7659, England, UK
Journal of Information and Computing Science
Vol. 2, No. 1, 2007, *pp*. 48-54

An Algorithm for Evaluating Impact of Requirement Change

He-Biao Yang [+], Zhi-Hong Liu, Zheng-Hua Ma

School of Computer Science and Telecommunication Engineering, Jiangsu University, Zhenjiang, 212013, China

(*Received June 1, 2006, Accepted August 2, 2006*)

Abstract. In software development, it is difficult to evaluate the impact brought by the requirement changes because of the complexities and dependencies among the changed and unchanged requirements. In this paper, the relations of these dependencies are analyzed and established. The scope of the impact caused by requirement changes is identified by a back-tracing algorithm, and impact was quantified. An algorithm developed to quantitatively evaluate the effect of requirement changes is presented. Lastly, the feasibility of this evaluation algorithm is shown through a case-study.

Key Word: Requirement Change, Dependency, Evaluation Algorithm

1. Introduction

With the rapid development and broad applications of computer technology, both the scale and complexity of the software project increase at an unprecedented rate. It is well known that the requirement changes have an enormous influence on all aspects of software projects including progress control, cost analysis, life cycle and etc. The Chaos Report, released by Standing Group showed that among many factors, which cause the failure of the software project, the requirements change account for 11.8%[1]. IBM's Santa Teresa Laboratory reported that for a typical project, on an average, about 25% of the requirements for a typical project would go through changes before completion of the project [2]. It is inevitable that the requirements change occurs in the life cycle of software development [3]. Therefore, the evaluation of the impact attributable to requirements change has become an important part of requirements change control, and it can result in a positive impact on software project management and control risk reduction. The requirements changes, discussed in this paper, are modifications to existing requirements or new requirements that may or may not affect existing requirements.

The evaluation of requirements change impact includes determination of the scope of requirements change and analysis of the potential influence brought by it[4]. The major methods used to evaluate the impact of requirement change is to determine the impact caused by requirements change on software work products, such as design and coding, by using requirements traceability link or traceability matrix[5,6,7]. While complicated dependency exists objectively among the requirements, the impact brought by dependency factor is rarely considered in the existing evaluating methods. As a result, it introduces uncertainty to evaluation results. Therefore, how to identify and define the scope of dependencies become very important.

To solve the aforementioned problems, in this paper, the relations of these dependencies are analyzed and established. The scope of the impact caused by requirement changes is identified by a back-tracing algorithm, and impact was quantified. An algorithm developed to quantitatively evaluate the effect of requirement changes is presented. Lastly, the feasibility of this evaluation algorithm is shown through a case-study.

2. Requirements dependency

Due to the dependencies between requirements, the change of some requirements would have an impact on correlated requirements, which leads easily to the diffusion of impact. This phenomenon makes the evaluation of impact both uncertain and difficult, therefore, the key to resolving this issue is to correctly determine the scope of impact of changing requirements.

[+] *Email address*: yhbjj@ujs.edu.cn (He-Biao Yang), broad-axe@263.net (Zhi-Hong Liu), Zheng-Hua Ma (ma031124@126.com)

2.1. Definition of requirements dependency

Dependencies are common among the requirements of a real software system [8,9,10].

Definition 1. A requirement dependency is a relationship that signifies that the change of a single or a set of requirement elements requires the change(s) of other requirement elements for their specification or implementation. This means that the complete semantics of the depending elements is either semantically or structurally dependent on the definition of the supplier element(s)[11].

That a requirement R_t depends on the other requirement R_s. is denoted as $dep(R_s, R_t)$, and the requirement R_t does not depend on the requirement R_s. is denoted as $\neg dep(R_s, R_t)$, where R_s is the source of requirements dependency, and R_t is the target of requirements dependency.

The requirements dependency is practically regarded as a description of relations among requirements at the coupling aspect, which can be divided into implicit dependency and explicit dependencies. The explicit dependency is usually identified at the phase of modeling requirements, on contrast, the recognization of the implicit dependency is often postponed to later phases, such as design or coding, it can be confirmed with the traceability of requirements.

