Rule-Based Change Impact Analysis Method in Software Development

Yiheng Wang[a], Jun Zhang[b] and Yujing Fu[c]
School of Information Science and Technology, Dalian Maritime University, Dalian 116026, China;
[a]yiheng_wang@sina.cn, [b]zhangjun@dlmu.edu.cn, [c]kk824437084@163.com

Keywords: Software development, change impact analysis, entity dependency graph, change propagation rules.

Abstract. There are many stages in software development in which there are many changes. A rule-based change impact analysis method for software lifecycle objects was designed from the perspective of the whole software development process. The main ideas are as follows. Firstly, five types of entity dependency and change were introduced and the corresponding change propagation rules were designed. Secondly, a change impact analysis algorithm based on change propagation rules was proposed. Finally, the relevant simulation experiments were carried out to show the effectiveness of the algorithm.

1. Introduction

The software development process includes requirement, design, coding and testing stages, and software changes may occur in all stages of software development. When an entity changes, relevant entities will probably been affected directly or indirectly [1-2]. An excellent and comprehensive software change impact analysis method can greatly reduce the cost of software development, so software change impact analysis has been studied in decades and a lot of analysis methods were proposed [3]. But at present, many methods are limited to a certain stage in the software development, such as requirements change impact analysis [4-5], designs change impact analysis [6], source change impact analysis [7], there is also a lack of change impact analysis method that covers the whole software development process.

After many main change impact analysis methods were investigated, we propose a change analysis method based on the rules of dependency between software development process entities, in which software development process entities' temporal properties were considered. The method is applicable to all stages in the software development process. The software development process entities are also called the software lifecycle object [2].

2. Fundamental

2.1 Change Impact Analysis Sets

The related sets required to process the impact analysis are shown in Table 1 [2, 8-9].

Table 1. Sets for Impact analysis

Abbreviation	Name	Explanation	Purpose
SS	System Set	The set of all entities in the system and all the other sets are subsets of this set.	Evaluation
SIS	Starting Impact Set	Entities that are impacted directly	Input
EIS	Estimated Impact Set	Entities that are impacted possibly	Output
AIS	Actual Impact Set	Entities that are impacted by a change	Evaluation

The SIS typically serves as input to impact analysis approaches that are used for finding the Estimated Impact Set. The EIS always includes the SIS and can therefore be seen as an expansion of the SIS. The expanding part is the collection of entities that are indirectly affected. Ideally, the SIS

and EIS should be the same, meaning that the impact is restricted to what was initially thought to be changed.

The system set and AIS are used for evaluation. All the other sets are subsets of system set. In the best-case scenario, the AIS and EIS are the same, meaning that the impact estimation was perfect.

2.2 Entity Dependency and Change

There are a variety of dependencies between software entities. The change analysis method proposed in this paper is based on the analysis of these dependencies. Therefore, we give the following five types of entity dependencies and their properties, see Table 2.

Table 2. Properties of Entity Dependency

Entity Dependency Type	Definition	Properties			Scope	Objects	Expression	Evaluation
		Reflexivity	Symmetric	Transitive				
Use Relationship	An entity E_1 uses an entity E_2 if E_1 is fulfilled only when E_2 is fulfilled.	×	×	√	entire software system	different entities	used-entity → use-entity	The most basic relationship.
Refine Relationship	An entity E_1 refines an entity E_2 if E_1 is derived from E_2 by adding more details to its properties.	×	×	√	entire software system	similar entities	basic-entity → refine-entity	Similar to the object-oriented Programming language on the inheritance relationship.
Evolutionary Relationship	An entity E_1 evolutes from an entity E_2 if E_1 is derived from E_2 by adding more details to its properties and E_1 and E_2 are different life cycles.	×	×	√	entire software system	the same but different life cycle entities	old-entity → new-entity	Temporality.
Contain Relationship	An entity E_1 contains entities E_2 ... E_n if E_2 ... E_n are parts of the whole E_1.	×	×	√	internal entity	similar entities	parent-entity → child-entities	Part-whole hierarchy.
Conflict Relationship	An entity E_1 conflict with an entity E_2 if the fulfillment of E_1 excludes the fulfillment of E_2 and vice versa.	×	√	×	entire software system	different entities	Conflict-entity ↔ Conflict-entity	Mutually exclusive.

