Intelligent Component Selection

Valerie Maxville
Edith Cowan University
Perth, WA, Australia
vmaxvill@student.ecu.edu.au

Jocelyn Armarego
Murdoch University
Perth, WA, Australia
jocelyn@eng.murdoch.edu.au

Chiou Peng Lam
Edith Cowan University
Perth, WA, Australia
c.lam@ecu.edu.au

Abstract

Component-based Software Engineering (CBSE) provides solutions to the development of complex and evolving systems. As these systems are created and maintained, the task of selecting components is repeated. The Context-driven Component Evaluation (CdCE) project is developing strategies and techniques for automating a repeatable process for assessing software components. This paper describes our work using Artificial Intelligence (AI) techniques to classify components based on an ideal component specification. Using AI we are able to represent dependencies between attributes, overcoming some of the limitations of existing aggregation-based approaches to component selection.

1 Introduction

One of the dilemmas facing software developers is how to maintain a high level of quality and trust in systems that are increasingly complex and rapidly evolving. Component-based Software Engineering (CBSE) is based on reusable software components that can be replaced or updated easily [1]. These components are developed in-house and/or acquired from third party vendors. Commercial-Off-the-Shelf (COTS) components share many issues with COTS software in general. Trust in third party software tends to be low as the application developer has not been involved in the development and testing, and could fear that malicious code is included in the software [2]. Component testing and/or certification can help to increase the level of trust in individual components [3]. However, many of the issues with components appear when they are placed in context, with mismatches and context-dependencies affecting the correctness of the component's performance in the target environment [4]. The process of selecting components is made more complex by the generality of the components. It is important that the selection process be context-aware to ensure that the chosen component will work correctly in the target environment. The selection process must also be repeatable and scalable to allow confidence that sufficient numbers of components have been considered before a choice has been made.

Other issues making component selection difficult are the evolution of systems, documentation that is available and methods of assessment. Frequent releases of components and changes in the software marketplace will dictate that the selection process needs to be repeated and ripple effects may result in a continous cycle of updates [5]. Component developers and brokers do not have a standard for describing their components, resulting in varied levels of documentation that are difficult to compare. An intrinsic issue with components is that there are few (if any) artifacts available from the development process, and access to the source code is unlikely [6]. With a growing market of components from which to choose, and rapidly evolving systems, it is important to minimise the manual effort required to assess each component.

In the Context-driven Component Evaluation (CdCE) Project[1] we approach the selection of third party components by creating an ideal specification of the requirements for the component(s) in a particular project. Any CBSE development will need to document requirements for aquisition of components, so this specification does not cause additional work for developers. The ideal specification is used to generate training data for the AI classifiers, as well as for the generation of tests for dynamic assessment. AI is a largely unexplored area in component selection. We seek to use AI to automate the assessment of components, allowing a larger number to be considered. Maintenance and reassessment of components is simplified as the model can be reused for subsequent searching and component evaluation. Many of the existing component selection techniques use an aggregation approach for determining a recommendation. This assumes independence of selection criteria, which is unlikely to be true. Our approach incorporates interplay between criteria. We are investigating AI techniques that can learn these dependencies and classify the components ac-

[1] Formerly known as the Context-driven Component Testing Project

Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC'04)
0730-3157/04 $20.00 © 2004 **IEEE**

cordingly.

In the following section looks at current approaches for selecting components and the application of AI in this area. We then describe our project and the CdCE Process for component selection. Section 4 shows our work using AI techniques to classify components and we conclude with a summary of our observations and a description of the future work for this project.

2 Component Selection

Much of the work in component quality and trust relies on a correct and detailed specification of the component under consideration. Evaluation of components aims to determine the suitability of a component to a particular project based on the project requirements and the attributes of the candidate component. There are number of models for component specification [11][12], usually grouping the attributes as functional and non-functional. In most cases a template is offered where the values or scores are entered for each attribute. Other approaches provide a process for identifying and organising the criteria for each selection task [7][13]. COTS component selection needs to be more flexible on requirements as an exact match may not be possible and a loosening of criteria may be required [6]. Some approaches deal with this issue by iterating stages of defining and refining criteria then assessing the components returned from a matching process, e.g. using a goal-driven approach [14]. It should be noted that many of the selection processes require a manual assessment on each criterion, with those adopting the AHP requiring a pairwise comparison between all components on each criterion [7].