2.2. Type and property of requirements dependencies

The forms of dependencies vary, and their relations are comparatively complicated. By the analysis and induction on them, we categorize the dependencies into Sub Dependency, Full Dependency, Super Dependency and Loop dependency:

Definition 2. The Sub Dependency means that R_b is partially dependent on R_a, symbolized as $subDep(R_a, R_b)$, $\forall R_a \forall R_b \left(dep(R_a, R_b) \right)$, $\exists R_c \left((R_c \subseteq R_a) \wedge \neg dep(R_a - R_c, R_b) \right)$.

Definition 3. The Full Dependency means that R_b is fully dependent on R_a, symbolized as fulldep (R_a, R_b), $\forall R_a \forall R_b \left(dep(R_a, R_b) \wedge \neg subDep(R_a, R_b) \right)$.

Definition 4. The Super Dependency means that partial R_b is fully dependent on R_a, symbolized as $superDep(R_a, R_b)$, $\forall R_a \forall R_b \exists R_c \left(dep(R_a, R_b) \wedge (R_c \subseteq R_b) \wedge \neg dep(R_a, R_b - R_c) \right)$.

In the process of software development, it is widely acknowledged that the Sub Dependency and Super Dependency are derived from the large granularity of modeling requirements. The non-atomic requirements model may possibly cover partial exact dependencies, which increases the inaccuracy of estimating impact caused by requirements change to some extent. Subsequently, the dependencies of Sub Dependency and Super Dependency should be eliminated as soon as possible in the real software development. To simplify the discussion, the dependencies mentioned below are all Full Dependency.

Two lemmas of Dependencies:

Lemma 1: Transitivity

$$\forall R_a \forall R_b \forall R_c \left(fullDep(R_a, R_b) \wedge fullDep(R_b, R_c) \right) \Rightarrow fullDep(R_a, R_c).$$

If the requirement R_b fully depends on requirement R_a, and the R_c fully depends on R_b also, we can obtain a conclusion that the R_c fully depends on R_a.

Lemma 2: Inreversibility

$$\forall R_a \forall R_b Dep(R_a, R_b) \neq \forall R_a \forall R_b Dep(R_b, R_a)$$

The dependency $dep(R_a, R_b)$ and the dependency $dep(R_b, R_a)$ are two different dependencies.

Definition 5. The Loop Dependency means that the relation among R_1,$R_{2...}R_i$ is loop, denoted as: loopDep(R_1,$R_{2,...,}R_i$),

$$\forall R_1 \forall R_2 \forall R_n \left(fullDep(R_1, R_2) \wedge fullDep(R_1, R_2) \wedge ... \wedge fullDep(R_{i-1}, R_i) \wedge fullDep(R_i, R_1) \right)$$

where $i \geq 2$.

The loop dependency is universally considered as a kind of strong coupling relation structurally, which

lowers the adaptability of model to changes and the reusability of model. Hence, loop dependency should be eliminated as soon as possible in the development of system. The evaluation algorithm presented in this paper detects existing loop dependencies, to remind analyzer and designer of system to improve their work products.

3. Identifying dependency set

This paper uses the backtracing algorithm to solve the dependencies among requirements.

3.1. The composing of solution space

The solution space of the question is presented as directed graph in the algorithm. For the directed graph $G=(N_G,E_G)$, where $N_G=\{Ri|i=1...n\}$, Ri is requirement I labeled as Vertex I, $E_G \subseteq N_G \times N_G$. For any arc $<u,v>\in E_G$, there is a $dep(u,v)$, where v is called tail and u is head. The InDegree of a vertex v is the number of arcs directed towards the vertex v. The OutDegree of a vertex u is the number of arcs directed outwards from the vertex u. A directed graph can be represented by an adjacency matrix, which is defined as:

$$A[i,j]=\begin{cases}1, <R_i,R_j>\in E_G\\ 0, <R_i,R_j>\in E_G\end{cases}$$

3.2. The structure of solution space

The structure of solution space can be represented as a full n-tree which height is $n+1$. i layer $(i\le n)$ of the solution space, every node has n children, and every child represents one of possible dependencies in N_G. The No. n+1 level is the leaf node. Figure 1 shows a spanning tree of the solution space structure, which has 3 elements in its dependencies set.