On the basis of entity dependency, the change types of entity should be distinguished when the rule based change impact analysis is to be done [4]. Change types of entities includes add entities, delete entities, and update entities. And update entities includes add property to entity, delete property of entity, and modify property of entity.

2.3 Change Propagation.

Before discussing the rules of change propagation, we explain the concept of two dependency states, namely, persistent relationship state and immediate relationship state. The persistence relationship state refers to the fact that the relationship is used throughout the life cycle of the entity. The use, refine, contain, and conflict relationship are persistent. The immediate relationship state means that the relationship only constraint in original to build a relationship for two entities, and there is no constraint after the relationship is established. The evolutionary relationship belongs to the immediate relationship.

Combining dependency types and entities change, we can obtain the change propagation conditions on the entities dependency graph as follows:

a) add or delete entities

- When entities do add or delete changes, since the use, refine, evolutionary, and contain relationship are non-reflexive, non-symmetric, and transitive, it is known from the property that changes are made to an entity that only affects the subsequent vertex, and the degree of output is not affected.
- For the conflict relationship, by its property, to make a change to a party, will affect the other party.

b) update entities

When entities do update changes, there are different situations for different entity types, as follows:

- Use relationship is persistent relationship status, when implemented changes between two entities with in use relationship, divided into two cases: if the modified property is used by

use-entity, the use-entity will be affected; if the modified property is not used by use-entity, the use-entity is no impact. The use relationship is non-symmetric, therefore, the change of the use-entity does not affect the used-entity.

- The refine relationship also is persistent relationship status, when implemented changes between two entities with refine relationship, change the basic-entity is also divided into two cases: if the modified property is the part of refinement, the refine-entities will be affected; if the modified property is not the part of refinement, the refine-entities will be no impact. The refine relationship is non-symmetric, therefore, the change of the refine-entities does not affect the basic-entity.
- Evolutionary relationship is an immediate relationship, that is, after the new entity is generated, it is not bound by the parent entity, because the evolutionary relationship is the expression of the same entity in different tenses when considering the tense dimension, and when the time dimension is mapped to the two-dimensional space, the entity of the evolutionary relationship becomes two different entities.
- When a change is made on the parent-entity with contain relationship, if no other relationships exists on each child entity, only those child-entities associated with the change is affected, and making changes on the child-entity will necessarily affect its parent-entity.
- When a change is made on one of the two entities in which the conflicting relationship exists, the party is considered to be implemented in the system, and the other party defaults.

In summary, the different types of changes in the various dependencies under the corresponding situation, see Table 3.

Table 3. Change Propagation Rules

Change Type	A uses B	A refines B	A evolutes from B	A contains B	A conflicts with B
Add A	B no impact	B no impact	B no impact	Add B	B failure
Add B	Add A	Add A	Add A during the life cycle of A	A adds the child-entity B	A failure
Delete A	B no impact	B no impact	B no impact	Delete B	B take effect
Delete B	A failure	A failure	A no impact	A lost the child-entity B	A take effect
Add property p for A	B no impact	B no impact	B no impact	① if p is added in B, then B adds property p ② if p is not added in B, then B no impact	The property of B which is conflicts with p is failure
Add property p for B	① if p is used by A, then A impact ② if p is not used by A, then A no impact	① if p is refined by A, then A impact ② if p is refined by A, then A no impact	A no impact	A add property p	The property of A which is conflicts with p is failure
Delete property p of A	B no impact	B no impact	B no impact	① if p belongs to B, then B deletes property p ② if p not belongs to B, then B no impact	The property of B which is conflicts with p take effect
Delete property p of B	① if p is used by A, then A impact ② if p is not used by A, then A no impact	① if p is refined by A, then A impact ② if p is refined by A, then A no impact	A no impact	A lost property p	The property of A which is conflicts with p take effect
Update property p of A	B no impact	B no impact	B no impact	① if p belongs to B, then B updates property p ② if p not belongs to B, then B no impact	The property of B which is conflicts with p possible take effect
Update property p of B	① if p is used by A, then A impact ② if p is not used by A, then A no impact	① if p is refined by A, then A impact ② if p is refined by A, then A no impact	A no impact	A update property p p	The property of A which is conflicts with p possible take effect

