifying a software system or component after delivery to correct faults, improve performances or other attributes, or adapt to a changed environment"[19, 20]. The post-delivery nature of maintenance is also emphasised similarly in the ISO/IEC [ISO95] definition [20, 21]. The modifications to a system during the maintenance phase will always preserve the integrity of the software product [20, 22]. If the software design of a system needs to be altered substantially, the alternation will not be done as a piece of maintenance work but rework which will lead to new version of the software product. An example is that Microsoft usually release a newer version of its Windows operating system every period of say 3-4 years, or sometimes shorter.

3. Overview of the method of assessing rework

We anticipate that our method of assessing rework enable us to understand to what extent a system needs to be altered for implementing the required changes. Based on our previously developed methods of specification and classification [23, 24], we have identified that some requirements changes can be implemented in more than one way, which we refer to as change implementation possibilities/options. We aim to realize the following:

1. A numerical representation of the assessment of rework required to implement a requirement change for all possible implementation options.
2. Selection of the option which requires a lesser invasive to the software design of the system, ergo is of lesser rework.

3. Comparison of the assessment of rework between multiple requirements changes.

This method is a continuation of the findings of the specification and classifications methods [23, 24] and change analysis methods [25] previously established by the authors. The use of these methods in the rework method is detailed in Figure 1.

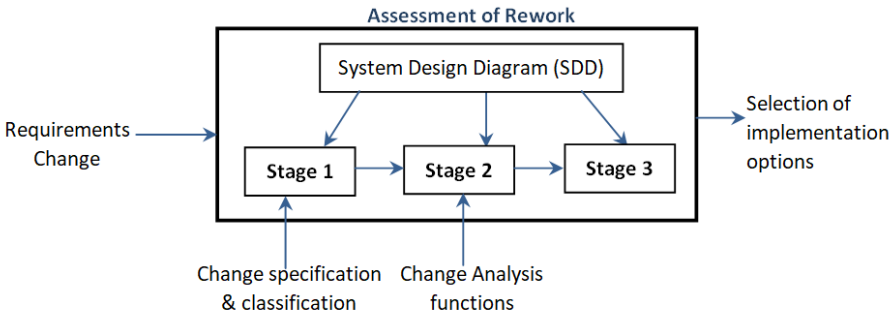


Fig. 1. Overview of the method

According to Figure 1, the method will use as input the requirements changes and the system design diagram (SDD). The output of the method is executed in three stages:

Stage 1: Identification of the change

The change is identified and categorised using the methods which we have developed and reported in [23, 24].

Stage 2: Identification of the system activities affected due to the change

Once the change is identified, we apply the change analysis functions of the change analysis method which was developed and reported in [25]. As a result, the change is further exposed, enabling us to identify the activities that are directly affected by the change. Using the SDD, we then map the directly affected activities (DAA) to identify the indirectly affected activities (IdAA). The IdAA are the activities that are connected to DAA through input and/or output. IdAAs are considered in this assessment as modifications to a DAA which may have a direct impact on the activities associated with the DAA via the input-output links. The SDD can be any design diagram that shows the relationships between different objects and activities. Typically various forms of UML [26] diagrams such as activity diagrams, class diagrams, etc. can be used for this purpose.

Stage 3: Assessing the rework required

Once all the activities related to the change are identified, we can assess the rework. In order to do this, we adopt the methods that are introduced in [27-29]. From [27] and [28], the concept we adopt is referred to as interaction frequency. This frequency refers to the ratio of the number of interactions (input-output) performed by the affected operations (of a change) and the number of interactions performed by all operations of the interface. A similar concept is used in [29] where instead, the number of interfaces are used. Given that the interactions between the activities are identified and indicated in the

SDD, we can use this concept to assess the rework and thereafter make a selection of the implementation option with a lesser rework.

4. The details of the method

In this section, we describe in detail, the stages presented above.

4.1. Identification of the changes – Stage 1

To identify a requested requirements change, we use the change specification and classification methods which we have developed and reported in [23, 24]; and a summary of them can be found in Appendix 1. Change specification denotes a way of specifying a change so that communication ambiguities between business and IT staff can be avoided. Once a requirements change has been initiated from the client side, this method will use the SDD as input to map the location of the change. In order to create the specification template, we use two established methods, i.e. Goal Question Metrics (GQM) [30] and the Resource Description Framework (RDF) [31]. We also use a set of additional questions to enable better identification when using the specification template output.

The change classification method uses the outcomes of the specification template to expand on the type of change along with preliminary guidance on the action to be taken in managing the change. The classification itself is based on the concepts of the change taxonomy found in the existing change management literature [32-35] and is refined using the unstructured interviews of practitioners in the field of change management. The outcomes of the change classification will provide software developers with a better understanding of what the change is and offers preliminary guidance on how the change implementation can be carried out. The detailed change classification is shown in Table 1. The term link mentioned in Table 1 refers to the input-output connection between the activities. The term activity refers to the process activities in a SDD.

At implementation time, the key elements of the two methods (specification and classification) are incorporated into a single table (see Table 2). In the table, change number refers to the number given to each change as they are requested. The object, purpose and focus in Table 2 correspond to the specification method (please refer to Appendix 1).

Table 1. Detailed change description

Change focus	Answer to Additional Question	**Change type**	**Action**
Add	No	Matched links	Add new activity without changing the current activity or any connected links
	Yes	Mismatched links	Add new activity by changing the activity and/or connected links
Modification	No	Inner property modification	Modify the implementation of an activity without changing the connected links
	Yes	Input data modification	Modify the input link and internal properties of an activity
	Yes	Output data modification	Modify the output link and internal properties of an activity
Delete	No	Matched links	Delete activity without changing connected activities
	Yes	Mismatched links	Delete activity by changing connected activities and links
Activity Relocation	No	Relocation with matched links	Relocate existing activity without changing the activity or connected links
	Yes	Relocation with mismatched links	Relocate new activity by changing the activity and/or connected links

Object → the activity name according to the SDD (this is the activity affected by the change)
Purpose → the reason for the change
Focus → the activity selected from the list - Add, Delete, Modification or Activity relocation

Change type and action can be sourced from Table 1 based on the information provided for the object, focus and additional question, respectively. The option columns represent how each change may be described using different foci. This may not apply to all changes. This feature was added to the implementation template to provide more diversity and flexibility for communicating a change. Having multiple options also provides flexibility as to how the change can be implemented.

