2008

Supporting Component Selection with a Suite of Classifiers

Valerie Maxville

Chiou Peng Lam
Edith Cowan University

J Armarego

10.1109/CEC.2008.4631334

This article was originally published as: Maxville, V. R., Lam, C. P., & Armarego, J. (2008). Supporting Component Selection with a Suite of Classifiers. Proceedings of IEEE World Congress on Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). (pp. 3946-3953). Hong Kong. IEEE Explore. Original article available here

Supporting Component Selection with a Suite of Classifiers

Valerie Maxville, Chiou Peng Lam and Jocelyn Armarego

Abstract— **Software selection involves the assimilation of information and results for each candidate to enable a comparison for decisions to be made. The processes and tools developed assist with software selection to enhance quality, documentation and repeatability. The CdCE process aims to retain and document the information used in selection to assist decisions and to document them for reference as the system evolves. This paper describes the CdCE process and our approach to assist the shortlisting of candidates through a suite of classifiers. The application of the suite is illustrated using a selection and evaluation case study. Applying this approach helps retain the multidimensional nature of the selection process and enhances user awareness in the decision making process.**

I. Introduction

Increasingly, component-based software development (CBSD) is becoming an alternative approach to developing software system: the associated benefits include reduced development cost and effort as well as improved quality of the software system. CBSD involves a plug and play framework where commercial off-the-shelf components (COTS) are integrated to provide the system functionalities. The Component-Based Enterprise Software Process Model [1] for CBSD consists of the following sequential stages: Analysis and Component Acquisition, Component-Oriented Design, Component Composition, Integration Test, System Test. In recent years, various approaches have been developed for the component acquisition phase involving COTS selection. Amongst these are REACT [2], PECA [3], RCPEP [4], PORE [5], OTSO [6], SCARLET [7], EPIC [8], CISD [9] and IusWare [10]. It is widely accepted that selection of the most appropriate COTS is a key aspect in successfully developing the software system in CBSD.

As the use of third party software increases, the common task in the Analysis and Component Acquisition phase of the CBSD life cycle is the selection of software or components to suit developer requirements. Approaches to the selection of COTS software may consider functional and non-functional characteristics, be based on quality models or on the dynamic evaluation of the available software [11]. Maintenance of component-based systems has also becomes more complex with increased vendor reliance, leaving developers vulnerable to their external changes including software updates and retraction of support for old versions. A project may need one or many pieces of third party software, and the evolution of the system may force the acquisition of software to replace or extend their functionality. The approaches to COTS selection vary, with proofs and logic being applied where software has detailed specifications [12] while others use a manual assessment against criteria [13]. Much of the effort in the selection processes is manual, labour-intensive and subjective, reducing repeatability and scalability. In addition, this is an ongoing, iterative process due to internal and external forces associated with the evolution of the components.

Component candidate selection applies to sourcing components in a new development as well as in the evolution of existing software packages in terms of the combinations of components to be included in the next release. It involves querying component brokers and repositories (external or in-house) to seek out likely matches to requirements. With repositories in a component market place storing tens of thousands of software components and applications (e.g. Freshmeat.net [14], Component Source [15]) and massive overheads involved in any in-depth evaluation of components, users need automated techniques and tools to manage the selection process as well as to ensure that the components on their short-list for further evaluation is likely to be relevant, and that suitable items have not been missed. Ideally, component brokers and repositories should adopt a standardised template for documenting their components to support the automation of component assessment. This would allow the harvesting and processing of component information using Internet agents. However, in reality, raw data from these repositories have their own data model and format. For example, Freshmeat.net has 38 attributes in a DTD and the data is available as RDF files. SourceForge [16] has a relational database or free text summary files. This implies that each repository's template has to be transformed before subsequent processing.

