A SOFTWARE TRACEABILITY MODEL TO SUPPORT CHANGE IMPACT ANALYSIS

Suhaimi Ibrahim, Norbik Bashah Idris
Centre For Advanced Software Engineering, Universiti Teknologi Malaysia, Kuala Lumpur, Malaysia
suhaimi@case.utm.my, norbik@case.utm.my

Malcolm Munro
Department of Computer Science, University of Durham, United Kingdom
malcolm.munro@durham.ac.uk

Aziz Deraman
Fac. of Technology & Info. System, Universiti Kebangsaan Malaysia, Bangi, Selangor, Malaysia
a.d@pkrisc.ukm.my

Abstract

It is inevitable that a software undergoes some change in its lifetime. With some change requests comes a need to estimate the scope (e.g. size and complexity) of the proposed changes and plan for their implementation. Software traceability and its subsequent impact analysis help relate the consequences or ripple-effects of a proposed change across different levels of software models. In this paper, we present a software traceability approach to support change impact analysis of object oriented software. The significant contribution in our traceability approach can be observed in its ability to integrate the high level with the low level software models that include the requirements, test cases, design and code. Our approach allows a software traceability between software components not only within a workproduct but also across different workproducts in SDLC. It supports the top down and bottom up traceability in response to tracing for the potential effects. We developed a software prototype called CATIA to support C++ software and applied it to a case study of an embedded system, and finally we discuss the results.

Keywords: Requirements traceability, impact analysis, static analysis, dynamic analysis.

1. Introduction

It is inevitable that a software undergoes some change in its lifetime. With some change requests comes a need to estimate the scope (e.g. size and complexity) of the proposed changes and plan for their implementation. The main problem to a maintainer is that seemingly small changes can ripple throughout the system to cause substantial impact elsewhere. It seems that determining the effects of change requests has been something that a maintainer traditionally accomplishes it by analyzing the existing dependencies or relationships among the software components of which the job is very tedious, time consuming and error prone.

Software change impact analysis [1], or impact analysis for short, offers considerable leverage in understanding and implementing change in the system because it provides a detailed examination of the consequences of changes in software. Impact analysis provides visibility into the potential effects of the proposed changes before the actual changes are implemented. The ability to identify the change impact or potential effect will greatly help a maintainer or management to determine appropriate actions to take with respect to change decision, schedule plans, cost and resource estimates. Many works of impact analysis are centered around the code level such as [2,3,4].

To manage impact analysis at a broader perspective is considerably hard as it involves traceability within and across different models in the software life-cycle, such as from the design model to code model, etc. Ramesh relates
traceability as the ability to trace the dependent items within a model and the ability to trace the corresponding items in other models [5]. Such kind of traceability is called requirements traceability [5]. Pursuant to this, Turner and Munro [6] assume that a system traceability implies that all models of the software are consistently updated.

Requirements traceability has been mandated as a pivotal element that needs to be included in the software development and maintenance activities by many software engineering standards (e.g. IEEE/EIA 12207). However, not much elaboration on what types of information needed and how a strategy to achieve this are described in the software standards, guidelines and documentation.

Research on requirements traceability has been widely explored since the last two decades that focus up to now on the analysis of dependencies between classes, either at the design or code level [7],[8],[9]. We would like to explore a requirements traceability for change impact analysis from which we should be able to capture the impacts of a proposed change. It should allow a software traceability between software components not only within a workproduct but also across different workproducts in software development life cycle (SDLC). It should support the top down and bottom up traceability in response to tracing for the potential effects.

2. Our Traceability Model

Figure 1 reflects the notion of our model to establish the relationships between artifacts. Four main artifacts that include the requirements, design, test cases and code represent the workproducts of the software development phases which can be extracted from the requirements and specification, design, testing and code documents respectively. The thick arrows represent the direct relationships while the thin arrows represent the indirect relationships. Both direct and indirect relationships can be derived from static or dynamic analysis of component relationships. Direct relationships apply actual values of two components, while indirect relationships apply intermediate values of relationship e.g. using a transitive closure.

Static relationships are software traces between components resulting from a study of static analysis on the source code and other related models. Dynamic analysis on the other hand, results from execution of software to find traces such as executing test cases to find the impacted codes. We classify our model into two categories; vertical and horizontal traceability [10]. The detailed implementation of our model was presented in [11].

