Learning to Select Software Components

Valerie Maxville[1], Chiou Peng Lam[1], Jocelyn Armarego[2]
[1]Edith Cowan University, [2]Murdoch University
vmaxvill@student.ecu.edu.au, c.lam@cowan.edu.au,
jocelyn@eng.murdoch.edu.au

Abstract. Developers using software components need to be confident in their selection of the most suitable component. Manual searching is time consuming and unlikely to be able to consider large numbers of components. The Context-driven Component Evaluation (CdCE) project is investigating ways to use Artificial Intelligence to assist the selection process. This paper describes our Machine Learning approach where we train a system to recognise candidates that match an ideal component specification. We utilise automated test generation techniques to create data for training the system. This results in a generic assessment system that can automatically short-list components for further investigation.

1. Introduction

Software Engineering is a movement to apply engineering principles to software development. Component-based Software Engineering (CBSE) uses software components as the building blocks for new systems, similar to hardware components. Software components are replaceable, reusable modules of executable code with well-defined interfaces [1]. As CBSE becomes more popular, we are presented with a range of components for a given application. Developers need a means for selecting the most suitable components from the growing number available in repositories and broker sites. This is not only during initial development, but also when updating components or the surrounding system.

The component selection task is normally undertaken by experts who use heuristics to determine which components are to be selected or investigated further. The desire to evaluate components using a repeatable, traceable method leads us to develop evaluation processes, such as the Context-driven Component Evaluation (CdCE) Process. Structured processes allow us to standardise how we deal with candidates, but a manual assessment is unable to scale to large numbers of components. We propose that Artificial Intelligence (AI) techniques be applied to automate parts of the selection process to allow the consideration of larger numbers of candidates.

A common approach to assessing components is to take weighted scores against a list of attributes and aggregate them. An expert's holistic view of a candidate may incorporate interplay between attributes – conflicting or reinforcing its suitability. This interplay can be recorded as a series of relations between attributes. Rules associated with these relations can then interact with the candidate's "scores" against attributes and their overall evaluation. This interplay between attributes is lost in a numerical aggregation. In this paper we describe our approach to selecting components, which works from a specification of the ideal component, then uses machine learning and test case generation techniques to train the system to automatically evaluate candidate components. Our Selector system automates the determination of rules and building the knowledge base so that the user interface is simple and intuitive. We address the issues of scalability, attribute interplay and the ability to explain the reasoning behind a selection decision.

The following section discusses component selection, AI techniques and the application of AI to component selection. Section 3 describes our Machine Learning approach to selection. A case study is presented in Section 4. The CdCE Project is described in Section 5, with conclusions and future work in Section 6.

2. Related work

Selection of components is a similar problem to selection of Commercial Off-The-Shelf (COTS) software, and COTS research can be applied to component selection. Research in component selection begins with defining the selection criteria. Most selection approaches have a component model that describes the criteria or attributes to be used in the assessment, often implemented as a hierarchy. A discussion of these models is given in [3]. Other schemes develop a hierarchy for the specific problem [4][5]. The relative importance of the criteria may be determined using a structured approach such as AHP [2][5]. An assessment of each component against

the criteria is then carried out, most often as a manual process. A recommendation or ranking can then be determined. This normally involves an aggregation of results using the Weighted Scoring Method (WSM) or the AHP [2]. In other cases, techniques such as Outranking are used [6]. Recent research has begun to use AI techniques to address issues with assessing components, in particular the inherent problems with aggregating results. Neuro-fuzzy [7] and Rough fuzzy sets [8] have been used to deal with imprecision and uncertainty in component assessment, while overcoming some overheads of determining the original fuzzy sets. Most techniques are more applicable to in-house repositories where the documentation of components can be standardised and detailed, with up to 1320 attributes for each component [9]. Our project is concerned with third party components sourced from a range of repositories. We then have a very large number of components to screen and rudimentary information about them. This leads us to AI to carry out both coarse screening and more in-depth analysis of the technical features of candidate components. It is important that the overheads for the AI technique are low as each selection process will have new requirements and is thus a new problem.

Artificial Intelligence is a field that provides a range of techniques for representing and processing knowledge. When selecting an AI technique, it is important to consider the features that are needed, and which are more critical to the particular problem. In the component selection problem, we are trying to classify the components as being acceptable or rejected. We also want to be able to adjust thresholds to include or exclude more candidates, where criteria may have been too restrictive or lenient. Working with metadata from various sources introduces inconsistency to our data, so some tolerance for missing or uncertain data is important.