The Context-driven Component Evaluation (CdCE) project [17] has developed a process for component selection which can also be applied to COTS and OSS (Open Source Software) selection. One of its aims is to implement and assess strategies to enhance and automate the component selection process. To facilitate this, one output of CdCE is the development of a standardised component specification template, consisting of 37 attributes described in an XML schema. Instances of this schema form the basis for an ideal component specification which may be created to describe single or multiple components. The work described in this paper focuses on the filtering (short listing) stage of selection (Step 2 in the CdCE process illustrated in Figure 1), involving a specification for a desired component and the raw data from a repository as input to output a short-list of candidates. Unlike alternate approaches in this project that had formulated component candidate selection as a classification problem [18], the problem of short-listing potential candidate components is recast here as a search-based problem. The paper describes an approach which can be incorporated into the CdCE process for short-listing components in the COTS selection phase. The aim here is to demonstrate that the search-based approach originally proposed by Bagnall [19]

978-1-4244-1823-7/08/$25.00

in selecting components for the next release problem is also suitable for shortlisting candidate components as part of the component acquisition phase in CBSD. This approach is evaluated using the Freshmeat.net repository (with 41,885 entries for software components). The significance of the approach is that it provides an automatic progression from a specification for a desired component and the raw data from a repository and generates a suite of classifiers, each of which provides a short-list of candidates. The user can make an informed decision in selecting the short-list(s), thus providing a technique to improve recall and relevance in the COTS selection process.

The following section describes our project and the CdCE Process for component selection. Section III gives details of how the new approach assists the user in selecting shortlists of candidates and some discussion on the interpretation of the criteria graphs. We then show the process and techniques applied to a real-world case study (Section IV) and discuss related work in Section V. We conclude with a discussion of the effectiveness of the approach and future work for the project.

II. Context-driven Component Evaluation

The CdCE Project has built on the premise of providing a usable, repeatable process with tool support. We use a fixed template of attributes that can be tailored via the ideal specification - the requirements for the component in XML and Z notation. A difficulty in this field is that there is no standard specification for COTS or components. We have considered the characterisations proposed by the research community, along with what is used in commercial and Open Source repositories and included necessary and objective attibutes in the CdCE specification template. This has been successfully applied in the transformation of a repository of over 41,000 items into CdCE format, and in a number of software selection case studies.

The ideal specification is used as a model to generate training data for two machine learning classifiers to shortlist and later to evaluate the available components. Each of these classifiers may be reused in subsequent selection tasks. The **non-functional** attributes (e.g. development status and operating system) are used to shortlist the components. The smaller set of candidates then go through a **functional** evaluation based on tests generated from the formal specification of required behaviour. These tests, with adaptation, can be reused to assess newly located components as the system evolves, allowing valid comparisons with those previously considered. The results of the tests are compiled, with the resulting scores against metrics becoming the input to the second classifier which evaluates the short-listed components and provides the recommendation for selection.

A. CdCE Process

The CdCE Process comprises eight steps (Figure 1), to provide structure and repeatability in COTS and component selection. The process is driven by an ideal specification in XML, which includes the contextual information required to recognise a suitable candidate. Each of the steps is linked through XML files to provide traceability throughout the process.

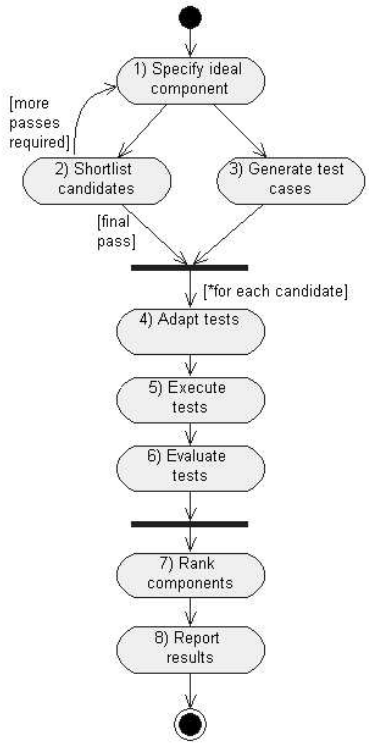

Fig. 1. The CdCE Process for Component Selection

The ideal specification is created in **Step 1** and uses XML and Z notation to describe the functional and non-functional characteristics of the ideal component. We consider testing, **in context**, to be essential to the evaluation process and require a Z notation specification of the key behavioural aspects of the required software to allow automated test and test data generation. Context testing that is included in the CdCE process are performance (CX_P), reliability (CX_R), stress (CX_S), and usage (CX_U). The ideal specification also requires values to be entered for the evaluation metrics listed in Table I on a scale of 0-10, 10 representing full success. In addition to the context metrics, we have metrics for functional fit (FFIT), functional excess (FEXS), adaptation effort (AEFT), test fit (TFIT) and test results (TRES).