Let × represent the condition of no impact, √ represent the condition of impact, ○ represent the condition of possible impact, then the Table 3 can be simplified into Table 4.

Table 4. Simplify Change Propagation Rules

Change Type	A uses B	A refines B	A evolutes from B	A contains B	A conflicts with B
Add A	B×	B×	B×	B√	B√
Add B	A√	A√	A○	A√	A√
Delete A	B×	B×	B×	B√	B√
Delete B	A√	A√	A×	A√	A√
Add property p for A	B×	B×	B×	B○	B.p√
Add property p for B	A○	A○	A×	A√	A.p√
Delete property p of A	B×	B×	B×	B○	B.p√
Delete property p of B	A○	A○	A×	A√	A.p√
Update property p of A	B×	B×	B×	B○	B.p○
Update property p of B	A○	A○	A×	A√	A.p○

3. Change Impact Analysis

3.1 Strategy

The analysis in this paper is based on the entity dependency graph, so the definition is given first:

Definition of Entity Dependency Graph: use directed graph EDG = <V, E, ER> to denote entities dependency graph. In the EDG, V denotes vertex and $V = V_R \cup V_D \cup V_M \cup V_C$, E denotes edge and $E = V \times V \times ER$, ER denotes entities dependency and ER= {USE, REF, EVO, COT, COF}. V_R denotes the set of requirement, V_D denotes the set of design, V_M denotes the set of model, and V_C denotes the set of component. USE denotes use relationship, REF denotes refine relationship, EVO denotes evolution relationship, COT denotes contain relationship, COF denotes conflict relationship.

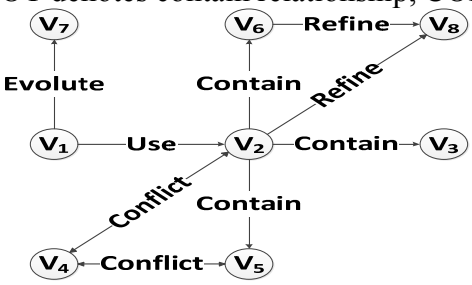

Fig. 1 Entity dependency graph

The figure 1 is an example of entities dependency graph, V_1~V_7 denotes entities and we can see that V_2 uses V_1, V_7 evolutes from V_1, V_2 contains V_3、 V_5 and V_6, V_4 conflicts with V_5, and V_8 refines V_6. V_2 conflicts with V_4 and refined by V_8 because V_5 and V_6 are the child-entities of V_2.

As stated in the previous section, we stand in the perspective of the entities dependency graph, classifying the impact as follows:

a) When the relationship type is ‘use’ or ‘refine’, the change entity which is in-degree is not impact the entity which is out-degree, and the change entity which is out-degree will certainly affect the entities which are in-degree. For example, when A uses or refines B (i.e. B →A), there is no effect on B no matter what changes do on A, and when any changes are made on B, it will necessarily affect A.

b) When the relationship type is ‘evolutionary’, there will impact the entity which is out-degree only when adding a entity which is in-degree during the life cycle of the in-degree entity, and in addition do not affect. For example, when A evolutes from B (i.e. B→A), only when adding B during the life cycle of A will add A, and except this, change both sides is not impact each other.

c) When the relationship type is ‘contain’, the change entity which is in-degree (i.e. child-entity) will certainly affect the entities which is out-degree (i.e. parent-entity), and the entity which is in-degree (i.e. child-entity) will be impacted only when change the entity which is out-degree (i.e. parent-entity) and this change is action in it.

d) When the relationship type is ‘conflict’, on the one side to do any change will affect the other party, and when do update changes, see whether this property of update as conflict property.