Tables 1 and 2 in this document show how traditional and hybrid AI systems perform against eight criteria, all which are quite important for the automated selection of components. Knowledge representation is important to component selection as our process is working with the metadata supplied by vendors and brokers, and the results need to be understandable to users. As we will be working with information from diverse sources, there is a risk of missing and uncertain data. Many of the selection criteria for components can be considered “a match” or “not a match”, but the facility to deal with imprecision may be more useful when looking at how well a description meets our needs, e.g. how close the cost of the component is to our ideal.

Our interest in AI is to automate the selection of components. An automated assessment is unlikely to be trusted unless there are explanation facilities to give the reasoning behind any decisions. The traditional AI systems that perform well on explanation ability rate poorly on adaptability, learning and maintenance. This would lead to a trade off where the reasoning can be explained, but there is a heavy load on the expert to develop and tune rules – diminishing the advantage of using AI. Neural expert or neuro-fuzzy systems may overcome this, assuming we can generate data to train the neural network. Our interest in looking for components

Table 1. Comparison of Traditional AI Techniques, adapted from [10]

Feature	Expert Systems	Fuzzy Systems	Neural Networks	Genetic Algorithms	C4.5 Classification
Knowledge representation	+	++	--	-	+
Uncertainty tolerance	+	++	++	++	-
Imprecision tolerance	--	++	++	++	--
Adaptability	--	-	++	++	++
Learning ability	--	--	++	++	++
Explanation ability	++	++	--	-	++
Knowledge discovery and data mining	--	-	++	+	+
Maintainability	--	+	++	+	++

Table 2. Comparison of Hybrid AI Techniques

Feature	Neural Expert Systems	Neuro-fuzzy Systems	Evolutionary Neural Networks	Fuzzy Evolutionary Systems
Knowledge representation	+	++	--	++
Uncertainty tolerance	++	++	++	++
Imprecision tolerance	++	++	++	++
Adaptability	++	++	++	+
Learning ability	++	++	++	+
Explanation ability	++	++	--	++
Knowledge discovery and data mining	--	-	++	+
Maintainability	++	++	++	+

on the Internet does imply an interest in data mining and an AI technique that can extend to knowledge discovery would be an advantage. Although not one of the criteria in the comparison tables, we also want to be able to deal with the interplay between attributes. Any of expert systems, fuzzy systems or neural networks is capable of encoding these dependencies.

It is clear that an AI technique for component selection would ideally rate well in all of the above categories. For this investigation, we have chosen to use the C4.5 decision tree classifier [11]. C4.5 takes labelled data and grows a large tree which is pruned to create a decision tree of understandable size. As can be seen in Table 1, this approach rates well in all features except uncertainty and imprecision tolerance. We will be addressing these very important issues in future work by investigating other AI techniques for classifying data. An evaluation of the relative performance will then be carried out.

3. Intelligent Selection

Any selection process begins with a requirements specification for comparison with candidate components. We work with an ideal specification based on an XML Schema template [12]. The ideal specification includes all attributes of interest to the application developer. The specification is annotated with information regarding the priority of attributes and any interplay between them. Our system combines the ideal specification with the schema definition to create an internal model of the desired component. The system is not tied to the CdCE component model and can import any XML Schema and instance documents for a generic selection problem.

The attributes describing the components are split into four categories, according to datatype. This binding is determined from the Schema document. The simplest is "string" where there can only be a single value per candidate. An example is the `dc:creator` attribute – the Dublin Core [13] tag representing the software developer. We also have date and numeric attributes where an optimal value is specified along with optional minimum and maximum values. The final datatype is the multiString. A multiString is used in situations where an attribute can have one of a set of values. MultiString attributes are split into multiple attributes for input into the machine learning software. For example, the desired values for the `operatingSystem` attribute may be `UNIX` and `Linux`. We map these to `operatingSystem_UNIX` and `operating-System_Linux` for training data and the data to be assessed.

An issue in using supervised learning techniques[1] is that they require either large amounts of historical data, or a manual evaluation of input data. We have used techniques from test data generation to create a set of training data from the internal model of the ideal component. Values for each attribute are grouped into equivalence classes, and the attributes themselves are grouped according to how they influence the evaluation. The internal model provides enough information to determine whether a component is accepted or rejected, which is used to attach a result to the generated datasets.