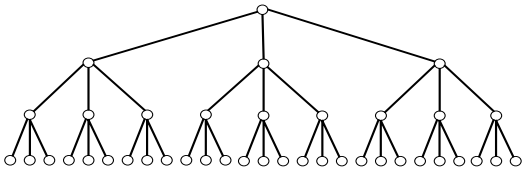

Figure 1. A spanning tree of the solution space

When $n>1$, all nodes are requirement nodes except root node.

3.3. Description of algorithm

This algorithm adopts depth-priority search method, which traverses solution space to find the result(s). In the recursive algorithm Backtrack, it traverses to leaf nodes and outputs a set of dependencies when $i>n$. In case that the initial node is i_0, where $i\le n$, the current extending node (assumption for Z), where is in solution space, which has n child nodes, check its availability here and traverse those valid sub trees recursively using the depth-first search method or trim the invalid ones.

It is indispensable to detect and record loop-dependencies in this process. To achieve this, we need to check whether the arc connecting vertex i-1 to i, and the arc connecting the vertex i to i_0 exists. The existence of all of these arcs means a Loop Dependency is detected.

Before the beginning of the Algorithm backtrack, delete these nodes whose out-degree and in-degree are both zero in the graph G. As a result we can get a set of vertices and arcs, which is stored in an array x[].

```
Backtrack (i) {
if (i>n) {
    //Output dependencies set and dependencies path
}
else {
    if(A[x[i-1],x[i]]=1&&A[x[i],x[i-1]]=1){
            //Output dependencies path
```

```
        }
    }
    if (i>=3)
        if(A[x[i-1],x[i]]=1&&A[x[i],x[1]]=1){
            //Output loop-dependency ;
        }
    else {
        for(j:=1;j<=n;j++) {
            x[i]:=j;
          if(A[x[i-1],x[j]]=1)  //select next search vertex;
              backtrack(i+1);
          else
                    //Output dependencies set and dependencies path
          }
    }
    }//End Backtrack
```

4. An Algorithm of evaluating the Impact of Requirements Change

4.1. Impact factor

The whole scope of impact affected by requirements change is obtained using the Backtrack algorithm above. To measure the degree of impact between requirements, this paper introduces the impact factor (abbreviated to FI), and quantifies the impact factor between two requirements using the following formula.

$$FI\left(Rs,Rt\right)=workload\left|Rt/workload\right|Rs,dep\left(Rs,Rt\right)\wedge change\left(Rs\right) \quad (1)$$

When a requirement Rs is changed, the workload|Rs denotes the added workload of Rs. As the changes of Rs influences requirement R_t, workload|Rt is the added workload of Rt. At the different stage of system development, the FI (Rs,Rt) is likely to be different.

In the course of an application system development, the impact factor is not always the simple ratio as above, and it is likely the exponential or discrete values. To discuss conveniently, we measure the impact factor between requirements using the ratio of workload.

In the process of quantifying FI(Rs,Rt), a lot of reasons such as the capability of software developer, rationality of project schedule, complexity of function structure, etc are possible to affect the accuracy of FI(Rs,Rt).Therefore according to actual situation, the FI(Rs,Rt) is able to be adjusted on the basis of analysis and statistic of empirical or previously recorded data.

4.2. Measure of impact degree

According to the method of quantifying the impact factor, the quantitative formula, which measures the degree of impact caused by requirements change, is as follows:

$$DI=\sum Ws*FI\left(Rs,Rt\right) \quad (2)$$

where Ws is added workload of Rs.

Figure 2 shows the dependencies between requirements

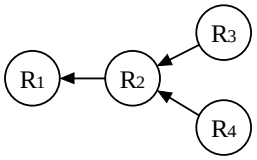

Figure 2. The dependencies between requirements

Assume $FI(R_1, R_2) = 0.5$, $FI(R_2, R_3) = 1.4$, $FI(R_2, R_4) = 0.8$.

