**INTRODUCTION**

- The systematic classification and analysis of defect data bridge the gap between causal analysis and statistical quality control, provide valuable in-process feedback to the system development or maintenance, as well as improve system and software quality.
- The analysis results based on systematic defect classifications enable us to understand the major impact types of system defects and to pinpoint specific problematic areas for focused problem resolution and quality improvement.
- Orthogonal Defect Classification (ODC) is the most influential among general frameworks for software defect classification and analysis.

**Issues with Manual ODC Generation**

- Manual ODC defect classification based on assimilation of existing defect repositories (e.g., defect/bug reports) is extremely effort consuming esp. for novices.
- In the GSM Base Station Systems (GSMBS) software development group at Motorola GSM Products Division, 5 experienced development engineers who are familiar with both project requirements and ODC were assigned to classify all post-release defects since 1996 by defect “impact” attribute. It took them approximately 6 minutes to classify each defect. Considering a 1000 defect data set, it would take approximately 2.5 weeks for 5 experts to complete the classification only for one single ODC attribute.
- The types of follow-on ODC-based defect analyses are often restricted by the limited defect classification results. Multi-way defect analyses may not be supported.

**Major Contribution of AutoODC**

- From a software engineering perspective, we semi-automate ODC defect classification and evaluate our approach on the ODC “impact” attribute.
- From an AI perspective, we propose the annotation relevance framework, which aims to improve automated ODC classification by enabling a machine learning algorithm to exploit additional experts’ domain knowledge expressed in the form of relevant annotations.

**Why Relevant Annotations are Needed: Two Motivating Examples**

<table>
<thead>
<tr>
<th>ID</th>
<th>Summary</th>
<th>Description</th>
<th>Relevant words/Phrases</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Prevent (last) Admin User from self-destruction</td>
<td>My client somehow managed to delete himself, meaning that no admin users continued to exist, so no new accounts - admin or otherwise - could be created, and no one could log into the site! Fortunately the site was in its infancy, so a simple database restore got it up and running again. But imagine the situation if a larger active site had not been backed up and all the admins had deleted - if principle this shouldn’t be possible, but is in fact permitted by the current Elgg core implementation, which provides an “guard code” to ensure that an admin user cannot self-delete, self-un or self-remove admin. The attached code differs provide that functionality, and it would be very useful to see these appear in the forthcoming Elgg 1.7 release, so that the update won’t overwrite the code changes I have made to my client sites.</td>
<td>getting 404 after “system setting” - this is a fresh inst of elgg and after I hit the “save” button on the “system settings” page I get 404 with the message “Oops! This link appears broken.” The link to the address bar points to “<a href="http://computer">http://computer</a> systems website login/” Can someone please help me out there? Thanks.</td>
</tr>
<tr>
<td>2</td>
<td>Generate additional features</td>
<td>annotations</td>
<td>Annotations - automatically generate additional keywords based on ODC instance</td>
</tr>
</tbody>
</table>

**Example 1: Weakened Knowledge**

Long defect description lessens the impact of the important signature, and can potentially cause this report to be misclassified. The phrase “would be” in the last sentence is a strong indicator of the Requirements category (e.g., “would be useful”, “would be great”, “would be cool”), which implies a desired requirement currently missing from the system. Employing “would be” as a relevance annotation highlights its importance, allowing the learning algorithm to focus on this piece of evidence in spite of the long report and correctly classify this example.

**Example 2: Confusing Information**

Though the presence of the word “install” is evidence that the bug belongs to the Improvement category, it is actually an example of Reliability. Exploiting relevance annotations for automated classification could avoid this mistake: the phrase “link appears broken” was marked as evidence that it is an example of Reliability.

