Recovering And Using Use Case Diagram To Source Code Traceability links

Mark Grechanik, Kathryn S. McKinley, Dewayne E. Perry

ESEC/FSE ‘07

2008. 5. 14

Park, Seung-Hun

© KAIST SE LAB 2008
Contents

- Introduction
- LeanArt approach
- Navigating traceability links
- Experimental evaluation
- Related work
- Conclusion
- Discussion
Introduction (1/2)

- Traceability links between use case diagrams (UCDs) and source code
  - Help programmers to find initial points and understand the meaning of program entities when maintaining and evolving software
  - Establishing the links is rarely done because it is a manual, tedious, and laborious process

- Previous researches for recovering the links
  - Depend on exact matches between the names of elements of UCDs and the names of program entities
In this paper

- Propose a lightweight approach for recovering traceability links, LearnArt (LEarning and ANAlyzing Requirements Traceability)
  - Combines program analysis, run-time value of program entities, and machine learning
  - Automatically propagates a small set of initial traceability links to additional unlinked program entities thereby recovering new links
Overview

- Relations between use case elements
- Relations between program entities

LeanArt approach (1/4)

- Initial traceability links
- Executable program
- Names and the values of program entities
LeanArt approach (2/4)

- **Considered relations**
  - $\alpha$ relation: $(t, c) \in \alpha$
    - $\alpha$ is the traceability relation
    - $t$ is a program entity
    - $c$ is an element of some UCD
    - Obtained by programmers when defining initial links or by machine learning
  - $\gamma$ relation: $(c_p, c_q) \in \gamma$
    - $\gamma$ is the relations between elements in UCDs
    - $c_p$ and $c_q$ are elements of some UCDs
    - Extracted from UCDs
  - $\delta$ relation
    - Relations between types and types, types and variables, and variables and variables
    - Types-types are obtained using type checking algorithm
    - Variables-variables are obtained by performing control and data flow analysis
LeanArt approach (3/4)

- **Validation algorithm**
  - Guesses traceability links for untraced program entities using existing traces and $\delta$ and $\gamma$ relations
  - Validates traces determined by the machine learning techniques
  - Suggests three types of composition rules
    - $\sigma = \delta \circ \alpha : (t_m, t_n) \circ (t_n, c_p) \Rightarrow (t_m, c_p)$
    - $\sigma = \alpha \circ \gamma : (t_n, c_p) \circ (c_p, c_q) \Rightarrow (t_n, c_q)$
    - $\sigma = \delta \circ \alpha \circ \gamma : (t_m, t_n) \circ (t_n, c_p) \circ (c_p, c_q) \Rightarrow (t_m, c_q)$
  - Validation of the traces recovered by the machine learning
    - Suppose that the two relations recovered: $\sigma(t_m, c_q), \sigma(t_m, c_w)$
    - There is a corresponding relation $\sigma(t_m, c_q)$ by the composition rules
      - $\sigma(t_m, c_q)$ is validated
      - $\sigma(t_m, c_w)$ is flagged as possibly false since there is no corresponding relation
LeanArt approach (4/4)

- **Learning algorithm**
  - Based on the classification problem
    - Learner classifies program entities with the probabilities that these entities can be traced to contain elements of UCDs
  - Trained by the different learners separately on the names of program entities and their runtime values
    - Whirl and Naïve Bayes classifier
  - Multistrategy learning approach

Names and probabilities of matching the variable a of the elements of UCDs
Navigating traceability links

- Eclipse plugin for LeanArt
  - Enables programmers to navigate to program entities linked to elements of UCDs by selecting these elements, and vice versa
### Subject programs

<table>
<thead>
<tr>
<th>Program name</th>
<th>LOC</th>
<th># of UCDs</th>
<th># of elements</th>
<th># of $\gamma$ relations</th>
<th># of $\delta$ relations</th>
</tr>
</thead>
<tbody>
<tr>
<td>VA</td>
<td>3,484</td>
<td>3</td>
<td>11</td>
<td>23</td>
<td>2,417</td>
</tr>
<tr>
<td>SCMS</td>
<td>16,332</td>
<td>5</td>
<td>22</td>
<td>68</td>
<td>12,811</td>
</tr>
<tr>
<td>ASM</td>
<td>12,294</td>
<td>3</td>
<td>23</td>
<td>117</td>
<td>18,033</td>
</tr>
<tr>
<td>FreeCol</td>
<td>6,855</td>
<td>2</td>
<td>17</td>
<td>83</td>
<td>13,672</td>
</tr>
<tr>
<td>Jetty</td>
<td>4,613</td>
<td>2</td>
<td>6</td>
<td>12</td>
<td>540</td>
</tr>
<tr>
<td>Megamek</td>
<td>23,782</td>
<td>4</td>
<td>25</td>
<td>76</td>
<td>16,263</td>
</tr>
<tr>
<td>PMD</td>
<td>3,419</td>
<td>3</td>
<td>12</td>
<td>51</td>
<td>913</td>
</tr>
<tr>
<td>VMT</td>
<td>2,926</td>
<td>3</td>
<td>8</td>
<td>24</td>
<td>1,739</td>
</tr>
<tr>
<td>IHIS</td>
<td>1,883</td>
<td>4</td>
<td>14</td>
<td>35</td>
<td>1,208</td>
</tr>
</tbody>
</table>