Table 2. Template for implementation

	Change No.	Option 01	Option 02	Option n
Specification Method	**Object**			
	Purpose			
	Focus			
	Additional Question			
	RESULT			
Classification Method	**Change type**			
	Action			

Running example

In order to explain this stage and the following stages, we will consider the following running example.

Diskwiz is a company which sells DVDs by mail order. Customer orders are received by the sales team, which checks that the customer details have been completed properly on the order form (for example, delivery address and method of payment). If they are not, a member of the sales team contacts the customer to obtain the correct details. Once the correct details are confirmed, the sales team passes a copy of the order to the warehouse team to pick and pack, and a copy to the Finance team to raise an invoice. Finance raises an invoice and sends it to the customer within 48 hours of the order being received. When a member of the warehouse team receives the order, they check the real-time inventory system to make sure the discs ordered are in stock. If they are, they are collected from the shelves, packed and sent to the customer within 48 hours of the order being received, so that the customer receives the goods at the same time as the invoice. If the goods are not in stock, the order is held in a pending file in the warehouse until the stock is replenished, whereupon the order is filled. This process is illustrated in the following system design diagram.

The change scenario:

The management is not satisfied with some parts of the process and points out that the following issue should be rectified: “It is identified, due to a design error, there is no communication between Finance and the Warehouse to confirm discs are in stock so that the order can be shipped. Therefore, Finance could be raising invoices when the order has not been sent.”

One of the reasons for having no communication between Finance and Warehouse is because there is no communication between A_4 and A_5, where A_4 represent one activity of the Warehouse and A_5 represents Finance. Another way to view this would be that there is no communication between A_5 and A_6, where A_6 is another activity of the Warehouse.

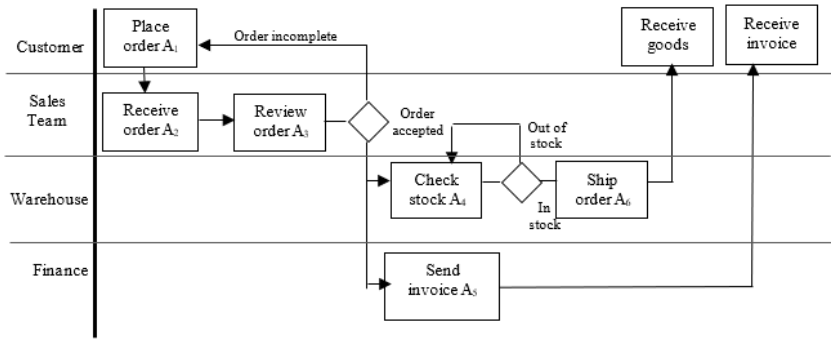


Fig. 2. Diskwiz customer order fulfilment process diagram

Therefore, the objects to be considered are A_4, A_5 and A_6. The purpose of this change is to resolve an existing design issue (according to the change scenario). A software engineer may use different focus based on the different combinations of objects which are A_4, A_5 and A_6 (based on the views taken in the above paragraph).

Table 3 is the complete application of Stage 1 of the rework method, i.e. this is a populated form of Table 2. If we consider option 01 in Table 3, the objects selected are A_4 and A_5, with a purpose of resolving a design error. Based on the rationale given between the non-existence of communication between the two objects, it would be feasible to add this communication between the objects. Therefore, the "Add" focus was chosen as the focus. Now this focus can be mapped to Table 1. From this point onwards, the rest of the fields in Table 2 will follow the directions related to the change focus "Add" in Table 1. The same rationale can be applied to options 02 and 03.

Table 3. Change classification outcome

Change 01	**Option 01**	**Option 02**	**Option 03**
Object	A_4 and A_5	A_4 and A_5	A_5 and A_6
Purpose	Resolution of design error	Resolution of design error	Resolution of design error
Focus	Add	Modify	Modify
Additional Question	Need addition input/output? Y	Input/output modification? Y	Input/output modification? Y
Result			
Change Type	Add new function between A_4 and A_5 (Mismatched links)	Inner property modification and output data modification A_4 and input data modification of A_5	Inner property modification and output data modification A_6 and input data modification of A_5
Action	Add new function by changing the function and/or connected links of A_4 & A_5	Modify A_4 to send message to A_5	Modify A_6 to send message to A_5

4.2. Identification of the system activities affected by the change(s) – Stage 2

We use a part of the change analysis method which we have developed and reported in [25] for expanding further the change identified; and a summary of this analysis method can be found in Appendix 2. Using this expansion, both DAAs and IdAAs are identified using the system design diagram. The change analysis functions are based on the change foci identified in [23, 24]: add, delete, modify and relocation. We use the category of primary change analysis functions to expand the changes. The category of primary functions can be used for building a block of more complex functions. The need to do this is due to the fact that it is hard to project every possible way of implementing the changes so practitioners can use this type of block to help them facilitate the changes.

The following terminologies are used for the functions:

The term activity in this method is used to represent process activities in the SDD.