Given a set of attributes, the ideal specification (requirements) and the values presented by a particular component, the screening or short-listing process needs to combine the data to create a ranking or recommendation. Much of the literature uses the Weighted Sum Method (WSM) to aggregate a value by summing attribute weights multiplied by their respective values [15][16]. Criticism of the WSM includes the summing of differing types of data (e.g. cost plus memory plus quality), lack of process for determining attribute weights and the inherent problem with the formula losing dependency information between attributes (e.g. conflicts and co-requisites). A commonly used alternative is the AHP, which includes a method for determining weights and component scores against attributes [5][7][17]. These scores are based on pairwise comparisons, and thus use the same 'units', even when combining qualitative and quantitative data. Features of the AHP are that it organises the criteria into a hierarchy (e.g. group quality attributes as subnodes of the quality node) and that scores can be consistency checked. Disadvantages are the number of pairwise comparisons (and therefore time) required and that the interplay between the attributes is lost as the final aggregation is essentially the WSM formula. A technical criticism of the AHP is the rank-reversal problem, which can be addressed by using a multiplicative formula for aggregation [18].

The common problems with WSM and AHP stem from the assumption that attributes are independent, resulting in compensations in scores and 'passing' unworkable combinations of values (e.g. C# with Linux). Another option for COTS and component selection is the outranking approach using the ELECTRE family of methods [19]. These methods rank each candidate on each attribute and determine an outranking relationship to categorise attributes into preferred and non-preferred. As with the AHP, comparisons are made between candidates on each attribute, removing the issue of units and attribute types. Although it has been successfully used for software evaluation [20][21], there are issues with explaining the reasoning for decisions and that a complete ranking may not be possible [5].

As we consider attribute dependencies to be important, along with reducing time and effort through automation, we look to the field of AI for applicable techniques. Those currently being used with components concentrate on fuzzy logic. Some work on the fuzzy retrieval of components via faceted classification [22][23], while others use formal specifications to carry out fuzzy clustering [24]. The formalisation of specifications in repositories allows for the use of a range of AI techniques, however we do not include formal specification of candidates in the scope of our work.

3 Context-driven Component Evaluation

The CdCE Project provides a formalised process for the evaluation and testing of components, and is developing tools and strategies for a high level of automation. We approach component selection as a classification problem: assigning each component to a class based on its assessed suitability to the project. As part of this project we have produced the CdCE Process for assessing components (Figure 1).

Our process begins with the development of the ideal component specification (Step 1). A critical factor in automation of component assessment is that brokers and repositories adopt a standardised template for documenting their components. This will include context-specific values, priorities and interplay of attributes to determine the classification criteria. We then train the system to recognise suitable components, compiling a short-list of components from selected repositories in Step 2. If the short-list is not acceptable, the developer is provided with an analysis of the component data to guide the refinement of the ideal specification. The formal specification of the behaviour of the component, which can include context-specific aspects such as usage profiles and interfacing systems, is used to