2.1 Vertical and Horizontal Traceability

Vertical traceability refers to the association of dependent items within a model and horizontal traceability refers to the association of corresponding items between different models.

The impact analysis of a system can be interpreted as follows.

$$S = (G, E)$$

$$G = GR \cup GD \cup GC \cup GT$$

$$E = ER \cup ED \cup EC \cup ET$$

The system impact, S is a set of inter-artifact relationships. G represents the artifact levels (i.e. requirements, design, code, test cases) and E represents a relationship between artifacts of different levels. Each level of horizontal relationships can be defined from the following perspectives.

Figure 1: Meta-model of traceability system

i) Requirement Traceability

$$ER \subseteq GR \times SGR$$

$$SGR = GD \cup GC \cup GT$$

$$GR = \{R_1, R_2, \ldots, R_n\}$$

ER is defined as a relationship between a requirement and other artifacts of different levels.

ii) Design Traceability

$$ED \subseteq GD \times SGD$$

$$SGD = GR \cup GC \cup GT$$

$$GD = \{D_1, D_2, \ldots, D_n\}$$

ED is defined as a relationship between design and other artifacts of different levels.

iii) Test case Traceability

$$ET \subseteq GT \times SGT$$
E SGT = GR ∪ GD ∪ GC
GT = \{T_1, T_2, \ldots, T_n\}
ET is defined as a relationship between a test case and other artifacts of different levels.
iv) Code Traceability
EC ⊆ GC x SGC
SGC = GR ∪ GD ∪ GT
GC = \{V_1, V_2, \ldots, V_i\}
EC is defined as a relationship between a code component and other artifacts of different levels.
The relationships at code level can be further constructed between variables, methods and classes based on OO dependencies [12].

2.2 Traceability Techniques
Intrinsically, traceability provides a platform for impact analysis. Our approach applies the following techniques.
1. Traceability via explicit links
   Technical means of explicitly defining some program constructs e.g. inter-class relationships in a class diagram modeled using UML.
2. Traceability via name tracing
   Searching items with names similar to the ones in the starting model.
3. Traceability via domain knowledge and concept location
   Tracing for concepts using some predefined knowledge about how different items are interrelated. We developed a tool, called CodeMentor to implement this process of test scenarios between the requirements and implementation code in order to establish a requirement-code relationship. This process applies a reconnaissance technique [13].

3. Methodology
The research is conducted in three stages.
i) First Stage
   We constructed a software traceability model mainly based on the current literatures. The approach incorporates some existing impact analysis techniques into an integrated traceability model and mechanism. To further strengthen our model, we conducted a survey by questionnaires to software professionals in the industries.

ii) Second Stage
   We developed a prototype, called CATIA based on the model defined earlier. The prototype was then tested and implemented to determine the effectiveness and significant use of the approach.

iii) Third Stage
   We applied the prototype to a case study of software project with a complete set of documentation. We performed both dynamic and static analysis on the system (with the help of documentation) to establish a traceability mechanism before the actual impact analysis can take place. We then experimented the case study by allowing some software professionals (i.e. students with working experience) to use the prototype in a controlled laboratory setting.

3.1 Case Study
   We applied our traceability approach to a case study of software project, called the Automobile Board Auto Cruise (OBA). The OBA project is an embedded software system of about 4k LOC with 480 pages of documentation. The project was built with a complete project management and documentation standard adhering to DoD standard, MIL-STD-498 [14]. The software design was built based the UML specification and design standards [15] with the code written in C++.

   We analyzed the code and obtained from it the program dependencies of class and method relationships. We had also spent a substantial time and effort translating each requirement into its implementation code using CodeMentor as mentioned earlier. In our context of study, it is immaterial whether this process is done manually or by semi-auto as our objective here is to establish the requirement-code relationships prior to the implementation of traceability analysis. This process gave us both the SIS and AIS of requirement-method (R-M), requirement-class (R-C) and requirement-package (R-P) relationships. The transformation from R-M to R-C and R-P was handled automatically by our tool.