If W_1 equals 2. By the formula (2), the results is as follows:

$$W_2 = W_1 \times FI(R_1, R_2) = 4 \times 0.5 = 2;$$

$$W_3 = W_2 \times FI(R_2, R_3) = 2 \times 1.4 = 2.8;$$

$$W_4 = W_2 \times FI(R_2, R_4) = 2 \times 0.8 = 1.6;$$

The total increased workload $DI = 2.8 + 1.6 + 2 = 6.4$.

4.3. Implementation of evaluating algorithm

We can obtain the whole set of impact at the initial state of dependencies. This set is expressed as C_a. While some requirements changes take place, we need to evaluate the propagation of impact and impact on software development.

The detailed approaches are as follows:

1) To extract the impact set of changed requirements required to evaluate from C_a.

2) To eliminate redundant dependencies of change impact set using the cascade method, which is to get minimal subset of dependencies by eliminating repetitive nodes and branch of path.

3) To quantify impact factor (FI) using the formula (1).

4) To evaluate the impact of changed requirements use the formula (2).

The design of evaluation algorithm is as follows:

Store the impact set of changing requirements with an adjacency matrix a, at the beginning $a[i, j] = 0; i, j = 1, 2, ..., n$;

The source of requirements change is represented as $SETcs$, $SETcs = \{Ri \mid Ri \subseteq N\}$.

The impact set of requirements Rs change is represented as $ImpactSet(Rs) = \{Ri \mid Ri \subseteq C_a\}$.

The workload of requirements stored in the InitSet set. $IniSet = \{< Ri, Wi > \mid Ri \in SETcs\}$, where Wi is workload.

The workload of requirements stored in the EvaSet set after the occurrence of requirements change, $EvaSet = \{< Ri, Wi > \mid Ri \in \text{Im } pactSet\}$,where Wi is workload.

Algorithm 1 EliminateRedundance: to eliminate redundant dependencies of change impact set and quantify them.

```
EliminateRedundance (a,ImpactSet) {
for (each < Rs, Rt > ∈ ImpactSet ) {
a[Rs, Rt] = FI(Rs, Rt);
    }
}
```

Algorithm 2 Evaluation of Impact

```
Evaluation()
{ for ( each Ri ∈ SETcs )
{ ImpactSet = C_a(Ri);   // obtain the impact set of Ri change
   EliminateRredundance (a,ImpactSet);   // eliminate redundant dependencies
    }
 for ( each Ri ∈ SETcs )
{ get Ri and corresponding Wi;
```

$Rs = Ri$; $Wsi = Wi$;

$EvaSet = Evaset \cup \{< Rs, Wsi >\}$;

for ($\forall Rt \in ImpactSet(Ri) \&\& < Rs, Rt > \in E(ImpactSet(Ri))$){

$WtI = Wsi \times a[Rs, Rt]$;

$EvaSet = Evaset \cup \{< Rt, Wtr >\}$;

$Rs = Rt$; $WsI = WtI$;

}

}

//compare Evaset with IniSet to gain the results of evaluation.

}

5. Case study

In this section, we use an example to illustrate the evaluation algorithm which we have presented in the previous sections. As a demo, we extract a set of requirements and some dependencies among them. The impact factors(FI) are chosen from statistic data in the real development process. A diagram to depict the dependencies is given as follows:

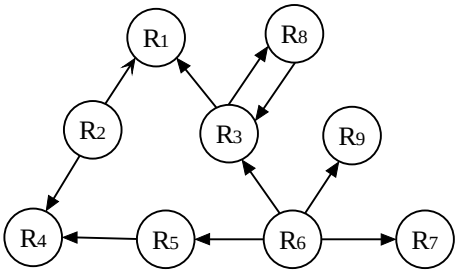

Figure 3. A demo of requirements set

Table 1. Initial workloads of requirements set

Requirement No	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9
Workload	5	4	3	5	7	3	7	8	4

Table 2. The impact factors of requirements set

Rs	R1	R1	R3	R3	R4	R4	R5	R7	R8	R9
Rt	R2	R3	R6	R8	R2	R5	R6	R6	R3	R6
FI(Rs, Rt)	0.80	0.60	1.00	2.67	0.80	1.40	0.43	0.43	0.38	0.75

Figure 4 shows an diagram representing minimal requirements dependencies subset output from our evaluation model, in which loop-dependencies are marked as broken line. The requirements highlighted in dark blue are the source of requirements change.