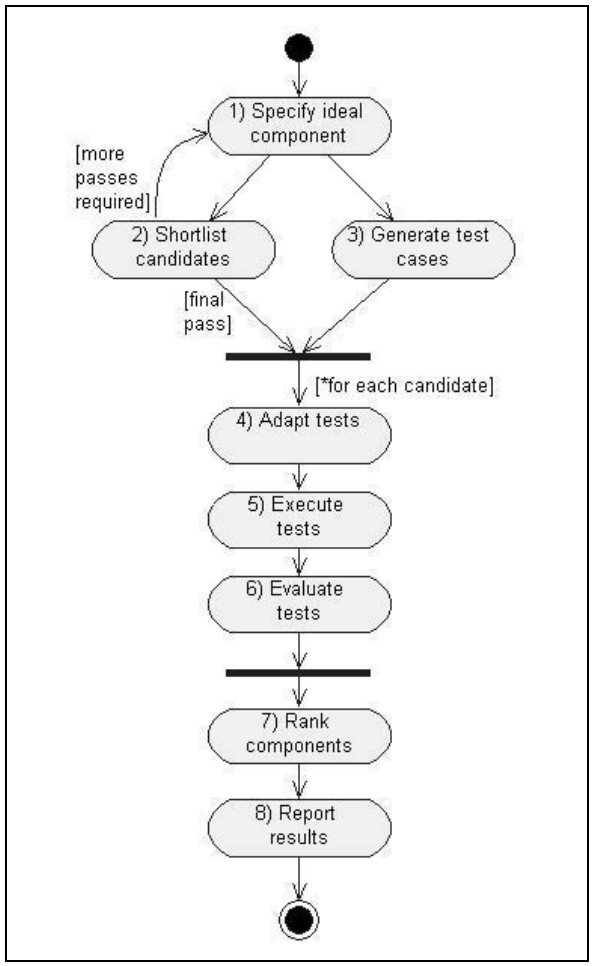

Figure 1. The CdCE Process for Component Selection

generate tests in Step 3. These are abstract tests that will be adapted to each candidate component in Step 4. Using a test-suite generated from the ideal specification allows us to make meaningful comparisons between components. The tests can also be used for system and regression testing. The results from executing the tests in Step 5 are evaluated in Step 6. If an alternate or supplementary evaluation is preferred, Steps 3, 4, 5 and 6 can be combined with other assessment methods.

Step 7 involves the assessment of components and their ranking. We have investigated and used a number of techniques for taking the actual values a component displays for the attributes and giving an overall assessment. These include heuristics (manual), Weighted Score Method (WSM) [7], the Analytical Hierarchy Process (AHP) [8] and Expert Systems. Our current interest is to train classifiers to recognise suitable components based on AI techniques. The

```
<?xml version="1.0"?>
<Description xmlns="http://www.scis.ecu.edu.au/swvML/1.0/"
      xmlns:dc="http://purl.org/dc/elements/1.0/"
      xmlns:swv="http://www.scis.ecu.edu.au/swvML/1.0/" >
  <dc:description type="mandatory">scientific</dc:description>
  <dc:description>calculator</dc:description>
  <swv:devStatus type="mandatory">mature</swv:devStatus>
  <swv:licence type="preferred">GPL</swv:licence>
  <swv:price type="preferred" min="25" max="50">40</swv:price>
  <swv:technical>
    <swv:devLanguage type="preferred">Java</swv:devLanguage>
    <swv:devLanguage>C++</swv:devLanguage>
    <swv:operatingSystem type="mandatory">Linux
    </swv:operatingSystem>
    <swv:systemRequirements>
      <swv:memory type="preferred" min="15" max="50">20
      </swv:memory>
      <swv:diskSpace type="preferred" min="30" max="50">40
      </swv:diskSpace>
    </swv:systemRequirements>
  </swv:technical>
</Description>
```

Figure 2. Ideal Component Specification

two described in this paper are the C4.5 algorithm for generating decision trees [9] and neural networks. All of the techniques listed are applicable to both the short-listing and ranking tasks (Steps 2 and 7). The final step generates a report on the results which includes the reasoning behind the recommendation(s) and information to assist in the adaptation of component(s) to the target system. A more detailed description of the CdCE process appears in [10].

4 Classifying Components

The scenario for this case study is the selection of components to provide scientific calculator functionality. As it is an exploratory case study, the selection criteria are simple. There are four *mandatory* and six *preferable* criteria. Attributes not in those categories remain as the default priority *other*. We assigned adjustable thresholds to require four out of four *mandatory* and three out of six of the *preferred* criteria to be satisfied. The ideal component specification is given in Fig. 2.

For this investigation we selected two AI techniques to generate classifiers. They are Quinlan's decision tree algorithm (C4.5), and a neural network. Both classifiers are available through the Weka software package for machine learning [25].

4.1 Training Data Generation

Our two classifiers were trained on the same data, generated from the ideal specification. The data generator has been developed to create a data distribution that captures the complexity of the criteria used in the assessment, while avoiding an internal bias. Each training dataset is validated

using Weka's 10-fold cross-validation and those used in this case study scored over 96%. The generator application automates the labelling of the data into output classes, allowing us to use supervised learning techniques to train the classifers.

Early tests with data generation showed that the C4.5 classifier was highly sensitive to the distribution of the data[2]. If too high a proportion of the instances were rejected (e.g. 82%), then the classifier rejected them all. The neural network did not oversimplify the classification, but also saw performance improvements with the optimised training data.

