A Representation Method to Simplify Traceability Links between Software Artifacts

Mitsuharu Kitamura, Masanori Takagi, Keizo Yamada and Jun Sasaki
Graduate School of Software and Information Science of Iwate Prefectural University
152-52, Sugo, Takizawa, Iwate, JAPAN
kitamura@naska.co.jp, takagi-m@iwate-pu.ac.jp, k-yamada@iwate-pu.ac.jp, jsasaki@iwate-pu.ac.jp

Abstract—Requirement traceability is essential in verifying compatibility between software requirements and various work products in development phases. However, many organizations cannot employ traceability efficiently because of the large number and complex nature of traceability links. It is necessary to reduce the number and complexity of traceability links for effective verification. We propose a method with which to simplify the traceability links between software requirements and work products in software development processes. We describe the effectiveness of the proposed method in a case study tested of a real software development project.

I. INTRODUCTION

The benefits of requirement traceability have been widely recognized in software development projects for large enterprise systems. There has been much research on traceability, especially in recent years. The main topics of traceability have been the tracking of requirements from their specification through to both their development and maintenance lifecycle (e.g., for verification and validation analysis) [2]. However, these research outcomes have not yet been feasible in industry [3]. There are still various problems to be solved. For instance, many of the proposed traceability modeling concepts are considered too complex for actual use, and the traceability links visualization have been too complex or inadequate for practitioners [3][4]. In particular, the large numbers and complex nature of traceability links make it difficult to understand the overall relationship between artifacts and to break the relationship down into efficient verification units. To verify work products effectively and efficiently, it is important to simplify the traceability links.

We propose a representation method with which to simplify the traceability links between work products in software development processes. In our previous paper [1], we proposed the theoretical basis for the methodology. In the present paper, we describe the effectiveness of the proposed method in a case study of a real application development project.

II. OUTLINE OF THE PROPOSED METHOD

A. Inheritance Relationships

There are many categories of traceability links in software development. To simplify the discussion, we focus on the traceability links between upper works and lower works as shown in Figure 1. We call these inheritance relationships. Inheritance relationships can be represented as a bipartite graph.

![Figure 1 Basic model for inheritance relationships (bipartite graph).](image)

B. Units for Verifying Inheritance Relationships

An upper disjoint set represents a set of work products within a proceeding process, and a lower disjoint set represents a set of work products within a succeeding process. Connected graphs within the target area of inheritance relationships for verification are referred to as units for verifying inheritance relationships, or more concisely, inheritance units.

C. Basic Units for Verifying Inheritance Relationships

According to the definition of the basic unit, an inheritance unit consists of primitive work units are referred to as basic units for verifying inheritance relationships, or more concisely, basic units. Basic units are classified into the three types shown in Figure 2.

![Figure 2 Categories of basic units](image)

a) Injection type: Verification in this case is to check an inheritance relationship to determine whether one work product of a proceeding process is satisfied by one work product of a succeeding process.
b) Decomposition type: Verification in this case is to check inheritance relationships to determine whether one
work product of a proceeding process is satisfied by multiple work products of a succeeding process compatibly and exhaustively.

c) Composition type: Verification in this case is to check inheritance relationships to determine whether multiple work products of a proceeding process are satisfied by one work product of a succeeding process without any omission.

We refer to a link in the basic unit as a basic unit link.

D. Structure of Basic Units within an Inheritance Unit

According to the definitions of basic units, an inheritance unit consists of one or more basic units. For example, Figure 3 illustrates an inheritance unit consisting of a basic unit of composition type and a basic unit of decomposition type. If an inheritance unit consists of multiple basic units, each pair of basic units of different types has a common edge (called the basic unit coupling link). The link from node 1 to node 4 in the figure is the basic unit coupling link.

E. Complexity of an Inheritance Unit

The measurement of complexity of an inheritance unit is to measure the number of basic unit links and basic unit coupling links within an inheritance unit. Figure 4 shows characteristics of the complexity.

F. Simplification Methods

1) Increasing orthogonality: Redundant links are reduced by eliminating overlapping relationships, containment relationships and inheritance relationships among nodes within the same disjoint set. Figure 5 exemplifies this simplification method.

2) Concentrating links: By concentrating links in nodes with multiple links, the nodes can be converted such that they have only one link. The number of basic unit links is thus reduced. Figure 6 exemplifies this simplification method, where links are concentrated such that they are connected to node 1.

3) Specializing nodes: Nodes within a lower disjoint set can specialize in a requirement of a node within an upper disjoint set. Figure 7 exemplifies this simplification method, where upper requirements are broken down and intermediate nodes 5, 6, 7, and 8 are created. As a result, there are fewer coupling links.