A_N – New activity, A_O – Old activity, A_T - Target activity, Pt – Pointer, A_R – Relocating activity, A_C – Connected activity

V – Value: the value passed onto the function for data manipulation

L – Link: the connection between two activities

The primary category consists of the following set of functions:

1. Function to create a new activity
   CreateFunc(String, V) $\rightarrow A_N$
2. Function to link a new activity with existing activities
   CreateLink(A_N, A_O, V)
3. Function to link existing activities
   CreateLink($A_{X\text{-}O}$, $A_{Y\text{-}O}$, V)
4. Function to delete an activity
   DeleteFunc(A_O)
5. Function to delete links between activities
   DeleteLink($A_{X\text{-}O}$, $A_{Y\text{-}O}$)
6. Function to modify inner property of an activity
   ModifyInner(A_T,V)
7. Function to modify input data of an activity
   ModifyIn(A_S, A_T, V)
8. Function to modify output data of an activity
   ModifyOut(A_S, A_T, V)
9. Function to create a pointer to an existing activity
   CreatePointer(Pt, A_T)
10. Function to delete a pointer
    DeletePointer(Pt)

Once the change has been expanded, the activities identified in the functions are mapped to the SDD. These are the DAAs. In the SDD, any activity connected as the input and/or output of a DAA is considered an IdAA.

Running example stage 2

In accordance with this example and Table 3, the change can be implemented using one of the three options. In stage 2, we apply the preliminary functions from the change

analysis method for the 3 options and we generate the following expansions of the change:

Table 4. Expansion of change options

Option 1	Option 2	Option 3

CreateFunc(String, V) $\rightarrow A_N$
CreateLink(A_N, A_4, V)
{
ModifyInner(A_4,V)
ModifyIn(A_N, A_4, V)
ModifyOut(A_N, A_4, V)
}
CreateLink(A_N, A_5, V)
{
ModifyInner(A_5,V)
ModifyIn(A_N, A_5, V)
ModifyOut(A_N, A_5, V)
}
	ModifyInner(A_4,V)
	CreateLink(A_4, A_5, V)
	ModifyOut(A_4, A_5, V)
	ModifyIn(A_4, A_5, V)
		ModifyInner(A_6,V)
		CreateLink(A_5, A_6, V)
		ModifyOut(A_6, A_5, V)
		ModifyIn(A_6, A_5, V)

Based on Table 4, we are able to identify the DAAs for each option. Then by mapping the DAAs to the SDD, we are able to identify the IdAAs for each DAA. In this paper when selecting the IdAAs, we consider only the first impact level. Investigation of further levels can be considered as a future enhancement, which is outside the scope of this paper.

Table 5. Identification of DAAs and IdAAs

Options	DAAs	IdAAs
1	A_4	A_3, A_6
	A_5	A_3
2	A_4	A_3, A_6
	A_5	A_3
3	A_5	A_3
	A_6	A_4

4.3. Assessing the rework required – Stage 3

Through the numerical values generated, we are able to assess the rework to be carried out as a result of the change. In order to ensure the assessment of the rework is based on both the total interactions of the activities to be reworked as well as the difficulty level of implementing the change action, we use the number of affected interactions as well as the change weights introduced in the change analysis method [25]. The values for the weights are adopted from [36]. It has been established that in the change analysis method, each change action / type has a different difficulty level. Therefore, this difficulty level needs to be represented in the rework.

The assessment of the work required to implement a change involves the following calculations:

1. The interaction comparison (IC) of the affected activities (direct and indirect)

2. The interaction weight (IW) using the change weights of the affected activities (direct)
3. The rework based on IC and IW

As a result of the values generated from IC and IW, developers will have a numerical view of the assessment of the rework for implementing a change. If there are more than one option of implementation, then based on the combination of IC and IW, the developer can choose the lesser invasive option, which would result in the option with lesser rework.

When choosing the lesser invasive option, first preference is given to the lesser value of IC as this denotes lesser number of connections in the software design of the system will need to be altered. In the event that the IC value is the same for two or more options, IW will be considered. Use of IW is explained in the following sections.

Interaction comparison (IC) Calculation

Interaction comparison is the identification of the percentage of interactions that need to be altered in order to accomplish the required change. An interaction is a connection between two or more process activities (input-output links) in a SDD. This is in comparison to the total number of interactions identified in the SDD. Using the SDD, the following steps are used to calculate IC:

- For each activity (DAAs and IDAAs) involved in the change, identify the number of interactions. These interactions will be the number of connections each activity has with the other activities of the system.
- Identify the total number of interactions in the entire system.
- Calculate IC.

Running example stage 3 (IC calculation)

Using the above example, we show how the value of IC is calculated for all the options.

IC calculation for option 1:

The number of interactions for each identified activity based on Table 5 is as follows:
A_4 – has 2 interactions (Connected to A_3 and A_6)
A_5 – has 1 interaction (Connected to A_3)
A_3 – has 4 interactions (Connected to A_1, A_2, A_4 and A_5)
A_6 – has 1 interaction (Connected to A_4)

Considering all the interactions, the system design contains six activities. The interaction count for each activity is as follows:
A_1 – has 2 interactions (Connected to A_2 and A_3)
A_2 – has 2 interactions (Connected to A_1 and A_3)
A_3 – has 4 interactions (Connected to A_1, A_2, A_4 and A_5)
A_4 – has 2 interactions (Connected to A_3 and A_6)
A_5 – has 1 interaction (Connected to A_3)
A_6 – has 1 interaction (Connected to A_4)
The way of calculating the value of IC is adopted from [27].

$$IC_{CO} = \frac{N_I}{N_{TI}}$$

Formula 1: IC caclulation

Where CO is the Change Option number, N_I is the number of interactions per change action and N_{TI} is the total number of interactions for the system according to the SDD.

$$N_I = \sum_{x=1}^{n} N_{Ix}$$

Formula 2:No. of interactions affected by change

where x is the number of activities affected by the change action and N_{Ix} is the interactions for each affected activity.

$$N_{TI} = \sum_{x=1}^{n} N_{TIx}$$

Formula 3: Total no. of interactions in the system

where x is the total number of activities of the system and N_{TIx} is the interactions for each activity.