Exhaustive generation of data is not practical. With 32 attributes in use of simple types (Boolean for this calculation), we would need over 4,000 million data entries to cover all combinations of the data. The algorithm for generating the data targets groups of entries as "lessons" to train the system to learn a specific aspect of the assessment. The lessons first focus on distinguishing datasets to help the system learn where the border between acceptance or rejection of a component lies. It also creates lessons around areas of complex interaction between attributes to reinforce the learning process. The size of the generated dataset is dependent of the number of attributes in the selection task, and the amount of interplay between them. Training the C4.5 classifier is not greatly affected by the size of the dataset, and takes a few seconds. We have used the same data with an Artificial Neural Network which takes over 20 minutes to generate the classifier.

A balance is required between the amount of data in each output category. Early training sets oversimplified the decision to "reject all" due to the selection of training data. A component selection process is likely to reject a high percentage of candidates, but using data that follows that distribution skews the training. We are continuing to investigate how to balance the training to work with an optimal size and number of lessons. The best results to date have come from generating between $^1/_2$ to $^2/_3$ of the training data falling within the acceptable class.

4. Case Study

In this case study we revisit a manual selection exercise working with real world data, updating it to use machine learning. The scenario for the case study is the selection of a software component to provide scientific calculator functionality. The XML for the ideal specification is given in Figure 1. It uses the CdCE Schema as a base

[1] Supervised learning relies on a manual labelling of the training data for the system to learn the patterns for each classification. Unsupervised learning works with the patterns formed within the training data and attempts to group them into clusters. Data falling inside a cluster can them be labelled according the closest cluster.

with 32 attributes in the following categories: three numeric, one date, 21 String and seven multiString (enumerated). The Schema allows some tags to be repeated, which map to multiString attributes in our system. Potential component information was taken from four online sites. These had been assessed previously with a manual application of the CdCE process. A summary of that assessment is in Table 3. It gives an indication of the rejection rate and the number of possiblilites that would need to be considered in a component selection problem. The automation of selection is highly dependent on the adoption of a standard specification format by component brokers.

The attributes used in the case study are shown in Table 4. The attributes are classified as being *mandatory*, *preferred* or *other*. *Mandatory* attributes must all be met for the candidate to be accepted. A threshold value modifies the number of *preferred* attributes that need to be matched to accept the candidate. The *other* attributes do not affect the assessment. This provides three equivalence classes for our data generation. Test generation uses equivalence classes to reduce the number of test cases by having one value represent the whole class of values, and may have rules for the output based on the input class. For training the system, we use the equivalence classes to enumerate the combinations of attribute values inside and between classes, and the corresponding classification for that component. In this case, three of the attributes are *mandatory* and five are *preferred*, the remaining attributes are categorised as *other*. We have arbitrarily selected a threshold of 0.5 which rounds down to two out of five preferred attributes for acceptance.

We use the Weka system [15] to access machine learning algorithms. Weka uses ARFF format files where attributes and values are listed, then the training and/or test data. For supervised learning, the last attribute denotes the classification of the entry, in this case `result=accept/reject`. As mentioned previously, the generated training data is grouped into lessons. We start with lessons in acceptable attribute values, then look at what values will lead to rejection. Parameters on the generation can adjust the number and size of lessons. The lessons focus on the patterns of attribute values that are near the border of acceptable/unacceptable. Random selection of training data would almost certainly result in all candidates being classified as rejected. Our solution is to apply Boundary Value Analysis (BVA) techniques. We select training data that sits close to the boundary between acceptance and rejection, along with some more straight-forward entries. This has prevented the classifier from over-simplifying its decision tree and allows us to work with relatively small training sets.

Table 3. Case Study Manual Assessment [14]

Site	Number of entries	Number of candidates
I	>8,000	1
II	>12,000	7
III	>36,000	4
IV	>30,000	0
Total	>86,000	9 (3 duplicates)

Table 4. Case Study Ideal Specification

Attribute	Type	Importance	Values
Description	Multi-String	Mandatory	Scientific Calculator
Development Status	String	Mandatory	Mature
Licence	String	Preferred	GPL
Price	Numeric	Preferred	$0-$75
Development Language	Multi-String	Preferred	Java C++
Operating System	Multi-String	Mandatory	Linux
Memory	Numeric	Preferred	5-70Mb
Disk Space	Numeric	Preferred	10-90Mb