We default all entity dependencies have been confirmed, and stored them in the database in accordance with the rules of EDG. When a change is made, the impact analysis steps are as follows:

Step 1: confirm change entity and change type;

Step 2: start from the change entity, traverse EDG to get its dependency sub map subEDG;

Step 3: add all adjacency entities on the subEDG to its starting impact set SIS;

Step 4: EISC that is the candidate of estimated impact set of change entity is equal to SIS;

Step 5: access the entities in the EISC one by one, get the estimated impact set EIS for the entity, there is the following treatment:

1) If the EISC is an empty set, skip to step 7;
2) If the entity has already visited, skip to step 6; otherwise, remove the entity from the EISC and proceed to the next step;
3) Get all adjacency vertices of the current entity on subEDG, and store the entities that have not been accessed into the EISC;
4) If the change type is ‘add entities’ or ‘delete entities’, then:
   a) If this entity with the relationship edge type of its adjacency entities is ‘use’ or ‘refine’, and the change entity as out-degree, then join those adjacency entity which as in-degree in the EIS;
   b) If this entity with the relationship edge type of its adjacency entities is ‘contain’, then join all adjacency entities of this relationship type in the EIS, and remove those adjacency entities which as out-degree from the EIS;
   c) If this entity with the relationship edge type of its adjacency entities is ‘conflict’, then join all adjacency entities of this relationship type in the EIS, and remove them from the EISC;
   d) If this entity with the relationship edge type of its adjacency entities is ‘evolutionary’, then remove all adjacency entities of this entity from the EISC;
   e) Continue to the next step;
5) If the change type is ‘update entities’, then:
   a) If this entity with the relationship edge type of its adjacency entities is ‘use’ or ‘refine’, then join those in-degree entities which is rely on this property in the EIS;
   b) If this entity with the relationship edge type of its adjacency entities is ‘contain’, then join those adjacency entities which as out-degree and those adjacency entities which the modified property belongs to them in the EIS, and remove those adjacency entities which as out-degree from the EISC;
   c) If this entity with the relationship edge type of its adjacency entities is ‘conflict’, then join those adjacency entities of this relationship type which conflict with this property in the EIS, and remove all adjacency entities of this relationship type from the EISC;
   d) If this entity with the relationship edge type of its adjacency entities is ‘evolution’, then remove all adjacency entities of this entity from the EISC;
   e) Continue to the next step;

Step 6: repeat step 5;

Step 7: return EIS.

3.2 Algorithm

Based on the above, we design a Change Impact Analysis algorithm based on change propagation rules , i.e. RBCIA algorithm. This algorithm first confirms whether the change entity exists. If the change entity exists, the change impact analysis will be performed; otherwise it will return the empty set. When performing change impact analysis, the change propagation function is called in order to obtain the estimated impact set of change entity.

Algorithm 1: Rule-Based Change Impact Analysis (RBCIA)

Input: ce, change entity
ct, entity change type
ERES, the set of entity relationship edge
subEDG, entities dependency graph of change entity
Output: EIS, estimated impact set of change entity

```
1:  Set SIS = getBorderEntities(ce, subEDG);
2:  Set EISC = SIS;
3:  Set EIS = empty-set;
4:  Set ieERES = empty-set;
5:  If (isBelongEDG(ce)) Then {
6:  Set subEDG = traverseEDG(ce);
7:  Set ERES = getAllEdge(ce);
8:  For (each ie ∈ subEDG) {
9:  If ((ie.isvisited()) Or (EISC is empty-set)) Then{ Return EIS; } End If;
10: For (each ie ∈ EISC) {
11: EISC.deleteEntity(ie);
12: SIS = getBorderEntities(ie, subEDG);
13:  ieERES = getAllEdge(ie);
14: If (SIS is empty-set) Then { Return EIS; } End If;
15: For (each ie1 ∈ SIS) {
16: If (!ie1.isvisited()) Then { EISC.addEntity(ie1); } End If;
17: } End For;
18: EIS = EIS + ChangePropagation(ct, ie, ieERES);
19: }End For;
20: }End For;} End If;
21: Return EIS;
```