The ideal specification is used to generate a classifier for shortlisting in **Step 2**. We use the C4.5 algorithm within the Weka machine learning tools [20] to generate the classifiers as it provides human-readable decision trees. The repository data is converted to CdCE format and a selection of transformations is provided to improve the relevance and recall of results. The converted data is classified and if the resultant shortlist is unsatisfactory, we iterate to Step 1 to refine (loosen, tighten, alter) the ideal specification. The first

TABLE I
EVALUATION METRICS

Metric	Calculation
FFIT Functional Fit	(# interfaces matched /# interfaces required)
FEXS Functional Excess	(# interfaces matched /# interfaces in component)
AEFT Adaptation Effort	(# interfaces adapted /# interfaces required)
TFIT Testing Fit	(# tests possible /# test cases)
TRES Test Result	(# tests passed /# test cases)
CX_P Performance testing result	(# performance tests passed /# performance test cases)
CX_R Reliability testing result	(# reliability tests passed /# reliability test cases)
CX_S Stress testing result	(# stress tests passed /# stress test cases)
CX_U Usage testing result	(# usage tests passed /# usage test cases)

pass of the shortlisting uses the full specification to allow rough tuning of the selection criteria, particularly those that may lock out all candidates. The Weka system is used to view the statistics across the input data set to highlight issues (e.g. missing data) and create a usable base ideal specification, which we refer to as S10. From this ideal specification we identify the mandatory and non-mandatory criteria, with the flexibility in criteria made possible by non-mandatory criteria. As the refining process has been automated, we can now create a graph of the possible combinations of criteria by dropping one non-mandatory criterion at a time (Fig. 2) and the number of candidates that result from each set. This graph can be traversed to find an optimal criteria set to identify a shortlist of components. The search-based approach condenses the iteration process, removes some of the heuristics and subjectivity previously used, and provides a better overall view of the impact of including or excluding each criterion.

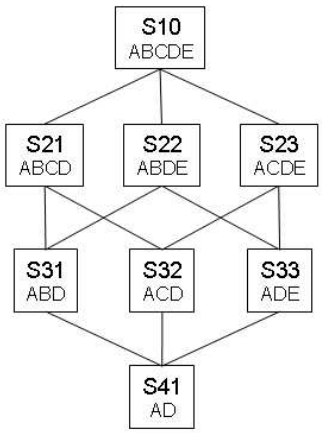

Fig. 2. Graph representing a series of sets, classifiers and subsequent shortlists. $\{A, D\}$ are mandatory and $\{B, C, E\}$ are non-mandatory.

In **Step 3** we generate tests from the formal specification. We use equivalence partitioning to generate the test cases and substitute supplied test data into the abstract test cases, which could be enhanced in the future by including the Classification Tree Method [21]. These abstract tests are adapted to the candidates in **Step 4**, which also produces input for calculating metrics for functional fit and adaptation effort (FFIT, FEXS, AEFT and TFIT in Table I). The tests are provided as abstract XML and must be ported to the local harness/environment, then executed in **Step 5**.

The results of the tests are evaluated in **Step 6** to calculate the remaining five metrics where appropriate - test result (TRES) plus the context metrics CX_P, CX_R, CX_S, and CX_U. The developer's preferences for these metrics are gathered from the ideal specification and a second classifier is generated in **Step 7**. As in Step 2, we can create a search graph or iterate between: modifying the ideal specification, regenerating the classifier and processing the data to produce a satisfactory shortlist. If issues with the functional specification arise, it can also be adjusted, requiring Step 3 onwards to be repeated. The automation within the process, and the ability to adapt to the available software, reduce the additional effort required for iteration.

Step 8 collates the data from the entire process to provide a report including the justification for the choices made (e.g. decision trees). It is possible that no suitable component is found, which would also need to be documented. A description of the original CdCE process and implementation appears in [22].

The CdCE ideal specification includes 37 attributes of five types: `numeric`, `date`, `text`, `longText` and `ontology`. Each are processed throughout the CdCE tools to maximise information utilisation. An example is in the `ontology` attributes, where we have adopted the Freshmeat trove as the ontology. This provides an established hierarchy to which we have added a thesaurus and distance matrix to capture 'close' values, e.g. flavours of Linux. Other attributes are more straightforward in their implementation.