Applying to the example option 1:

When calculating N_I we consider the interaction of all the activities (DAAs and IdAAs) of option 1 which include: A_4, A_5, A_3 and A_6 (extracted from Table5). Based on the interactions identified for these activities, N_I is;

$$N_I = 2 + 1 + 4 + 1 = 8 \qquad (1)$$

When calculating N_{TI} interactions of all the activities are considered. Based on the interactions identified for all activities, N_{TI} is;

$$N_{TI} = 2+2+4+2+1+1 = 12 \qquad (2)$$

$$IC_1 = \frac{8}{12} = 67\% \qquad (3)$$

According to this value, when considering option 1 for change implementation, 67% of all the interactions have to be altered in order to implement the required change.

IC calculation for option 2:

The number of interactions for each identified activity based on Table 5 is as follows:
A_4 – has 2 interactions (Connected to A_3 and A_6)
A_5 – has 1 interaction (Connected to A_3)
A_3 – has 4 interactions (Connected to A_1, A_2, A_4 and A_5)
A_6 – has 1 interaction (Connected to A_4)

The total number of interactions is the same as that of option 1
Therefore;

$$IC_2 = \frac{N_I}{N_{TI}} \qquad (4)$$

Applying the same principles as option 1;

$$N_I = 2 + 1 + 4 + 1 = 8 \qquad (5)$$

$$N_{TI} = 2+2+4+2+1+1 = 12 \qquad (6)$$

$IC_2 = \frac{8}{12} = 67\%$

According to this value, when considering option 2 for change implementation, 67% of all the interactions have to be altered for implementing the required change.

IC calculation for option 3:

The number of interactions for each identified activity based on Table 5 is as follows:
A_5 – has 1 interaction (Connected to A_3)
A_6 – has 1 interaction (Connected to A_4)
A_4 – has 2 interactions (Connected to A_3 and A_6)

The total number of interactions is the same as that of option 1
Therefore;

$$IC_3 = \frac{N_I}{N_{TI}} \quad (7)$$

Applying the same principles as option 1;

$$N_I = 1 + 1 + 2 = 4 \quad (8)$$

$$N_{TI} = 2+2+4+2+1+1 = 12 \quad (9)$$

$$IC_3 = \frac{4}{12} = 33\% \quad (10)$$

According to this value, when considering option 3 for change implementation, 33% of all the interactions have to be altered for implementing the required change.

Interaction weight (IW) Calculation

The interaction weight is the change weight corresponding to the directly affected interactions due to the requirements change. The change weight concept was established in the change analysis method [25]. The weights for the change categories are assigned, using the principles described in [36] and [37] and based on the knowledge they have gained in working in the industry as well as extensive research on requirements change management. In both studies the change weights are incorporated in mathematical formulas which compute a change complexity. IW adds depth to the IC value by providing a numerical representation of the difficulty level of implementing the change and how this relates to the interactions. The value of IW becomes further important in assessment and selection, when the value for IC can be the same for different options of a given change, as we demonstrated in the running example. We establish that the lower the IW, the less difficult it would be to implement a change. In order to calculate IW, the following steps are used:

- Identify the change types using the expanded change action steps (Stage 2).
- Calculate the total change weight based on the change analysis method.
- Use the interactions and the total change weight to calculate IW.

In order to calculate IW, we consider only the activities directly affected by the change. This is because the identification of change types are acquired from stage 2 where it only contains DAAs.

From the change expansion in stage 2, we consider the change functions *Create*, *Modify* and *Delete* when calculating IW.

Running example stage 3 (IW calculation)

Using the same running example, we use the outcome of Table 4 to identify the change types as follows:

Table 6. Change weight identification

Option 1	Option 2	Option 3

CreateFunc(String, V) $\rightarrow A_N$
CreateLink(A_N, A_4, V)
{
ModifyInner(A_4,V)
ModifyIn(A_N, A_4, V)
ModifyOut(A_N, A_4, V)
}
CreateLink(A_N, A_5, V)
{
ModifyInner(A_5,V)
ModifyIn(A_N, A_5, V)
ModifyOut(A_N, A_5, V)
}
	ModifyInner(A_4,V)
	CreateLink(A_4, A_5, V)
	ModifyOut(A_4, A_5, V)
	ModifyIn(A_4, A_5, V)
		ModifyInner(A_6,V)
		CreateLink(A_5, A_6, V)
		ModifyOut(A_6, A_5, V)
		ModifyIn(A_6, A_5, V)

Create Functions – 3 Modify Functions – 6 Delete Functions – 0	Create Functions – 1 Modify Functions – 3 Delete Functions – 0	Create Functions – 1 Modify Functions – 3 Delete Functions – 0

Using the weighting system introduced in the change analysis method, we develop Table 7 to calculate the change weight (CW):

- All create functions will have the Add weight of 3
- All modify functions will have the Modify weight of 2
- All delete functions will have the Delete weight of 1
- All other functions are a combination of the main three functions i.e. create, modify and delete

Table 7. Change weight calculation

Change Type	Option 1	Option 2	Option n
Add	No. of functions × CW Add	No. of functions × CW Add	…. × ….
Modify	No. of functions × CW Mod	No. of functions × CW Mod	…. × ….
Delete	No. of functions × CW Del	No. of functions × CW Del	…. × ….
Total CW			

Applying the findings of the running example of Table 6:

Table 8. Calculated change weights

Change Type	Option 1	Option 2	Option 3
Add	3 × 3 = 9	1 × 3 = 3	1 × 3 = 3
Modify	6 × 2 = 12	3 × 2 = 6	3 × 2 = 6
Delete	N/A	N/A	N/A
Total CW	21	9	9

$$IW_{CO} = \left(\sum_{X=1}^{n} N_{CO}\right) \times \sum CW_{CO}$$

Formula 4: IW calculation

where CO is the Change Option number and N_{CO} is the number of interactions per change action where only interactions of the DAAs are considered. We reiterate the reason for only considering DAAs is they are directly attached to the change actions (as seen in Table 4) and IdAAs are not. The number of interactions for the DAAs was identified when calculating the IC value. CW_{CO} is the total change weight for that option as shown in Table 8.