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="swvML_ap_1.1.xsl"?>
<Description xmlns="http://www.scis.ecu.edu.au/research/PhD/vmaxvill/swvML/1.0/"
    xmlns:dc="http://purl.org/dc/elements/1.0/"
    xmlns:swv="http://www.scis.ecu.edu.au/research/PhD/vmaxvill/swvML/1.0/"
    about="http://www.scis.ecu.edu.au/research/PhD/vmaxvill/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="0" max="75">25</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="5" max="70">20</swv:memory>
      <swv:diskSpace type="preferred" min="10" max="90">40</swv:diskSpace>
    </swv:systemRequirements>
  </swv:technical>
</Description>
```

Figure 1. Ideal Specification in XML

Our system provides great flexibility in the generation of training data. We use Weka's implementation of the C4.5 classifier which outputs a decision tree. It also gives an analysis of the resultant tree's performance against the training and test data. The derived decision tree matched the model of the candidate selection criteria and when applied to the training data, it correctly classified 100% entries. Another test of the classifier was run against simulated data and correctly classified all the components and selected 27 out of 2000 components as potential candidates.

We then ran the trained classifier over real component data where it identified 17 suitable components for the

578 that were considered. Although the four repositories offered over 86,000 entries, we worked with a subset of those matching the search criterion "calculator" as manual conversion of all entries to XML was impractical. Incorrect results were given for less than 7% of the data, in situations where values for attributes were missing. Classification of missing data is one of the limitations of C4.5. If it has not seen a particular value for an attribute, it will still try to classify the instance according to its decision tree, with unpredictable results. In our data, missing information was replaced with "-" for text attributes and -1 or 1000 for numeric attributes. We are investigating the substitution of average or default values for missing values, as well as alternate Machine Learning approaches to improve the handling of missing data.

At this point, we can consider updating or tuning the ideal specification. Using the facilities provided by Weka, we can look at the component data as individual attributes or as groups of attributes. Statistical information about individual attributes helps us to adjust ranges for numeric values. Clustering tools help us to find components that have a similar profile to our ideal specification. We can then adjust the ideal specification, retrain the classifier and re-run the component data to get a tighter match on suitable components.

5. Context-driven Component Evaluation[2]

This work is part of the CdCE Project. The Project aims to develop strategies for the assessment of software components, both through static comparison of developer requirements to a candidate component specification and by generating context-driven tests for the dynamic assessment of short-listed components. We address the issues of sourcing, selection and evaluation of software components, with indirect benefits in testing and trust. The process is driven by a specification of the ideal component and its operating context, which provides a foundation for the automation of the selection process. We focus on the selection of third party components from commercial and open source brokers and repositories where the format and detail of component documentation can vary widely.

An important attribute of third party software components is that they are written for the general case. They then require contextual information and testing to fully evaluate their suitability to an application [16]. The developer needs to know that the component is not only reliable and meets its specification, but that it is suited to the target system. Component certification can improve confidence and trust, but is not sufficient reason for a particular selection as it does not take context into account and cannot ensure that a component will behave correctly in another environment [17]. Our ideal component specification includes details of the requirements for the component and aspects of the target system to allow a context-aware evaluation of a component's suitability.

Figure 2 shows our process for component evaluation. In the first step we define the requirements which become the ideal component specification. The ideal component is specified on two levels, metadata for descriptive information, and a formal specification of the interfaces and behaviour in Z notation. Step 2 searches for candidates matching the ideal specification, resulting in a short-list for further examination. Abstract test cases are generated for the components in Step 3, based on the formal specification of the ideal component. The tests can also be used for system and regression testing. An adaptation model is developed for each candidate in Step 4 and used to adapt the abstract test cases to match each of the short-listed candidates.

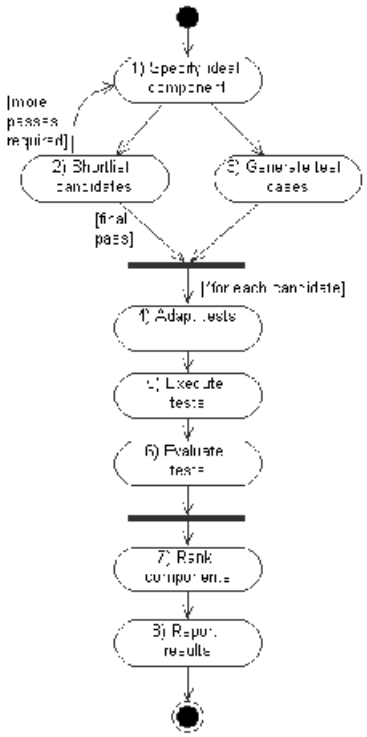

Figure 2. Activity diagram for CdCE Process