Investigation of data representation also led to a range of transformations to allow tuning of the matching mechanism. The transformations range from T1 - equivalent to a boolean or SQL matching query, to T5 - utilising distance metrics and abstraction of the ontology hierarchies. More detail on the knowledge representation aspects of the CdCE project are described in [23]. Case studies have indicated that the T5 transformation to be the most effective in identifying candidates and this is what we use in this paper's case study.

The process allows for flexibility in its application to suit organisational requirements. Alternatives to the C4.5 classifier can be substituted by creating corresponding output methods in the Generator and Transformer classes (Java classes developed for this project). The input data (Freshmeat), ontology and thesaurus can be substituted by changing parameter files and adhering to the respective XML schema. Alternative test generation techniques can also be applied within the CdCE process.

III. The Classifier Suite

When considering the shortlisting step, the CdCE process allows for iteration between the specification of the ideal component and the shortlisting of candidates to tune the specification by loosening or tightening criteria. In practice, the first iteration uses the most strict set of criteria which are loosened iteratively to find a suitable shortlist. This leads to an effective process but is subjective in choosing the next criterion to loosen. As the tools for generating the classifiers and extracting the shortlists have been automated, a more informed approach is considered where all possible sets of criteria are determined and the shortlists made available to the user via a suite of classifiers.

During the specification step, the user can flag certain criteria as mandatory, e.g. the description or the development language. Holding these criteria fixed, they can progressively loosen the criteria by dropping one at a time out of the selection set. For example, working with criteria $\{A, B, C, D, E\}$ they may decide that the mandatory set is $\{A, D\}$ and the non-mandatory are $\{B, C, E\}$. The resulting possible sets of criteria are:

$$\{ABCDE\}, \{ABCD\}, \{ABDE\}, \{ACDE\},$$
$$\{ABD\}, \{ACD\}, \{ADE\}, \{AD\}$$

This can be represented as a graph (Fig. 2) with each node representing a selection set and each edge the removal of one of the criteria. After the respective classifiers have processed the repository data, the user can then view the graph marked up with the number of items in each shortlist and the respective criteria to help to select one or more sets for their shortlist. The metadata for the software repository is available for each candidate on each shortlist which can be viewed to help the user's decision.

In terms of the processing of the repository data, in the past we have had many iterations to create a satisfactory shortlist, with no awareness of the untried criteria sets. In the new approach, we can get an overall picture of the variations, drill down as required and still have the option of returning to the specification to make changes before a second iteration. The case study in section IV shows the application of the new approach.

A. Scalability

The height of the graph is dependent on the number of non-mandatory criteria. Each level of the tree progressively removes one criterion from the set, starting with the full set at the top of the graph and finishing with the mandatory set at the bottom. The number of sets and classifiers in each level of the graph is:

$$N = setofnon-mandatoryelements$$
$$Number\ of\ non-mandatory\ elements = n$$

$$Graph\ height = n+1$$
$$Number\ of\ classifiers/sets\ required = 2^n$$

Note that the number of mandatory criteria, m, does not affect the size of the graph.

B. Interpreting the Graph

We are currently exploring ways to represent the graph effectively for the user to be able to have a more intuitive view of the possible shortlists. Currently we list the criteria included in the set, represented by letters (e.g. A-E), along with the number of items in the shortlist for that set. This works well for a small graph. However, where the number of non-mandatory items goes above four or five, the graph can become difficult to interpret. One way to reduce graph size is to split it on a well-understood criterion, e.g. date updated - F in Fig. 3. The user can then consider which criteria are important with, and in the absence of, the split criterion.