3.2 Controlled Experiment
   Twenty participants were involved in a one day experiment with the above case study, OBA used as a maintenance project. Participants were the post-graduate students of software engineering at the Centre For Advanced Software Engineering, Universiti Teknologi Malaysia. The subjects were familiar with the above case study as the OBA was used as a common teamwork
assignment they had to complete for their final year project. The subjects represented the software professionals as all of them had at least one year working experience and had some knowledge on software maintenance as this subject was taught in the post graduate programme. The subjects were divided into five groups of fairly distributed expertise and working experience. Each group was given a change request. With a change request, they had to transform it into some understandable software components (PIS-primary impact set) and use the CATIA tool to generate the potential effects (SIS-secondary impact set) in terms of the number of components, its sizes and complexities. Based on the SIS the groups had to draw the actual impacts (AIS-actual impact set). AIS need to be manually verified by exploring and searching through the right paths in the code and other components including design, test cases and requirements. They had to consult documentation, if necessary. Lastly, they had to modify, compile and submit the compiled code together with the data they had obtained. We double checked their work by examining through all the paths and steps they had undertaken to make sure that no careless mistakes occurred.

### Table 1: Impact analysis results

<table>
<thead>
<tr>
<th>Grp</th>
<th>PCR#</th>
<th>Levels</th>
<th>PIS#</th>
<th>SIS#</th>
<th>AIS#</th>
<th>Incl</th>
<th>S-Ratio</th>
<th>Ampl</th>
<th>Ch-Rate</th>
<th>Analy. Effort (PIS)</th>
<th>Spec. Effort (AIS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>Method</td>
<td>2</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>0.40</td>
<td>2.5</td>
<td>0.06</td>
<td>22</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Class</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0.50</td>
<td>1.0</td>
<td>0.09</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Package</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
<td>1.0</td>
<td>0.08</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>T. Case</td>
<td>-</td>
<td>34</td>
<td>16</td>
<td>1</td>
<td>0.47</td>
<td>-</td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reqt.</td>
<td>-</td>
<td>46</td>
<td>19</td>
<td>1</td>
<td>0.41</td>
<td>-</td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>Method</td>
<td>2</td>
<td>13</td>
<td>6</td>
<td>1</td>
<td>0.46</td>
<td>6.5</td>
<td>0.16</td>
<td>27</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Class</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>0.75</td>
<td>2.0</td>
<td>0.17</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Package</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0.67</td>
<td>1.5</td>
<td>0.25</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>T. Case</td>
<td>-</td>
<td>34</td>
<td>13</td>
<td>1</td>
<td>0.38</td>
<td>-</td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reqt.</td>
<td>-</td>
<td>46</td>
<td>17</td>
<td>1</td>
<td>0.37</td>
<td>-</td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>Method</td>
<td>3</td>
<td>10</td>
<td>4</td>
<td>1</td>
<td>0.40</td>
<td>3.33</td>
<td>0.13</td>
<td>30</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Class</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>0.50</td>
<td>1.33</td>
<td>0.17</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Package</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0.67</td>
<td>1.5</td>
<td>0.25</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>T. Case</td>
<td>-</td>
<td>34</td>
<td>16</td>
<td>1</td>
<td>0.47</td>
<td>-</td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reqt.</td>
<td>-</td>
<td>46</td>
<td>20</td>
<td>1</td>
<td>0.43</td>
<td>-</td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td>Method</td>
<td>3</td>
<td>11</td>
<td>3</td>
<td>1</td>
<td>0.27</td>
<td>3.67</td>
<td>0.14</td>
<td>35</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Class</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>0.75</td>
<td>1.33</td>
<td>0.17</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Package</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1.0</td>
<td>1.0</td>
<td>0.17</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>T. Case</td>
<td>-</td>
<td>34</td>
<td>34</td>
<td>1</td>
<td>1.0</td>
<td>-</td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reqt.</td>
<td>-</td>
<td>46</td>
<td>46</td>
<td>1</td>
<td>1.0</td>
<td>-</td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>5</td>
<td>Method</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>0.60</td>
<td>2.5</td>
<td>0.06</td>
<td>25</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Class</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1.00</td>
<td>1.0</td>
<td>0.09</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Package</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1.0</td>
<td>1.0</td>
<td>0.17</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>T. Case</td>
<td>-</td>
<td>34</td>
<td>8</td>
<td>1</td>
<td>0.26</td>
<td>-</td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reqt.</td>
<td>-</td>
<td>46</td>
<td>22</td>
<td>1</td>
<td>0.48</td>
<td>-</td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 3.3 Benchmarking