**APPROACH: AutoODC Overview**

1. **Defect Report**
2. **Pre-processing defect report**
3. **Generating annotations**
4. **Classification**
5. **ODC Classification**
6. **Annotating**
7. **Relevant Annotations**
8. **Basic Defect Classification Framework**
9. **Annotation Relevance Framework**

**Basic Defect Classification Framework**

**Step 1: Pre-processing defect reports**

- Tokenize and stem with the WordNet lexical knowledge base
- Use unigrams as features and represent each defect record as a binary-valued vector
- Normalize each vector to unit length

**Step 2: Learning ODC classification**

- Use Support Vector Machine (SVM) for classifier training
- One-versus-others training scheme to train one SVM classifier for predicting each class

**Step 3: Classification**

- Apply each trained classifier separately to classify an instance.
- Each classifier returns a confidence value. We assign to a defect record the class whose classifier returns the highest value among the set of values returned by all the classifiers.

**Annotation Relevance Framework**

**Extension 1: Generating pseudo-instances**

- Goal: Augment the training set with additional positive and negative training instances known as pseudo-instances. To create a pseudo-instance, we would remove from the defect report one or more relevant annotations
- create a feature-vector consisting of the remaining unigrams in the report
- Observation: Since a pseudo-instance lacks one or more relevant annotations, the correct SVM classifier should be less confident about its classification than a non-pseudo-instance.
- We implement this observation by creating additional inequality constraints in SVM’s optimization problem.

**Extension 2: Generating additional features**

- Goal: Exploit the relevant annotations to create additional features for training.
- Method: To reduce data sparseness, instead of using a relevant annotation directly as a feature, we create all possible bigrams (i.e., consecutive words of length two) from each relevant annotation and use them as additional features.

**Extension 3: Exploiting domain knowledge**

- Goal: exploit human-supplied domain knowledge to create additional training features.
- Method: Collect all relevant annotations from the training defect records. Have a human analyst partition them so that each cluster contains all and only synonymous relevant annotations.
- Assign a unique ID to each cluster.
- If a relevant annotation is present in the defect record, we create an additional feature that corresponds to the ID of the cluster containing the relevant annotation.

**EVALUATION**

**Experiment Setup**

AutoODC is experimented on classifying defect records under the ODC “impact” attribute. Combinations of basic defect classification system with the 3 extensions were experimented.

**Data Set**

Industrial defect report: 403 defect records in a social network project domain from an industrial company

**Training/testing data preparation**

- Two expert analysts independently classified the 403 defect records into 6 categories under the “impact” attribute with an initial agreement of 90% and then cross-validated to resolve the disagreements.
- Distribution over the 6 categories: Capability (28%), Security (11%), Performance (1%), Reliability (8%), Requirements (39), Usability (6%) (of total 100%)
- Both experts marked the words/phrases relevant for their assigning a defect record to a particular ODC category and identified the synonyms in these relevant words/phrases.

**Evaluation Methodology:** 5-fold cross-validation

**Evaluation Results (Partial)**

**Experiment Accuracy**

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Accuracy</th>
<th>Reliability</th>
<th>Capability</th>
<th>Integrity/Security</th>
<th>Usability</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic</td>
<td>74.2</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Basic+Ext1</td>
<td>71.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Basic+Ext2</td>
<td>76.4</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Basic+Ext4</td>
<td>73.9</td>
<td>0.0</td>
<td>0.7</td>
<td>0.1</td>
<td>0.4</td>
<td>0.0</td>
</tr>
<tr>
<td>Basic+Ext5</td>
<td>74.6</td>
<td>0.0</td>
<td>0.1</td>
<td>0.7</td>
<td>0.4</td>
<td>0.0</td>
</tr>
<tr>
<td>Basic+Ext6</td>
<td>80.2</td>
<td>0.0</td>
<td>0.7</td>
<td>0.0</td>
<td>0.4</td>
<td>0.0</td>
</tr>
</tbody>
</table>

**Conclusion**

- AutoODC produces satisfactory classification results with an accuracy of 80.2% when using manual defect classification as a basis of evaluation, where accuracy is computed as the percentage of defect records correctly classified by our classification system.
- A complement to the manual ODC classification, AutoODC improves the confidence of defect classification results by reducing the investigation set that a human analyst has to examine when performing ODC classification.