Another approach is to highlight nodes with criteria of interest. In the case study, criterion A was of particular interest as it caused dramatic changes in shortlist sizes. In Fig. 4 we highlight the sets/nodes that include criterion A, and the edge which links to the equivalent node without A. The difference between the numbers returned in the shortlists with and without A are listed below:

$$Differences\ resulting\ from\ criterion\ A:$$
$$\{A,B,C,D,E,F\} = 1 \ldots \{B,C,D,E,F\} = 7$$
$$\{A,B,C,D,E\} = 1 \ldots \{B,C,D,E\} = 6$$
$$\{A,B,C,E,F\} = 10 \ldots \{B,C,E,F\} = 24$$
$$\{A,B,D,E,F\} = 4 \ldots \{B,D,E,F\} = 10$$
$$\{A,B,C,E\} = 13 \ldots \{B,C,E\} = 27$$
$$\{A,B,D,E\} = 4 \ldots \{B,D,E\} = 14$$
$$\{A,B,E,F\} = 10 \ldots \{B,E,F\} = 23$$
$$\{A,B,E\} = 12 \ldots \{B,E\} = 37$$

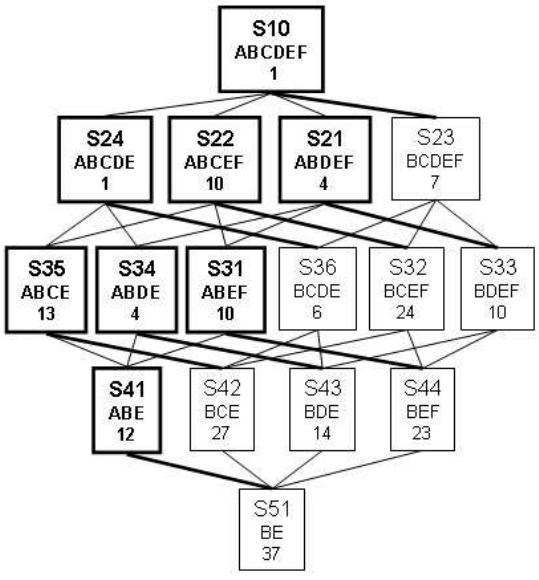

Fig. 4. Graph representation of criteria sets, highlighting criterion A

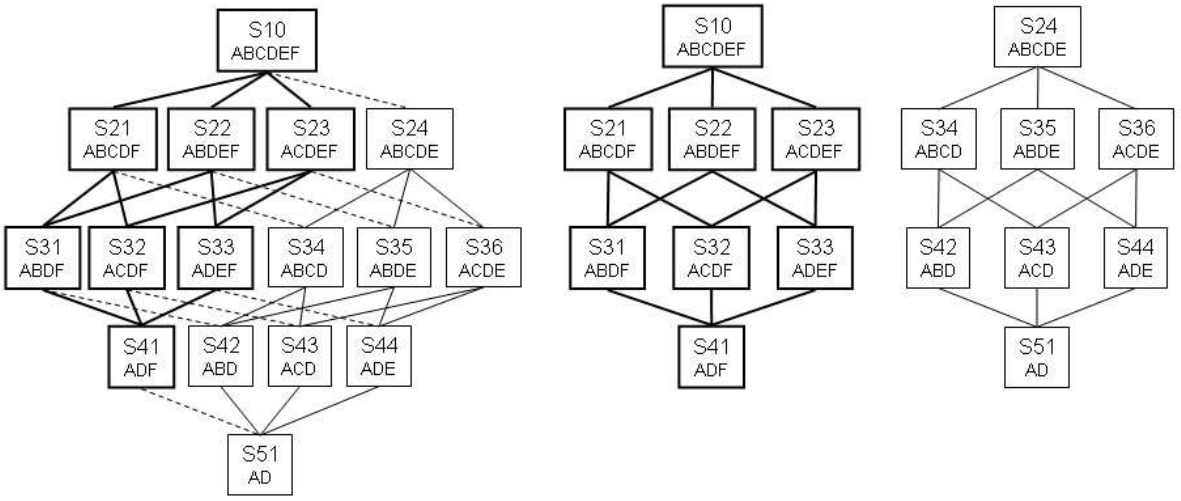

Fig. 3. Splitting the graph on criterion F

These patterns of differences are significant in the choice of criteria. To determine if a criterion is blocking (bad) or filtering (helpful), we need to drill down into the shortlist. An example from our case study showed that description (B) had this effect. As we were also using detail (detailed description - C) it could be that the attributes were redundant or one of them was rarely matched, that both were required, or that we needed one and not the other. Drilling into the shortlists we found that those that did not include 'description' were almost completely irrelevant, whereas those with description were almost entirely relevant. This knowledge gave the added benefit of adding another criterion to the mandatory set, halving the graph size. Such patterns are always of value, but would have different impacts depending on the data.