#### Creating UCDs

- Two groups of graduate students created UCDs based on the available source code and documentation
Experimental evaluation (2/8)

- Threat to validity
  - Students might make mistakes when recovering traceability links
    - This reflects a real world environment when programmers may also make mistakes when providing traceability links
  - Subject programs are of small to moderate size because it is difficult to find a large number of graduate students
    - Large applications may have different characteristics compared to the small to medium size subject programs
  - UCDs for open source programs were created after these programs had been written
    - These UCDs may not be identical to ones that would be created as part of the forward engineering process
Response variables

Basic measures

- **PE**: the number of program entities
- **ITL**: the number of initial traceability links
- **RTL**: the number of traceability links that should be recovered
  - PE – ITL, GTL + BTL (Good traceability links + Bad traceability links)
  - GTL : DTL + ATL
    - DTL : the number of correctly discarded traceability links
    - ATL : the number of correctly accepted traceability links
  - BTL : the sum of CTL and WTL
    - CTL : the number of correct traceability links that are mistakenly discarded by the Validator
    - WTL : the number of wrong traceability links that the Validator accepts
Experimental evaluation (4/8)

Response variables (cont’d)

- **Measures for the quality of LeanArt**
  - **ACC**
    - The ratio of correctly recovered traceability links
    - ATL / RTL
    - Evaluate the performance of the Learner
  - **VPR**
    - The ratio which shows how mistaken the Validator is when analyzing recovered traceability links
    - $\frac{GTL - BTL}{2 \times RTL} + \frac{1}{2}$
    - Shows the combined performance of the Learner and the Validator
  - **BTLR**
    - The ratio of bad traceability links
    - WTL/RTL
## Experimental evaluation (5/8)

### Experiments

- To determine how effective LeanArt is in recovering traceability links

<table>
<thead>
<tr>
<th>Program Name</th>
<th>Run min</th>
<th>PE</th>
<th>ITL</th>
<th>RTL</th>
<th>CTL</th>
<th>WTL</th>
<th>BTL</th>
<th>DTL</th>
<th>ATL</th>
<th>GTL</th>
<th>BTLR</th>
<th>VPR</th>
<th>ACC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Megamek</td>
<td>26</td>
<td>328</td>
<td>20</td>
<td>308</td>
<td>92</td>
<td>21</td>
<td>113</td>
<td>17</td>
<td>178</td>
<td>195</td>
<td>0.07</td>
<td>0.81</td>
<td>0.58</td>
</tr>
<tr>
<td>SCMS</td>
<td>18</td>
<td>296</td>
<td>18</td>
<td>278</td>
<td>26</td>
<td>5</td>
<td>31</td>
<td>19</td>
<td>228</td>
<td>247</td>
<td>0.01</td>
<td>0.89</td>
<td>0.82</td>
</tr>
<tr>
<td>ASM</td>
<td>28</td>
<td>218</td>
<td>13</td>
<td>205</td>
<td>22</td>
<td>8</td>
<td>30</td>
<td>38</td>
<td>137</td>
<td>175</td>
<td>0.04</td>
<td>0.85</td>
<td>0.67</td>
</tr>
<tr>
<td>FreeCol</td>
<td>20</td>
<td>527</td>
<td>31</td>
<td>496</td>
<td>113</td>
<td>15</td>
<td>128</td>
<td>80</td>
<td>288</td>
<td>368</td>
<td>0.03</td>
<td>0.43</td>
<td>0.58</td>
</tr>
<tr>
<td>Jetty</td>
<td>6</td>
<td>96</td>
<td>6</td>
<td>90</td>
<td>12</td>
<td>4</td>
<td>16</td>
<td>10</td>
<td>64</td>
<td>74</td>
<td>0.04</td>
<td>0.55</td>
<td>0.71</td>
</tr>
<tr>
<td>VA</td>
<td>12</td>
<td>119</td>
<td>7</td>
<td>112</td>
<td>10</td>
<td>9</td>
<td>19</td>
<td>8</td>
<td>85</td>
<td>93</td>
<td>0.08</td>
<td>0.83</td>
<td>0.76</td>
</tr>
<tr>
<td>PMD</td>
<td>9</td>
<td>176</td>
<td>11</td>
<td>165</td>
<td>47</td>
<td>22</td>
<td>69</td>
<td>40</td>
<td>56</td>
<td>96</td>
<td>0.13</td>
<td>0.26</td>
<td>0.34</td>
</tr>
<tr>
<td>VMT</td>
<td>14</td>
<td>143</td>
<td>9</td>
<td>134</td>
<td>16</td>
<td>2</td>
<td>18</td>
<td>30</td>
<td>86</td>
<td>126</td>
<td>0.01</td>
<td>0.92</td>
<td>0.87</td>
</tr>
<tr>
<td>IHIS</td>
<td>11</td>
<td>225</td>
<td>14</td>
<td>211</td>
<td>25</td>
<td>4</td>
<td>29</td>
<td>24</td>
<td>158</td>
<td>182</td>
<td>0.02</td>
<td>0.86</td>
<td>0.75</td>
</tr>
</tbody>
</table>