Where the function ChangePropagation performs change propagation rules, as follows:

Function 1: ChangePropagation (ChangeType ct, Entity ie, Set ieERES): Set

Input: ie, entity
ct, entity change type
ieERES, the set of entity relationship edge
Output: EIS, estimated impact set of change entity

```
1:  If ((ct is 'add entity') Or (ct is 'delete entity')) Then {
2:  For (each ere ∈ ieERES) {
3:  If ((ere is 'USE') Or (ere is 'REF') And (ere.source is e)) Then {EIS.addEntity(ie); } End If;
4:  If ((ere is 'COT') Or (ere is 'COF')) Then { EIS.addEntity(ie); } End If;
5:  If (ere is 'EVO') Then { EISC.deleteEntity(SIS); }End If; } End For; } End If;
6:  If ((ct is 'modify entity')) Then {
7:  Property P = getModifiedProperty(e);
8:  For (each ere ∈ ieERES) {
9:  If ((ere is 'USE') Or (ere is 'REF') And (ere.source is e) And (p ∈ ie)) Then {
10: EIS.addEntity(ie); } End If;
11: If (ere is 'COT') Then {
12: If ((ere.target is e) Or ((ere.source is e) And (p ∈ ie))) Then {EIS.addEntity(ie);} End If;
13: } End If;
14: If ((ere is 'EVO') Then {EISC.deleteEntity(SIS); } End If;
15: If ((ere is 'COF') And (p is conflict ie.property)) Then {EIS.addEntity(ie); } End If;
16: }End For;} End If;
17: Return EIS;
```

4. Experiment

This is a simulation experiment, and the software entities have been stored in the database according to the corresponding rules. The experimental process is illustrated by changing the requirement entity R2 as an example. The partial entity dependency graph of the requirement entity R2 is shown in Figure 2.

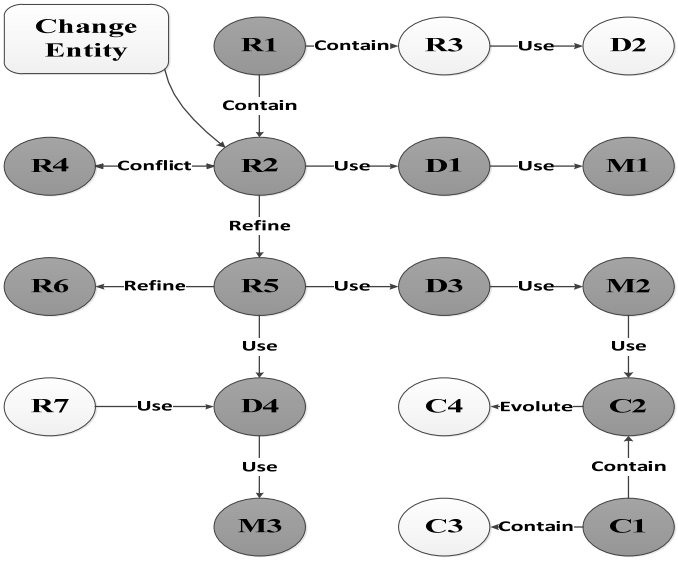

Fig. 2 Experimental Sample

The experimental process is as follows:

1) When a change is introduced, The RBCIA algorithm first searches the database for the entities corresponding to the change , then find the result is the requirement entity R2, that is, to change R2, then starting from R2 and search the connectivity path, and finally get entity dependency graph of R2. It is shown in figure 2.

2) On the entity dependency graph of R2, the algorithm calls the change propagation function to obtain the estimated impact set of the change entity R2.

The implementation of the RBCIA algorithm is shown in Table 5, where zero is the initial state, SIS is the local variable, and all adjacent entity candidates of the currently accessed entity are saved in SIS.