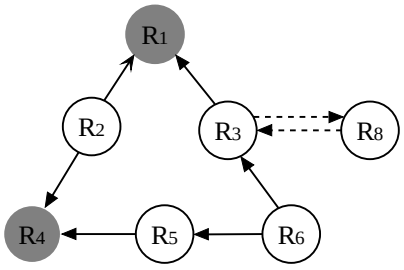

Figure 4. The minimal requirements dependencies subset

Based on the above discussion, a set of predicted data applying evaluation model to compare a set of actual data are listed in table 3.

Table 3. Comparison between predicted workload increment and statistics

Requirement No	R_1	R_2	R_3	R_4	R_5	R_6	R_8
Predicted	4.0	4.8	2.4	2.0	2.8	1.2	6.41
Statistics	3.5	4.0	2.9	2.0	3.3	1.1	7.1

By analyzing the table above, some conclusions are drawn:

1) The minimal requirements dependencies subset are in accordance with the scope of impact caused by requirements change;

2) In spite of the inaccuracy of measure approach and the absence of evaluation parameters, the results derived from 2 different approaches have a similar change trend.

6. Conclusions and Future work

This paper discusses the extent of impact on software system caused by requirements change at the aspect of requirements dependency. An algorithm to evaluate requirements change based on the dependency is presented. The algorithm is validated in a real-life development process. Becaused of the multi-level and multi-phase complexities, the understanding of all dependencies among the requirements and establish a reasonable measurement model aimed at all requirements dependencies is a long term effort; it is iterative and constantly improving process. A series of more complex questions should be studied in the future, such as how to analyze the requirements dependencies qualitatively and quantitatively, to identify the new dependencies introduced by the evolution of requirements and requirements dependencies in the different phases of software development process, to determine a measurement model with more rational and accurate mathematical description.

7. References

[1] The CHAOS Report, 1994, Cont'd. http: http://www.standishgroup.com/sample_research/chaos_1994_2.php.

[2] B. W. Boehm. *Software Engineering Economics*. Prentice-Hall, Englewood Cliffs, New Jersey, 1981.

[3] Saffena Ramzan, Naveed Ikram. Making Decision in Requirement Change Management Information and Communication Technologies. *ICIT First International Conference*. 2005, **27-28**: 309-312.

[4] R. S. Arnold, S. A. Bohner. Impact Analysis-towards a Framework for Comparison. *Proceeding of the International Conference on Software Maintenance.* 1993, 292-301.

[5] S. Lock, Dr. G. Kotonya. Requirement Level Change Management and Impact Analysis. http://info.comp.lancs.ac.uk/publications/Publication_Documents/1998-Lock-Internal.pdf.

[6] M. R. Strens and R. C. Sugden. Change Analysis: A Step Towards Meeting the Challenge of Changing Requirements. *Proceedings of the IEEE Symposium and Workshop on Engieering of Computer Based Systems(ECBS)*, 1996.

[7] L. Wen, R. G. Dromey. From Requirements Change to Design Change: A Formal Path. *Software Engineering and Formal Methods, Proceeding of the Second international Conference*, 2004, 104-113.

[8] P. Carlshamre, K. Sandahl. An Industrial Survey of Requirements Interdependencies in Software Product Release Planning. *Proceedings of Fifth IEEE International Symposium on Requirements Engineering, IEEE Computer Society*, 2001, 84-91.

[9] J. Giesen, A. Volker. Requirements Interdependencies and Stakeholders Perferences. *Proceedings of IEEE Joint International Conference on Requirements Engineering*, 2002, 206-209.

[10] B. Ramesh, M. Jarke. Toward Reference Models for Requirements Traceability, *IEEE Transactions on Software Engineering*, 2001, **27**(1): 58-93.

[11] The specification of UML 2.0. http://www.uml.org