The data is now generated using a technique similar to boundary value analysis in test case generation [26]. We concentrate on discriminating values close to the border between acceptance and rejection. Incremental experiments provided feedback on the distribution of data required for the classifiers to learn strongly visible patterns, such as identifying *mandatory* and *other* attributes. Further work is required for determining the best way to deal with ranges of numeric values. Validation and training results show that this approach is successful, but may be able to be improved.

4.2 C4.5

The most widely used decision tree classifier is the C4.5 algorithm [9]. C4.5 generates a tree incorporating all instances within the dataset and their classifications. It then groups the data to allow 'pruning' to create a manageable decision tree. The decision tree for this dataset has 147 nodes, including 74 leaves (classification points).

Table 1 details the results for the C4.5 classifier against the case study training and test sets. The results for the training dataset included twelve misclassifications out of 2736 (below 0.5% error rate). These errors were all in situations where the mandatory requirements were met and the assessment of the preferable criteria failed. We are investigating additional improvements to the generator algorithm to resolve these problems and move closer to 100% correct classification.

We then assessed unseen data to further evaluate the classifier. The datsets represent all combinations of attribute values that are acceptable (Unseen Dataset 1) and those that should be rejected (Unseen Dataset 2). Attributes that do not affect the decision (*other*) are randomised. Both datasets were classified with an acceptable level of accuracy. Unseen data will generally have more classification errors than the training data. Dataset 1 performed better than the training due to the distribution of the values and the size of the dataset. The training sets is focussed on the combinations close to acceptance/rejection, whereas many of the combinations in the unseen datasets were clearly in one of these classes (e.g. reject with no matches of mandatory attributes).

[2]An alternative approach to dealing with imbalance in the distribution of data is the use of cost matrices. This would have the side effect of lowering the overall performance of the classifer and is therefore not appropriate for our work.

Table 1. C4.5 Performance

Dataset	% Correctly Classified	Total Instances
Training	99.5614%	2736
10-fold Cross-validation	96.3085%	2736
Unseen Dataset 1	100%	96
Unseen Dataset 2	98.1183%	744

4.3 Artificial Neural Network

The neural network classifier is based on a simulation of the neurons of the human brain, organised into interconnected layers. In Weka's implementation, a backwards propogation algorithm updates the weights connecting the neurons and reinforces those that result in a correct classification.

There are many parameters available to tune a neural network. We used an empirical approach to determine their effect on the classification of the case study data. The default parameters were 500 epochs, a learning rate of 0.3, the momentum value of 0.2 and a network with 18 nodes in the hidden layer. Sensitivity analysis indicated that changing the number of epochs had little effect on the classification: the network had converged before 250 epochs. A learning rate of 0.1 performed poorly on the test sets, while each of 0.2, 0.3 and 0.4 had similar results. Increasing the momentum produced a downward trend in performance, although all of the training results were above 98%.

There are an infinite number of possible configurations of the neural network. For most problems, a network with two nodes in a hidden layer is sufficient. We used this as the baseline and sought an improved configuration. After conducting specific sensitivity analysis between sample data and the network configuration, we chose a network with eighteen nodes in a single hidden layer. This was of similar performance to a ten node hidden layer and over 10% better at classifying the training data than a two node configuration. We will need to investigate the configuration further with future datasets as networks with two hidden layers also performed well (10,5 nodes and 10,2 nodes) and may be better in more complex selection tasks.

The results for the eighteen node neural network are given in Table 2. The model correctly classified over 99.5% of the instances in the dataset. Investigation of the misclassified instances showed that they related to the boundaries

Table 2. Neural Network Performance

Dataset	% Correctly Classified	Total Instances
Training	99.5249%	2736
10-fold Cross-validation	96.6009%	2736
Unseen Dataset 1	90.625%	96
Unseen Dataset 2	90.3274%	744

of the numeric values. For example, a value of $14 for price may have been accepted, when it should strictly have been rejected as being below $15. All of the errors in classification resulted from similar borderline cases. We hope that modifying the generator algorithm will reduce these errors. The classification of unseen datasets produced good results that can be improved upon. Refinement of the generator algorithm should result in more accurate classification.