4) Abstracting nodes: The process of abstracting nodes is a kind of encapsulation. If there is an upper node connected with many basic unit links, simplification can be achieved by adding combining nodes into new intermediate nodes. Figure 8 exemplifies this simplification method, where new intermediate nodes 6 and 7 are added and M, N and O of inheritance units are created.
III. CASE STUDY

A. Software Development Project

We applied our proposed simplification method to an actual software development project as a typical case. The project has to develop an order and supply management system for suppliers and retailers. The company requesting the system has many requirement items and many function modules in the system design.

1) QFD representation

Before describing the traceability simplification, we introduce the representation method QFD (Quality Function Deployment), which is applied in a general software development. QFD is also called “Quality of House” [5]. QFD represents inheritance relationships in matrices. In the previous section, a graphical representation was used for clarity in simplifying traceability links. From this section, we use the QFD representation instead of the graphical representation to show how real simplification works. Table 1 shows the correspondence between the graphical and QFD representations.

Table 1 Correspondence between graphical and QFD representations

<table>
<thead>
<tr>
<th>Graphical expression</th>
<th>QFD expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertex</td>
<td>Column and row</td>
</tr>
<tr>
<td>Edge</td>
<td>marked cell “○”</td>
</tr>
<tr>
<td>Upper disjoint set</td>
<td>Set of columns</td>
</tr>
<tr>
<td>Lower disjoint set</td>
<td>Set of rows</td>
</tr>
</tbody>
</table>

2) Procedure for calculating complexity

a. Number of basic unit links: $C_p$

1. Identify the inheritance unit (denoted IU)
2. If the sum number of “○” within the IU equals 1,
   $C_p = 1$ (go to step 9).
3. Otherwise, select a column within the IU.
4. If the sum number of “○” in the column exceeds 1,
   Add the sum number to $C_p$.
5. Repeat steps 3 to 4 until select all columns in the IU.
6. Select a row within the IU.
7. If the sum number of “○” in the row exceeds 1,
   Add the sum number to $C_p$.
8. Repeat steps 7 to 8 until select all rows in the IU.
9. The $C_p$ calculation is complete.

b. Number of basic unit coupling links: $C_c$

$$C_c = C_p - \text{the sum number of “○” in the IU.} \quad (1)$$

3) Aggregate indexes for complexity of a zone

A matrix of QFD, called a zone, is an independent work unit for verification. Aggregate indexes for complexity of a zone are defined below. (sigma represents the scalar summation.)

**Average number of basic unit links** =

$$\Sigma C_p / \text{the number of basic units within a zone} \quad (2)$$

**Average number of basic unit coupling links** =

$$\Sigma C_c / \text{the number of inheritance units within a zone} \quad (3)$$

4) Application preparation

In upper phases of software development project, there are usually three zones which represent inheritance relationships as shown in Figure 9.

**Zone I:** Inheritance relationships between requirements in the upper disjoint set and functions in the lower disjoint set

**Zone II:** Inheritance relationships between functions in the upper disjoint set and system Components in the lower disjoint set

**Zone III:** Inheritance relationships between requirements in the upper disjoint set and system Components in the lower disjoint set

In this paper, we focus on only zone I in a case study to demonstrate the effectiveness of the proposed method.

Requirements can be divided into functional requirements and non-functional requirements. Therefore, zone I is divided into zones I-1 and I-2 from the
viewpoints of function and non-function (disconnecting function nodes), as shown in Figure 10.

Figure 9 QFD representation.

Figure 10 Disconnecting zone I according to viewpoint.

Figure 11 shows the correspondence of the simplification procedure between the case study (upper flow) and the proposed methods (under terms).

Figure 11 Simplification procedure used in the case study

B. Results of the Case Study

In this section, we show the application results for the proposed methods. These results are represented in matrices of QFD and the complexity measurements.

1) Refinement of requirements

Refinement of requirements corresponds to increasing orthogonality in the proposed method.

In Figure 12, “u18” exemplifies the initial status. In Figure 13, “u22” and “u23” exemplify application results of the refinement of requirements. By eliminating containment relationships among the requirements, the number of redundant links is reduced. Table 2 shows the results of complexity measurement, where “|E|” is the number of edges within an inheritance unit.

2) Reallocation of function elements

Reallocation of function elements corresponds to concentrating links in the proposed method.