Applying the equation to the running example:

For option 1:
The directly affected activities are A_4 and A_5. Therefore,

$$N_1 = 2+1 \quad (11)$$

$$CW_1 = 21 \quad (12)$$

$$IW_1 = (2 + 1) \times 21 = 63 \quad (13)$$

For option 2:
The directly affected activities are A_4 and A_5. Therefore,

$$N_2 = 2+1 \quad (14)$$

$$CW_2 = 9 \quad (15)$$

$$IW_2 = (2 + 1) \times 9 = 27 \quad (16)$$

For option 3:
The directly affected activities are A_5 and A_6. Therefore,

$$N_3 = 1+1 \quad (17)$$

$$CW_3 = 9 \quad (18)$$

$$IW_3 = (1 + 1) \times 9 = 18 \quad (19)$$

Rework calculation based on IC and IW

In section IC Calculation, IC was established to be the percentage of interactions that need to be altered in order to facilitate the required change and in section Running example stage 3, IW was established to be the change weight corresponding to the directly affected interactions due to the requirements change. Based on these two

values, the assessment of rework is a combined look at both the interactions that need to be altered in comparison to the full system depicted in the SDD and the difficulty of implementing the change action on those interactions. In order to display the comparison between the rework required for the changes requested and their multiple options, we use Table 9 as a template.

Table 9. Template of comparison between rework

	Change 1			Change 2	Change n
	Opt 1	Opt 2	Opt n		
IC					
IW					

Running example stage 3 (rework calculation)

To better understand this template, we populate it with the outcome of the running example:

Table 10. Outcome of comparison

	Change 1		
	Opt 1	Opt 2	Opt 3
IC	67%	67%	33%
IW	63	27	18

According to this example, one change was requested with three possible actions that can be taken to implement it. According to the above table, the value of IC is the same for options 1 and 2. Option 3 has a lower IC value than that of options 1 and 2. This is a good indication that option 3 is the lesser invasive option for implementing the change as a lesser number of interactions has to be altered. This fact is further validated by the IW value where option 3 has the lowest IW value corresponding to a lower difficulty level of implementing the change.

Based on the above results, it can be said that:

- option 1 and 2 require 67% of the interactions to be altered while option 3 requires only 33% alterations;
- based on IW, option 3 has a lesser difficulty level of implementation as compared to the other options; and
- therefore, the lesser invasive change implementation is option 3, based on both the IC and IW values.

5. Comparison with the related work

To the best of our knowledge, in the literature there has been no paper published on assessment of rework in the area of RCM. However, we are able to find two papers in the literature which focus on effort estimation related to implementation of requirements changes. Although these methods do not assess rework, they use requirements changes

and their impact in the calculation process in a similar manner to our method. We shall discuss below a comparison with these two pieces of work.

Requirements changes can occur at any phase of the development process and even after deployment. There are few estimation methods dedicated to change effort/cost estimation and the importance of such methods were established in the introduction. The following discussion elaborates on two methods that deal specifically with change effort/cost estimation that use a similar rationale to the method introduced in this paper.

The estimation method introduced by Jeziorek [29] attempts to estimate the cost of the impact of a design change to development. The author emphasises the importance of identifying the functional requirements and design parameters that are impacted by the change, before attempting to estimate the cost of change. He uses this identification in the form of a matrix to detect the physical interactions between components. These physical interactions are used to determine how the change propagates through the system. The model developed in [29] outputs the affected components, how they are affected and what the cost of impact will be. In this particular method, the use of interactions between components and the mapping of the propagation of the change through the system are similar activities as used in our method.

In the method established by Lavazza and Valetto [38], several different artifacts are used to calculate the change costs. The key feature of this method is the use of requirements instead of lines of code to calculate the cost. Therefore, the method utilizes the design document and traceability techniques for estimation. The estimation is carried out in two stages: 1) characteristics such as the size and the complexity of the code are estimated on the basis of the size of the complexity of the requirements and the skill and experience of the implementation team; 2) effort is estimated based on the knowledge of the relations that link the inputs, outputs and the resources required. Most parts of the estimation are based on historic data. The use of requirements to establish the complexity and the linking of inputs and outputs resonate with the rework method introduced in this paper.

We use the aforementioned work to describe the limitations of the existing work and compare our methods to define what has been achieved. The limitations focus only on the techniques comparable with our method.

Table 11. Comparison with related work

Technique	Limitations	What our method addresses
Jeziorek [29]	Initially, a lot of time needs to be spent in developing the matrices needed to identify the impact. These matrices are non-transferable and therefore for every project, new matrices need to be established.	New diagrams are not needed. The method uses the system diagram which a software project would usually have.
Lavazza and Valetto [38]	The use of historical data which may not be available for some projects and is therefore limited to systems development that has such data. The use of traceability methods that have inherent limitations such as informal development methods, insufficient resources, time and cost for traceability, lack of coordination between people responsible for different traceable artifacts, imbalance between benefits obtained and effort spent implementing traceability practices, and construction and maintenance of a traceability scheme proves to be costly [39-46]	The method uses data only from the current project. The change identification and analysis techniques used in this method do not use traceability techniques and therefore do not have the drawbacks associated with traceability techniques.