- Written for specific domains with well-defined terminologies
- Whose entity names are easy to interpret and classify
- Code does not use terminologies from any specific domain
Experiments (cont’d)

- To use the Learner trained for the VMT application to recover traceability links for other applications

Operate on similar data
Experiments (cont’d)

- To evaluate whether the validation algorithm can be used to recover traceability links without using the Learner
Experimental evaluation (8/8)

★ Case study

- To determine whether LeanArt and the Eclipse plugin enable programmers to evolve and maintain application efficiently

<table>
<thead>
<tr>
<th>Subject</th>
<th>Time, mins</th>
<th>Correct answers</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Test</td>
<td>Control</td>
</tr>
<tr>
<td>Programmer₁</td>
<td>82</td>
<td>150</td>
</tr>
<tr>
<td>Programmer₂</td>
<td>41</td>
<td>150</td>
</tr>
<tr>
<td>Programmer₃</td>
<td>89</td>
<td>150</td>
</tr>
<tr>
<td>Programmer₄</td>
<td>58</td>
<td>150</td>
</tr>
<tr>
<td>Programmer₅</td>
<td>43</td>
<td>122</td>
</tr>
<tr>
<td>Programmer₆</td>
<td>65</td>
<td>150</td>
</tr>
<tr>
<td>Programmer₇</td>
<td>84</td>
<td>150</td>
</tr>
<tr>
<td>Programmer₈</td>
<td>71</td>
<td>136</td>
</tr>
<tr>
<td>Programmer₉</td>
<td>63</td>
<td>150</td>
</tr>
<tr>
<td>Programmer₁₀</td>
<td>48</td>
<td>150</td>
</tr>
<tr>
<td>Programmer₁₁</td>
<td>85</td>
<td>127</td>
</tr>
</tbody>
</table>

Distinguished by the amount of information

Good performance
Related work

- ARTS
  - One of the earliest systems for automating requirements traceability
  - Allows users to enter programs and requirements manually

- TOOR
  - A tool based on a template-based approach for tracing requirements between different software development artifacts
  - Manually exploits relations between software artifacts

- TraceAnalyzer
  - A tool that detects traceability links between test and usage scenarios, models, and classes in the source code
  - Supports traceability links at a coarse grained level
Summary

- Offered a novel approach for automating part of the process of recovering traceability links between the source code and elements of UCDs
- Evaluated the approach on open source software projects and obtained results that suggest it is effective
- Visualized the recovered traceability links that enables programmers to answer important questions when evolving and maintaining software
Discussion

❖ Evaluation of the approach
  ▪ Limits to recover all the correct links
    • It is critical to precisely analyze change impact of changes
  ▪ Does not cover another models such as class diagram, sequence diagram
  ▪ Depends on the user finding acceptable initial traceability links
    • Accuracy of the randomly selecting initial traceability links may be low
    • It is difficult to decide what is the optimal % for the initial links

❖ Suggestion
  ▪ Utilize the structural information of source code
    • Comparing the dependencies of source code to the transitive relations may increase the precision of the approach
  ▪ Apply incremental approach to find the optimal %