In Figure 14, “u27” exemplifies application results of the reallocation of function elements. A function element that is satisfied with the requirement “r3” is separated from the function “ss-005” and is combined into “ss-014.” Table 3 shows the result for the reallocation of function elements.
### Functional Requirements

**Table 3** Result for the reallocation of function elements

<table>
<thead>
<tr>
<th>Units</th>
<th>E</th>
<th>Cp</th>
<th>Cc</th>
</tr>
</thead>
<tbody>
<tr>
<td>u27</td>
<td>7</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>u23</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

3) Disconnection of functions

Disconnection of functions corresponds to specializing nodes in the proposed method.

In Figure 15, “u31” and “u32” exemplify application results of disconnection of functions by disconnecting the link between the requirement “r3” and the function “ss-014”. Table 4 shows the result for the disconnecting of function.

### IV. DISCUSSION

In the case study for the method proposed in this paper, we could confirm that the number of redundant links is reduced and simplification can be achieved.

As a summary of the case study, Figures 16 and 17 show the transition of complexity of an inheritance unit within zones I-1 and I-2 respectively. We confirm that the complexity measurements decrease in step with each simplification method.
Table 5 Overall measurement results for each zone

<table>
<thead>
<tr>
<th>Zone</th>
<th>I.U. (1)</th>
<th>B.U. (2)</th>
<th>E</th>
<th>ΣCp</th>
<th>ΣCc</th>
<th>ΣCc/E</th>
<th>Ave.Cp</th>
<th>Ave.Cc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial status</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>1</td>
<td>25</td>
<td>63</td>
<td>117</td>
<td>54</td>
<td>0.86</td>
<td>4.7</td>
<td>54.0</td>
</tr>
<tr>
<td>II</td>
<td>14</td>
<td>16</td>
<td>32</td>
<td>34</td>
<td>2</td>
<td>0.06</td>
<td>2.1</td>
<td>0.1</td>
</tr>
<tr>
<td>III</td>
<td>1</td>
<td>43</td>
<td>114</td>
<td>224</td>
<td>110</td>
<td>0.96</td>
<td>5.2</td>
<td>110.0</td>
</tr>
<tr>
<td>Application results</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>4</td>
<td>23</td>
<td>44</td>
<td>78</td>
<td>34</td>
<td>0.77</td>
<td>3.4</td>
<td>8.5</td>
</tr>
<tr>
<td>II</td>
<td>15</td>
<td>17</td>
<td>44</td>
<td>47</td>
<td>3</td>
<td>0.07</td>
<td>2.8</td>
<td>0.2</td>
</tr>
<tr>
<td>III</td>
<td>5</td>
<td>22</td>
<td>47</td>
<td>74</td>
<td>27</td>
<td>0.57</td>
<td>3.4</td>
<td>5.4</td>
</tr>
</tbody>
</table>

(1)I.U.: The number of Inheritance Units within a zone
(2)B.U.: The number of Basic Units within a zone

Figure 18 Transition average numbers of basic unit links and basic unit coupling links in aggregate index in zone I-I.

Table 5 gives the overall measurement results for each zone, which were obtained by applying the same method for the described case study. The table shows that there are many basic unit links (Cp) and basic unit coupling links (Cc) initially in zone I. We see that the simplification methods are effective in decreasing the complexity measurements in zone I and zone III. In contrast, the complexity measurements are not reduced in zone II because of the addition of an intermediate node. Considering the overall condition, we confirm that simplification is achieved by our proposed numerical expression and simplification method.

In this paper, we described simplification methods and a case study for only the inheritance relationship between an upper layer and lower layer, which is referred to as horizontal traceability [6]. In actual software development, there is vertical traceability in the same layer of a development process. For review, inspection and verification works in system development, both forms of traceability should be considered. There was no measurement and simplification method of the complexity on vertical traceability. So, we at first have proposed a method using a graphical model for horizontal traceability, next, we have shown the simplification methods further.

V. CONCLUSION

This paper outlined simplification methodology for traceability between software artifacts using a graphical model. A case study of a real software development project and simplification procedure corresponding to the proposed method was shown the effectiveness of our methodology. We could break down the complex traceability links so that they were simplified for carrying out efficient verification of units. A method of quantitatively measuring complexity has shown that our approach can provide effective evaluation for better traceability. The effect of simplification in the case study was evaluated, and we confirmed that the proposed method is applicable to an actual software development project.

In further study, we will apply the proposed method to other software development projects and verify its reliability and applicability conditions. We will then consider vertical traceability and deficient verification. Simplification works have as associated work load, and thus, the whole process of software development will need to be considered and an optimal approach needs to be investigated from the viewpoint of total cost.

REFERENCES