6. Conclusions and future work

In this paper, we have presented a definition of rework – "*Rework in the field of software engineering is an activity within the area of Requirements Change Management (RCM), which makes modifications/alternations to a system which has a software design document and is being developed (pre-delivery) for implementing certain requirements changes, with the alternations/modifications normally introducing extra work and increasing the total amount of cost/effort for completing the software project; and assessing rework, a preliminary step to change cost/effort estimation which is another RCM activity, is about studying how a system needs to be modified/altered for implementing the changes."* We have also described a method of assessing rework for implementing software requirements changes. Once a change has been proposed, our method identifies the paths of implementation, which lead to the identification of the impacted activities of the system through the SDD. Using these activities, two values (IC and IW) are computed to help assess the rework required for all the possible options. Based on the IC and IW values, a developer can choose the lesser invasive option which requires lesser rework.

To demonstrate the viability of our method, we have applied it to the Diskwiz customer order fulfilment process as a running example. For the requested requirements change, we generated multiple implementation options and for each option, IC and IW were calculated. We have shown that when multiple options of implementation exist for one change, IC alone is not sufficient to make an assessment and selection. In the example, the change resulted in two options, which have the same IC value for implementations. In such scenarios, IW plays an important role in the assessment process. Based on the values of IC and IW, the rework was assessed, and comparisons were then made between the implementation options of a change and we were able to identify which option requires a lesser amount of rework.

The results of applying our method to this running example indicates that it is useful in the area of RCM. It enables developers to have a better understanding of the rework required by different options for implementing change. Given the fact that the implementation path is extracted from the SDD, our method can be applied during any phase of the software development, provided that the design document is available.

We can thus conclude that it can serve as a precursor to change effort estimation, whereby it is not necessary to carry out estimation for all the possible implementation options but the one which has been assessed to involve a lesser amount of rework. Hence, a related future work would be to develop a change effort method for estimating the time and the cost required for implementing a change.

References

1. B. W. Boehm, "Software engineering economics," in *Pioneers and Their Contributions to Software Engineering*: Springer, 2001, pp. 99-150.
2. S. Ferreira, J. Collofello, D. Shunk, G. Mackulak, and P. Wolfe, "Utilization of process modeling and simulation in understanding the effects of requirements volatility in software development," in *International Workshop on Software Process Simulation and Modeling, Portland, Oregon*, 2003.
3. D. Pfahl and K. Lebsanft, "Using simulation to analyse the impact of software requirement volatility on project performance," *Information and Software Technology*, vol. 42, no. 14, pp. 1001-1008, 2000.
4. S. Jayatilleke and R. Lai, "A systematic review on Requirement Change Management," *Information and Software Technology*, vol. 93, pp. 163-185, 2018. DOI: 10.1016/j.infsof.2017.09.004., doi: 10.1016/j.infsof.2017.09.004.
5. D. Kiritsis, K.-P. Neuendorf, and P. Xirouchakis, "Petri net techniques for process planning cost estimation," *Advances in Engineering Software*, vol. 30, no. 6, pp. 375-387, 1999.
6. H. Leung and Z. Fan, "Software cost estimation," *Handbook of Software Engineering, Hong Kong Polytechnic University*, pp. 1-14, 2002.
7. P. E. D. Love, D. J. Edwards, H. Watson, and P. Davis, "Rework in Civil Infrastructure Projects: Determination of Cost Predictors," *Journal of Construction Engineering and Management*, vol. 136, no. 3, pp. 275-282, 2010, doi: doi:10.1061/(ASCE)CO.1943-7862.0000136.
8. P. E. D. Love, "Influence of Project Type and Procurement Method on Rework Costs in Building Construction Projects," *Journal of Construction Engineering and Management*, vol. 128, no. 1, pp. 18-29, 2002, doi: doi:10.1061/(ASCE)0733-9364(2002)128:1(18).
9. K. Butler and W. Lipke, "Software process achievement at tinker air force base," Technical Report CMU/SEI-2000-TR-014, Carnegie-Mellon Software Engineering Institute (September 2000), 2000.