The tests are executed against the candidates in Step 5, and the test and short-listing results are combined in Step 6 to get an overall picture for each component. In Step 7 we look at the results across the candidate components to generate a comparison. This may involve aggregation for scores against criteria, or other methods such as the C4.5 classifier described in this paper. We can then move to Step 8 and provide a recommendation for component

[2] Formerly known as the Context-driven Component Testing (CdCT) project

selection, including reasons behind the recommendation, and information to assist in adapting and integrating the component. A more detailed description of the process appears in [18] and [14]. We are currently developing a tool to assist developers through the CdCE process, linking to classification and test generation software and compiling the results of each step for generation of the recommendation(s) in Step 8.

6. Conclusion

We have explored the use of Machine Learning algorithms for the selection of software components. Our case study results show promise, with the generated data training the C4.5 classifier and providing an appropriate decision tree. It then gave correct classification for all candidate components (in minutes) compared with a manual approach which missed some candidates and took over eight hours. Our training data generation overcomes a major issue with supervised Machine Learning in that it does not require large amounts of historical or statistical data as we generate the training data and labels (accept/reject) from a model. We also address the problems of aggregation-based component selection approaches where the relationships between components are lost.

Machine Learning is not normally economical for one-off classification problems. Each new search for a component is a new problem with different selection criteria. Our approach works from the ideal specification, which is always necessary for component selection. We automate the training data generation from the ideal specification using generic techniques and can easily train for the selection task at hand. The result is a considerable automation of the selection process requiring a small amount of expert time. We are currently applying this to the short-listing or filtering stage of component selection, but it can also be used for the more technical evaluation required later in the selection process (Step 7 of CdCE Process). The Machine Learning tools can also be used to adjust or tune the ideal specification based on statistical and clustering information.

This work is one way that AI can be applied to benefit those in the computing community. We plan to extend this work by investigating and evaluating other classifiers and Neural Networks to further utilise the generated training data for supervised learning of component selection criteria. We are also looking at improving the data representation so that more information can be fed back into the process via clustering and other learning techniques.

References

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

[2] T. L. Saaty, “The Analytical Hierarchy Process”, McGraw-Hill, 1990

[3] S. Sassi, L. Jilani and H. Ghezala, “COTS Characterization Model in a COTS-based Development Environment”, Asia-Pacific Software Engineering Conference (APSEC), Chiang Mai, Thailand, 10-12 December, 2003

[4] J. Kontio, “OTSO: A Systematic Process for Reusable Software Component Selection”. Tech. Report UMIACS-TR-95-63, University of Maryland, 1995.

[5] M. Ochs, D. Pfahl, G. Chrobok-Diening and Nothhelfer-Kolb, “A Method for Effective Measurement-based COTS Assessment and Selection – Method Description and Evaluation Results”. Tech. Report IESE-055.00/E, Fraunhofer IESE, 2000.

[6] M. Morisio and A. Tsoukis, “IusWare: a Methodology for the Evaluation and Selection of Software Products”, IEEE Proceedings of Software Engineering, Vol. 144(3), pp. 162-174, June 1997.

[7] Y.-H. Kuo, J.-P. Hsu and M.-F. Horng, “Neuro-fuzzy Based Search Robot for Software Components”, International Journal on Artificial Intelligence Tools, Vol.8(2), pp. 119-135, 1999.

[8] D.V. Rao and V.V.S. Sarma, “A Rough:Fuzzy Approach for Retrieval of Candidate Components for Software Reuse”, Pattern Recognition Letters, 26(6), March, 2003.

[9] 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.

[10] M. Negnevitsky, “Artificial Intelligence:A Guide to Intelligent Systems”, Addison Wesley, 2002

[11] J. Ross Quinlan, “C4.5: programs for machine learning”, Morgan Kaufmann Publishers Inc., CA, 1993

[12] World Wide Web Consortium. "XML Schema" [Web page]. Accessed 20/12/2003, from the WWW: http://www.w3.org/XML/Schema, 2003.

[13] Dublin Core Metadata Initiative. "DCMI Website." Accessed 20/12/02, from the World Wide Web: http://www.dublincore.org/, 2002

[14] V. Maxville. “Context-driven Component Testing Project Website” [web page]. Accessed 6/9/03, from the WWW: http://www.scis.ecu.edu.au/research/PhD/vmaxvill/, 2003

[15] I. Witten and E. Frank, “Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations”, Morgan Kaufmann Publishers, 2000.

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

[17] 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 Heidelberg2003.

[18] Maxville, V., Lam, C. P. and J. Armarego “Selecting Components: a Process for Context-Driven Evaluation”, Asia-Pacific Software Engineering Conference (APSEC), Chiang Mai, Thailand, 10-12 December, 2003

This page left intentionally blank.