C. Advantages and Disadvantages

Advantages of using the classifier suite include increased understanding of the options for varying the selection criteria, the low overhead of generating the criteria sets, classifiers and shortlists and the ability to make selection more visual and easier for the user to explore. The alternate approach of iterative refinement was subjective in the choice of criteria to loosen/tighten and risked missing criteria sets of interest unless an exhaustive iteration of all permutations was done.

Two main issues exist, the number of classifiers (and processing) required and the complexity of the data being interpreted. The size of the suite is dependent on the number of non-mandatory criteria, with the graph size equal to 2^n. There could be many improvements to the code for processing the data, which has been written in strict object-oriented Java. Even so, the processing time averages 2 minutes per criteria set - 30 minutes for four non-mandatory criteria and one hour for five criteria. The time taken to generate classifiers and shortlists is affected by the attribute types and the selected transformation (described in Section II.A). The CdCE tools are currently written in Java and are not optimised, and take an average of two minutes per set on a 2 GHz Pentium with 767 Mb RAM. In practise, the limitation for the size of the graph may be in the user's ability to interpret it than in the processing time required.

We have also shown ways to divide the graph for interpretation and to isolate criteria which may be able to be removed or shifted to 'mandatory'. Each reduction in the size of the non-mandatory set halves the size of the graph (and processing time and complexity). We see the the classifier suite as reducing subjective choices and improving awareness of a solution space which was already of the given size, but previously had no tools to assist the user to make decisions.

IV. Case Study

As a simple example of the use of the process and classifier suite generation, we consider the selection of COTS software to provide **XML editor** functionality. In **Step 1** of the CdCE Process, we define the requirements by creating an ideal specification. Figure 5 gives the ideal specification used in this case study along with the letters we use to represent them in this paper. We are sourcing XML editor software (Criteria A & B), produced in Java (E), to run on a Linux platform (F). In keeping with the Open Source nature of the particular development, we would like a GNU General Public License (C) and the product must be Mature (D). Other requirements are less specific, so we have set reasonable limits on the memory and disk requirements and how recently the software was updated (G). These ranges can be tuned through iteration. Specification can be time-consuming, so we note that Step 1 took approximately four hours in this case study.

To begin the shortlisting in **Step 2**, we use the full, most restrictive ideal specification and all attributes set to mandatory. We choose data Transformation 5 (see Section II.A)

```
A - Non-mandatory
B - Mandatory
C - Non-mandatory
D - Non-mandatory
G - Non-mandatory

E - Mandatory
F - Non-mandatory

    Missing
    Missing
```

```
<?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">XML editor</dc:description>
       <dc:detail type="mandatory">XML editor</dc:detail>
       <swv:licence type="mandatory">GNU General Public License (GPL)</swv:licence>
       <swv:devStatus type="mandatory">6 - Mature</swv:devStatus>
       <dc:date type="mandatory" min="01-01-2005" max="31-12-2007">31-12-2007</dc:date>
       <swv:technical>
           <swv:devLanguage type="mandatory">Java</swv:devLanguage>
           <swv:operatingSystem type="mandatory">Linux</swv:operatingSystem>
           <swv:systemRequirements>
              <swv:memory type="mandatory" min="15" max="50">20</swv:memory>
              <swv:diskSpace type="mandatory" min="30" max="50">40</swv:diskSpace>
           </swv:systemRequirements>
           <swv:Zspec>... removed for space reasons ...</swv:Zspec>
           <swv:Zcontext>... removed for space reasons ...</swv:Zcontext>
       </swv:technical>
       <swv:FFIT type="mandatory" min="8" max="10">10</swv:FFIT>
       <swv:FEXS type="mandatory" min="8" max="10">10</swv:FEXS>
       <swv:AEFT type="mandatory" min="6" max="10">10</swv:AEFT>
       <swv:TFIT type="mandatory" min="6" max="10">10</swv:TFIT>
       <swv:TRES type="mandatory" min="10" max="10">10</swv:TRES>
       <swv:CX_U type="mandatory" min="8" max="10">10</swv:CX_U>
</Description>
```

Fig. 5. Ideal specification for XML editor case study

from our available transformations as previous investigations have indicated it provides improved short-lists through the use of distance measures and ontologies for abstraction of some text attributes.