10. A. G. Cass, S. M. Sutton, and L. J. Osterweil, "Formalizing rework in software processes," in EWSPT, 2003, vol. 2786: Springer, pp. 16-31.
11. F. CeBASE eWorkshop, "Focusing on the cost and effort due to software defects," NSF Center for Empirically Based Software Engineering, 2001.
12. V. R. Basili, S. E. Condon, K. E. Emam, R. B. Hendrick, and W. Melo, "Characterizing and modeling the cost of rework in a library of reusable software components," presented at the Proceedings of the 19th international conference on Software engineering, Boston, Massachusetts, USA, 1997.
13. U. T. Raja, M.J., "Defining and Evaluating a Measure of Open Source Project Survivability," IEEE Transactions on Software Engineering, vol. 38, no. 1, pp. 169-174, 2012.
14. R. N. Charette, "Why software fails [software failure]," IEEE Spectrum, vol. 42, no. 9, pp. 42-49, 2005.
15. X. O. Zhao, L.J., "An approach to modeling and supporting the rework process in refactoring," in International Conference on Software and System Process (ICSSP), 2012, pp. 110-119.
16. J. Highsmith and A. Cockburn, "Agile software development: The business of innovation," Computer, vol. 34, no. 9, pp. 120-127, 2001.
17. C. Ghezzi, M. Jazayeri, and D. Mandrioli, Fundamentals of software engineering. Prentice Hall PTR, 2002.
18. P. E. Love and J. Smith, "Benchmarking, benchaction, and benchlearning: rework mitigation in projects," Journal of Management in Engineering, vol. 19, no. 4, pp. 147-159, 2003.
19. J. Radatz, A. Geraci, and F. Katki, "IEEE standard glossary of software engineering terminology," IEEE Std, vol. 610121990, no. 121990, p. 3, 1990.
20. K. H. Bennett and V. T. Rajlich, "Software maintenance and evolution: a roadmap," in Proceedings of the Conference on the Future of Software Engineering, 2000: ACM, pp. 73-87.
21. ISO12207 Information technology - Software life cycle processes, I. I. S. Organisation, Geneva, Switzerland, 1995.
22. G. Canfora and A. Cimitile, "Software maintenance," in Handbook of Software Engineering and Knowledge Engineering: Volume I: Fundamentals: World Scientific, 2001, pp. 91-120.
23. S. Jayatilleke and R. Lai, "A method of specifying and classifying requirements change," in Software Engineering Conference (ASWEC), 2013 22nd Australian, 2013: IEEE, pp. 175-180.
24. S. Jayatilleke, R. Lai, and K. Reed, "Managing Software Requirements Changes through Change Specification and Classification," Computer Science and Information Systems, vol. 15, no. 2, pp. 321-346, 2018, doi: 10.2298/CSIS161130041J.
25. S. Jayatilleke, R. Lai, and K. Reed, "A method of requirements change analysis," Requirements Engineering, pp. 1-16, 2017. DOI: 10.1007/s00766-017-0277-7., doi: 10.1007/s00766-017-0277-7.
26. P. Selonen, K. Koskimies, and M. Sakkinen, "Transformations between UML diagrams," Journal of Database Management, vol. 14, no. 3, p. 37, 2003.
27. T. Wijayasiriwardhane and R. Lai, "Component Point: A system-level size measure for component-based software systems," Journal of Systems and Software, vol. 83, no. 12, pp. 2456-2470, 2010.
28. S. Mahmood and R. Lai, "A complexity measure for UML component-based system specification," Software: Practice and Experience, vol. 38, no. 2, pp. 117-134, 2008.
29. P. N. Jeziorek, "Cost estimation of functional and physical changes made to complex systems," Massachusetts Institute of Technology, 2005.
30. R. Van Solingen, V. Basili, G. Caldiera, and H. D. Rombach, "Goal question metric (gqm) approach," Encyclopedia of Software Engineering, 2002.
31. M. Weiss, "Resource description framework," in Encyclopedia of Database Systems: Springer, 2009, pp. 2423-2425.

32. N. Nurmuliani, D. Zowghi, and S. P. Williams, "Requirements volatility and its impact on change effort: Evidence-based research in software development projects," in Proceedings of the Eleventh Australian Workshop on Requirements Engineering, 2006.
33. S. McGee and D. Greer, "A software requirements change source taxonomy," in Software Engineering Advances, 2009. ICSEA'09. Fourth International Conference on, 2009: IEEE, pp. 51-58.
34. N. Nurmuliani, D. Zowghi, and S. P. Williams, "Using card sorting technique to classify requirements change," in Requirements Engineering Conference, 2004. Proceedings. 12th IEEE International, 2004: IEEE, pp. 240-248.
35. H. Xiao, J. Quo, and Y. Zou, "Supporting change impact analysis for service oriented business applications," in Systems Development in SOA Environments, 2007. SDSOA'07: ICSE Workshops 2007. International Workshop on, 2007: IEEE, pp. 6-6.
36. Y. Li, J. Li, Y. Yang, and M. Li, "Requirement-centric traceability for change impact analysis: a case study," in Making Globally Distributed Software Development a Success Story: Springer, 2008, pp. 100-111.
37. S. Maadawy and A. Salah, "Measuring Change Complexity from Requirements: A Proposed Methodology," ed: IMACST, 2012.
38. L. Lavazza and G. Valetto, "Requirements-based estimation of change costs," Empirical Software Engineering, vol. 5, no. 3, pp. 229-243, 2000.
39. J. Cleland-Huang, C. K. Chang, and M. Christensen, "Event-based traceability for managing evolutionary change," Software Engineering, IEEE Transactions on, vol. 29, no. 9, pp. 796-810, 2003, doi: 10.1109/TSE.2003.1232285.
40. D. Zowghi and R. Offen, "A logical framework for modeling and reasoning about the evolution of requirements," in Requirements Engineering, 1997., Proceedings of the Third IEEE International Symposium on, 1997: IEEE, pp. 247-257.
41. R. Sugden and M. Strens, "Strategies, tactics and methods for handling change," in Engineering of Computer-Based Systems, 1996. Proceedings., IEEE Symposium and Workshop on, 1996: IEEE, pp. 457-463.
42. M. Strens and R. Sugden, "Change analysis: a step towards meeting the challenge of changing requirements," in Engineering of Computer-Based Systems, 1996. Proceedings., IEEE Symposium and Workshop on, 1996: IEEE, pp. 278-283.
43. O. C. Gotel and A. C. Finkelstein, "An analysis of the requirements traceability problem," in Requirements Engineering, 1994., Proceedings of the First International Conference on, 1994: IEEE, pp. 94-101.
44. R. Torkar, T. Gorschek, R. Feldt, M. Svahnberg, U. A. Raja, and K. Kamran, "Requirements traceability: a systematic review and industry case study," International Journal of Software Engineering and Knowledge Engineering, vol. 22, no. 03, pp. 385-433, 2012.
45. J. Cleland-Huang, R. Settimi, C. Duan, and X. Zou, "Utilizing supporting evidence to improve dynamic requirements traceability," in Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on, 2005: IEEE, pp. 135-144.
46. M. Heindl and S. Biffl, "A case study on value-based requirements tracing," in Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering, 2005: ACM, pp. 60-69.

Shalinka Jayatilleke holds a BSc (Hons) from Institute of Technological Studies (Affiliated to Troy University, USA), a MSc from Sri Lanka Institute of Information Technology and a PhD (in computer science) from La Trobe University, Australia. She is currently a lecturer at La Trobe University with an academic career, which started in 2004. Her current research interests are requirements engineering, change management, digital disruption and learning analytics.