We evaluated our impact analysis based on a framework as introduced by Arnold et al. [16]. The framework provides an aid to compare impact analysis approaches, assessing the strengths and weaknesses of individual impact analysis methods. In the benchmarking framework, Arnold sets out seven categories of desired trends ranging from the “best” to “not so good” in response to SIS and AIS. Bianchi’s work and assessment do contribute something to our validation, although it is different in objective [8].
4. Results and Discussion

The OBA version was specially selected from the best teamwork project of the year. We identified from the OBA project, 46 requirements, 34 test cases, 12 packages, 23 classes and 80 methods. The Table 1 reveals the results of impact analysis collected from the experiment. As each group was given a change request (PCR), each has to provide results at all levels in terms of PIS#, SIS# and AIS#. Then we computed the values for the rest of the metrics. They were also required to state the time duration in minutes they spent to complete the searching for PIS and AIS. In our study, the dependent variables are the scores and time duration while the independent variables are the tools, subjects and PCRs. The Inclusiveness metric was 1 in each case analyzed, meaning that the AIS was always contained within the SIS. This finding showed that the automated impact analysis was not affected by errors and the further metrics were meaningful. The S-Ratio metric expresses how far the SIS corresponds to the AIS. The data show that adherence gets worse (i.e. the S-Ratio decreases) as the degree of granularity gets finer. In this case we presume that the maintainer will have to devote a substantial maintenance effort to discarding the estimated components that will not actually be modified.

The Amplification metric expresses how much the SIS extends the PIS. Table 1 shows that the SIS increases considerably as the granularity becomes finer. This effect depends on the large number of links represented by this model. Finally the Change Rate indicates how large is the estimated impacts can be detected out of a system. Unlike the others, this metric gets worse (i.e. increases), the coarser grained the model. This effect cannot be avoided because it depends on the loss of detail of the estimate due to the coarser granularity.

We calculated the S-Ratio average of method, class and package levels to be 0.43, 0.7 and 0.87 respectively. Based on the Arnold’s benchmarking, our results fall under the category of:

| Method level: Expected. The EIS# contains the AIS#, and EIS# is not much bigger than the AIS#.
| Class level: good. Estimated impact set matches AIS#. If this happens regularly, usefulness of impact analysis is substantially increased.

Package level: Highly good. Desired |AIS#|=|EIS#| always.

Surprisingly, the results of the metrics at the test case and requirement levels were relatively unpredictable. This is due to the fact that a requirement itself is very subjective. This means we can capture the estimated impacts on code via test case execution but the actual impacted components (AIS) are arbitrary that can only be determined by the user. The above results show that more maintenance effort is required as the degree of granularity get finer to search for the actual impacted components. We also assessed two efficiency metrics of the maintenance process: Analysis Effort, that the subjects expended to select the PIS, and the specification Effort expended to select the AIS from the EIS and specify the changes to the components belonging to the AIS. These efforts were expressed in minutes. The data show that the Analysis Effort required is greater in group C and D. The increased effort could be explained by the increase in PIS# and SIS#. While the group B recorded the highest Specification Effort due to an increase in AIS#. In conclusion, the greater effort devoted to the analysis of a finer grained model is counterbalanced by the greater accuracy of the modification accomplished.

5. Conclusion

More maintenance effort is required as the degree of granularity get finer to search for the actual impacted components. At the coarse granularity, the approach tends to provide efficiency but with inaccuracy as the impacts are too generalized to consider. Therefore, the system administrator should decide whether to establish the efficiency or accuracy of the maintenance task as the main requirement to be satisfied and the granularity of the software level to be chosen accordingly.

At requirement and test case levels, since the AIS table (requirements-code relationships) was made available by the user earlier, our approach can simply apply this table instead of SIS table to observe the actual links involving the requirements with the rest of the component levels. Our assumption here is that there should be a separate mechanism to establish a software traceability between requirements and the implementation code before the actual traceability can be performed.

Our model and approach provide some leverage to the implementation of software engineering
standards, particularly on change impact analysis of software maintenance. Our work differs from others in that we attempt to integrate the software components that include the requirements, test cases, design and code. Another significant achievement can be seen in its ability to support top down and bottom up tracing from a component perspective. This allows a maintainer or management to identify all the potential effects before a decision can be made. Our traceability manages to directly link a component at one level to other components of different levels e.g. from a requirement to code with methods being considered as our smallest artifacts. This allows potential effects to be made more focused.

References