4.4 Interplay

Exploration of the ability to recognise interplay between attributes has also produced good results. In a simple example of interplay, we looked at three interrelated attributes: development language, framework and operating system. The scenario is that an organisation has expertise in Java, C++ and C#, ActiveX, Enterprise JavaBeans (EJB) and acceptable platforms are Windows and Linux. Clearly, there are combinations of the three attributes that will be preferred, for example, (Java,EJB,Linux) and (C#,ActiveX,Windows). Combinations to avoid include ('all',ActiveX,Linux) and (C#,EJB,'all'). The training data for this scenario is given an output class ranging from 0 (not acceptable) to 5 (recommended). Using this data to generate C4.5 and neural network classifiers resulted in 100% correct classification of the data for both techniques. 10-fold cross-validation on the data was also 100%. The resulting C4.5 decision tree (Fig. 3) shows the reasoning used by the classifier to allocate a 'score' to each component based on the attribute values. The different treatment of the attribute in each branch of the tree is the key to the correct assessment of the attribute interplay. In an aggregation based approach, a component with conflicting attribute values would still score highly as each attribute is independently valid. For example (Java,EJB,Linux) would score (5,5,5)=15/15 as would the inadvisable (C#,EJB,Linux). Weightings upon the attributes could not differentiate these results.

4.5 Observations

The case study results for both classifiers are promising in that they give a high degree of accuracy in identifying suitable components. A similar percentage of components

```
devLanguage = Java
|   operatingSystem = Linux
|   |   framework = EJB: 5
|   |   framework = ActiveX: 0
|   operatingSystem = Windows: 5
devLanguage = C++
|   framework = EJB: 0
|   framework = ActiveX
|   |   operatingSystem = Linux: 0
|   |   operatingSystem = Windows: 3
devLanguage = C#
|   operatingSystem = Linux: 0
|   operatingSystem = Windows
|   |   framework = EJB: 0
|   |   framework = ActiveX: 5
```

Figure 3. Decision Tree for Interplay Dataset

were classified incorrectly during training, but there was no overlap in the instances that caused confusion. Improvements to the generator algorithm are expected to correct most, if not all of these classification problems. Both classifiers performed well when classifying unseen data (over 98% and 90%). This indicates the classifiers are able to correctly identify suitable components with high accuracy, based on the ideal specification of the component. Results of over 96% in 10-fold cross-validation of the training data gives confidence that the data itself did not bias the training of the classifiers.

Our investigation of generating classifiers to recognise attribute interplay were also successful. A small study was carried out to ensure that the classifiers are capable of dealing with this more complex combination of data. Both classifiers correctly classified all instances in the data. We will be investigating interplay with a larger case study in the future.

5 Conclusion

In this paper we have described a technique for training AI classifiers to assist the selection of software components for development projects. The training data is generated from an ideal specification of the required component, using an XML Schema as a generalised template. Using the XML Schema and the instance document for the ideal component, the data generator creates an internal model of the component. The training data is automatically labelled into classes, overcoming one of the difficulties with supervised learning. The results from training a C4.5 decision tree algorithm and an Artificial Neural Network were better than 99%. Ten-fold validation of the training data set produced results over 96%.

The trained classifers managed to correctly identify 98% (C4.5) and 90% (neural network) of the test datasets. Planned improvements to the generator algorithm should bring about further improvements in these results. Another

trial for the classifiers was a study to test if they could capture the interplay between component attributes. The study demonstrated that both classifiers performed correctly when given data including interplay (both rating 100%).

This paper explores some of the possibilities of using AI in component selection. We have developed a specification model for software components that includes context and interplay information. This specification is used to automate a process for the assessment of components, allowing the process to be repeated, and to be applied to large numbers of components. As an alternative to aggregation-based techniques, we have used C4.5 and neural network classifiers to recognise suitable components, with a high level of accuracy. The techniques demonstrated are able to capture the interplay between attributes, unlike aggregation techniques. We have little overhead to our approach as the specification of the desired component must be developed for any selection task. The decision tree produced by C4.5 can provide reasoning for decisions that are made, allowing confidence and trust in the recommendations.

Future work for the CdCE Project is to complete a larger case study using data harvested from component repositories. We are investigating alternative data representations to suit clustering and mining of association rules. These tools will provide information for the refinement of ideal specifications. We will also enhance the training data generator.

References

[1] C. Szyperski. Component software: beyond object-oriented programming. New York: ACM Press, 1997.

[2] M. Sparling. Lessons Learned Through Six Years of Component-Based Development. Communications of the ACM, 43(10), pp. 47-53, 2000.

[3] J. Voas. Developing a Usage-Based Software Certification Process, IEEE Computer, vol. 33, pp. 32-37, 2000.