Richard Lai holds a BE (Hons) and a MEngSc from the University of New South Wales and a PhD from La Trobe University, Australia. He has spent about 10 years in

the computer industry prior to joining La Trobe University in1989. His current research interests include component-based software system, software measurement, requirements engineering, and global software development.

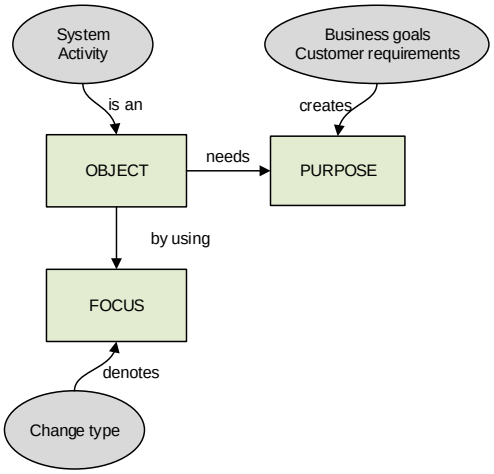
Received: February 21, 2020; Accepted: August 28, 2020

Appendix 1

The change specification method (Refer reference no. 24):

The specification method is made up of GQM and RDF. The GQM-RDF combination is a result of amalgamating ontology and terminology which in this paper, we refer to as onto-terminology. The method has both linguistic and logical principles. To ensure the correct combination of logic and terminology, we have selected two well-known methods where GQM represents terminology and the other RDF ontology. Three terms are extracted from GQM that can best describe a requirement change; Object, Purpose and Focus (of change). The terms extracted from RDF are Object, Attribute and Value, which is referred to as the RDF triplet. The logical relationship of the RDF triplet can be stated as Object O has an Attribute A with a Value V (Professor; Reads; a Book). The rationale behind the correspondence between RDF triplet and to the GQM terms is due to the similarity and the meanings of the terms, which is described in table below.

RDF term	GQM term	Correspondence	Rationale
Object	*Object*	One-to-one	Same concept
Attribute	*Purpose*	One-to-one	Both terms are activities. *Purpose* is an activity that is generated due to various business requirements.
Value	*Focus*	One-to-one	*Value* of RDF creates the significance for *Attribute* (of RDF). *Focus* of GQM creates the significance for *Object* (of GQM) by activating the term *Purpose* of GQM.



Onto-terminology Framework

The template designed for the change specification based on the framework above is given in the table below. By selecting the object of change using the system design diagram, designers and decision makers can accurately locate the main target of change, resulting in a clarification of the location of change. Knowing the reason for the change through the purpose ensures that change implementers are able to clarify the need for the change. The focus of change acts as advice on the basic implementation needed to

execute the change, resulting in the clarification of the action of change. It indicates to the designers what to do instead of how to do the change. We believe that clearly describing the location, need and action of a change request using this template will resolve much of the existing miscommunication issues.

	Description
OBJECT	The activity name according to the system design diagram
PURPOSE	The reason for the change (can be descriptive)
FOCUS	Select from Add, Delete, Modify or Activity Relocation

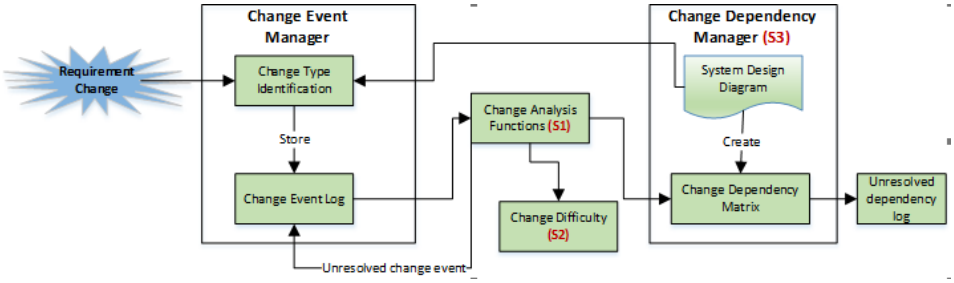
The change classification method (Refer reference no. 24):

The main purpose of change classification method is to ensure that change implementers are able to identify and understand unambiguously the requirement change. The classification is based on previous literature on the same and unstructured interviews of 15 practitioners in the field of change management. The result of this investigation is given in section 4.1 Table 1.

Appendix 2

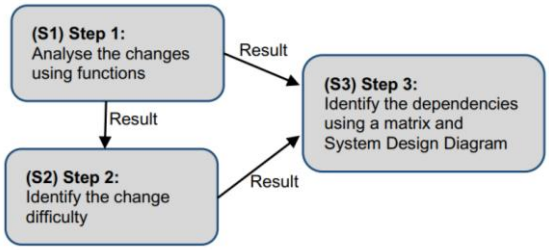
The Method of Requirements Change Analysis (Refer reference no. 25)

The method consists of three steps: namely, (1) analyzing the change using functions, (2) identifying the change difficulty; and (3) identifying the dependencies using a matrix. We have used step 1 in the rework method introduced in this paper.



Change analysis method

Once a change has been identified through the Change Event Manager (CEM), the method follows three steps:



Three step analysis process

- Step 1 (S1) is for expanding the identified changes and for discovering the more detailed information for the implementation as a result of the changes. As shown in Figure 2, the two categories of change analysis functions (herein after referred to as functions) described in section 3.2 are employed for carrying out this step.
- Step 2 (S2) identifies the difficulty of implementing the change. The result of this will be used later for assigning a priority to each of the requested changes.
- Step 3 (S3) identifies the conflicts and/or dependencies between the required changes. As shown in in Figure 2, the key elements involved are the Change Dependency Matrix (CDM) and the System Design Diagram (SDD). The conflicts and/or dependencies between the changes are identified once the changes have been mapped to the matrix.