Table 5. ImpactTrace executive condition

Times	Visit entities	SIS of current entity	EISC	EIS
0	R2	R1,R4,R5,D1	R1,R4,R5,D1	∅
1	R1	R2,R3	R4,R5,D1	R1
2	R4	R2	R5,D1	R1,R4
3	R5	R2,R6,D3,D4	D1,R6,D3,D4	R1,R4,R5
4	D1	R2,M1	R6,D3,D4,M1	R1,R4,R5,D1
5	R6	R5	D3,D4,M1	R1,R4,R5,D1,R6
6	D3	R5,M2	D4,M1,M2	R1,R4,R5,D1,R6,D3
7	D4	R5,M3	M1,M2,M3	R1,R4,R5,D1,R6,D3,D4
8	M1	D1	M2,M3	R1,R4,R5,D1,R6,D3,D4,M1
9	M2	D3,C2	M3,C2	R1,R4,R5,D1,R6,D3,D4,M1,M2
10	M3	D4	C2	R1,R4,R5,D1,R6,D3,D4,M1,M2,M3
11	C2	M2,C1	C1	R1,R4,R5,D1,R6,D3,D4,M1,M2,M3,C2
12	C1	C2	∅	R1,R4,R5,D1,R6,D3,D4,M1,M2,M3,C2,C1

5. Related work

There are also many rule-based software change impact analysis methods. For example, Reference [4] presented a technique for change impact analysis in software architecture. It uses the formal semantics of requirements relations, requirements changes and traces between R&A to identify candidate architectural elements for the impact of requirements changes in the architecture. But this architecture based on requirements cannot be applied to other stages of the software development process except for requirements. And it does not take into account the temporality of the entity. Reference [10] presented an approach combining impact analysis with the concept of multi-level modeling, to assist with maintaining software through impact analysis of different UML models, Java source code, and related JUnit tests. However, it is also not considered the temporality of the entity.

Our approach can support the whole software development process, and in which we introduce the temporal property, such as evolutionary relationship.

6. Conclusion

Rule-based change impact analysis method is a prospective method. We proposed a rule-based change impact analysis method to cover the whole software development process. The relevant simulation experiments were carried out to show our method is effective.

In the future, we will continue to work in two ways: (1) weighted edges on the entities dependency graph will be considered to measure the extent to which the entity is affected; (2) entities' temporality will be further studied in software change impact analysis, i.e. static entities dependency directed unweighted graph will be improved to temporal entities dependency directed weighted graph.

References

[1]. S. Lehnert. A review of software change impact analysis. Ilmenau University of Technology, Tech. Rep, 2011
[2]. Jönsson P, Lindvall M. Impact Analysis [M]// Engineering and Managing Software Requirements. 2005:117-142
[3]. SUN Xiao-bing, LI Bin, CHEN Yin et al. A Survey of Software Change Impact Analysis Techniques [J]. Acta Electronica Sinica, 2014, 42(12):2467-2476
[4]. Goknil A, Kurtev I, Berg K V D. A Rule-Based Change Impact Analysis Approach in Software Architecture for Requirements Changes [J]. 2016
[5]. Mokammel F, Coatanea E, Bakhouya M, et al. Impact analysis of graph-based requirements models using PageRank algorithm [J]. 2015
[6]. P H Tang. Design based change impact modeling for object-oriented software [D]. New York :State University of New York, 2003
[7]. Li B, Sun X, Leung H, et al. A survey of code-based change impact analysis techniques [J]. Software Testing Verification & Reliability, 2013, 23(8):613-646
[8]. Steffen Lehnert. A taxonomy for software change impact analysis. In Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th annual ERCIM Workshop on Software Evolution (IWPSE-EVOL 2011), pages 41–50, Szeged, Hungary, September 2011. ACM
[9]. Bohner SA, Arnold RS (1996) Software change impact analysis, IEEE Computer Society Press
[10]. Lehnert S, Farooq Q U A, Riebisch M. Rule-Based Impact Analysis for Heterogeneous Software Artifacts [J]. 2013:209-218