[4] E. Weyuker. Testing Component-based Software: A Cautionary Tale. IEEE Software, 15(5), pp. 54-59., 1998

[5] D. Kunda. STACE: Social Technical Approach to COTS Software Evaluation, In: Cechich et al. (Eds.) Component-Based Software Quality, LNCS 2693, pp. 64-84, Springer-Verlag Berlin Heidelberg, 2003.

[6] A. Cechich, M. Piattini, and A. Vallecillo, Assessing Component-based Systems, In: Cechich et al. (Eds.) Component-Based Software Quality, LNCS 2693, pp. 1-20, Springer-Verlag Berlin Heidelberg, 2003.

[7] J. Kontio. A COTS Selection Method and Experiences of its Use. In Proceedings of the 20th Annual Software Engineering Workshop, 1995.

[8] T. L. Saaty. The Analytical Hierarchy Process, McGraw-Hill, 1990

[9] J. R. Quinlan. C4.5: Programs for Machine Learning, Morgan Kaufmann, San Francisco, 1993.

[10] V. Maxville, C. P. Lam and J. Armarego. Selecting Components: a Process for Context-Driven Evaluation, in Proceedings of the 10th Asia-Pacific Software Engineering Conference, Chiang Mai, Thailand, 10-12 Dec, 2003.

[11] S. B. Sassi, L. L. Jilani and H. H. B. Ghezala, COTS Characterisation Model in a COTS-Based Development Environment, in Proceedings of the 10th Asia-Pacific Software Engineering Conference, Chiang Mai, Thailand, 10-12 Dec, 2003.

[12] B. Christiansson and M-T. Christiansson. The Missing Approach for Component Specification, 1st International Workshop on Component Based Business Information Systems Engineering, Geneva, Switzerland, 2nd Sept, 2003.

[13] M. Ochs, D. Pfahl, G. Chrobok-Diening and Nothhelfer-Kolb A COTS Acquisition Process: Definition and Application Experience. Tech. Report IESE-002.00/E, Fraunhofer Institut Experimentelles Software Engineering, 2000

[14] C. Alves and A. Finkelstein. Challenges in COTS Decision-Making: A Goal-driven Requirements Engineering Perspective. In Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE '02), 2002

[15] F. Williams. Appraisal and Evaluation of Software Products, Journal of Information Science, Principles and Practice, Vol. 18, pp. 121-125, 1992.

[16] H. Solberg and K. M. Dahl, "COTS Software Evaluation and Integration Issues," Norwegian Institute of Technology and Science November, 2001

[17] N. A. Maiden and C. Ncube. Acquiring COTS Software Selection Requirements, IEEE Software, pp.46-56, 1998

[18] E. Triantaphyllou. Multi-Criteria Decision Making Methods: A Comparative Study Kluwer Academic Publishers, 2001.

[19] B. Roy. The Outranking Approach and the Foundations of the ELECTRE Methods, Theory and Decision, Vol. 31, pp. 49-73, Kluwer Academic Publishers, Netherlands, 1991.

[20] E. Anderson. A Heuristic for Software Evaluation and Selection, Software Practice and Experience, 19(8), pp. 707-717, 1989.

[21] M. Morisio and A. Tsoukis. Iusware: a Methodology for the Evaluation and Selection of Software Products, IEEE Proceedings of Software Engineering 144(3), pp. 162-174, 1997

[22] E. Damiani and M. G. Fugini. Automatic thesaurus construction supporting fuzzy retrieval of reusable components. Proceedings of the ACM Symposium on Applied Computing, Tennessee, US, pp. 542 - 547, 1995

[23] W. Pedrycz and J. Waletzky. Fuzzy clustering in software reusability, Software - Practice and Experience, vol.27, no.3, p. 245-70, 1997

[24] S. Nakkrasae, P. Sophatsathit, and W. R. Edwards, Jr.. Fuzzy Subtractive Clustering Based Indexing Approach for Software Components Classification Proceedings of the 1st ACIS International Conference on Software Engineering Research & Applications (SERA'03), San Francisco, USA., June 25-27, 2003, pp. 100-105.

[25] I. Witten and E. Frank. Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations, Morgan Kaufmann Publishers, USA, 1999.

[26] G Myers. The Art of Software Testing, John Wiley & Sons, 1979.