After generating training data for the classifier, the repository data is transformed and classified, resulting in an empty short-list. Information provided by the Weka statistical tools shows that there is a high level of missing data on the `memory` and `diskSpace` attributes. We remove these from the ideal specification and and flag two criteria as mandatory - `detail` and `devLanguage`.

At this point we take advantage of the automation of the CdCE Process and generate a suite of classifiers for the combinations of sets of criteria. In the graph (Fig. 6) we have overlayed the `date` criterion as a second value in each node. The level 1 node - S10 has no items in its shortlist. As the aim is to gain the highest relevance of results while not having too many candidates on the shortlist, we look for larger lists by loosening criteria. In levels 2 and 3, we have a few results below ten, which is manageable. By level 4 the relevance of candidates will be reduced.

As described in Section III.B, by drilling down to the metadata we found that the `description` criterion should be mandatory as lists without it have high numbers of irrelevant items. This allows the removal of eight nodes: S23, S32, S33, S36, S42, S43, S44 and S51 from the graph to be considered. We find that two nodes: S21 and S34 are identical, which also helps to reduce options. We can choose S21 without `date` and have four items on the list. Previous case studies with the Freshmeat dataset have shown that `date` and `devStatus` are important for good projects. As we have lost `date` on the choice of S21, we look for nodes with this criterion along with as many others as possible (the right number of the graph). S22 and S24 have 2 and 0 items respectively, so we are left to choose from S31 and S35. As S31 is closely related to S21, we aim for variety and choose S35 which has four items. We accept two four item short-lists - S21-date, S35+date, giving a total of eight candidates with these changes.

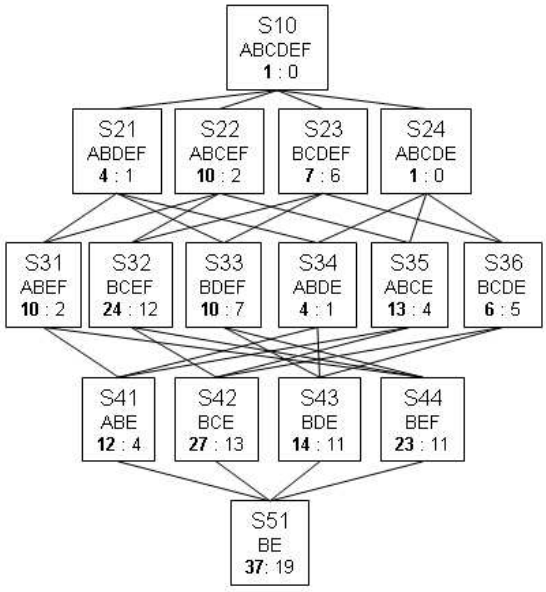

Fig. 6. Graph representation of case study shortlists. Date (G) is overlayed on the graph as the lower of the two shortlist counts.

Step 3 of the process takes the behavioural specification in Z notation and generates tests. The XML editor test suite has 12 test cases, concentrating on loading and validating XML documents and taking in test data (XML files) from the specification. To make the recording of results consistent, there is an XSLT transformation which converts the XML test cases in HTML forms for the user to record the results for each candidate. In **Step 4** each shortlisted candidate

is downloaded and the differences between the available and required interfaces are documented. This is currently a manual process. During the adaptation, we derive values for related metrics: Functional Fit (FFIT), Functional Excess (FEXS), Adaptation Effort (AEFT) and Testing Fit (TFIT). These are shown in columns 1-4 of Table II. Candidate C8 was not available for download as it has become a commercial product with no trial version.

TABLE II
COMPLETE EVALUATION FOR SHORTLISTED CANDIDATES

Candidate	FFIT	FEXS	AEFT	TFIT	TRES	CX_U
C1	6	10	8	3	2	4
C2	8	10	10	3	0	0
C3	4	6	10	3	3	5
C4	10	6	4	10	10	10
C5	8	10	8	9	5	8
C6	10	6	4	10	10	10
C7	8	10	4	9	5	10
C8	0	0	0	0	0	0

The adaptation documentation for each candidate is combined with the abstract tests and transformed into executable tests in **Step 5**. In this case study the tests are manually executed and results recorded in HTML forms. Each candidate is tested, and the results of the tests are collated in **Step 6** (see Table III), generating metrics for TRES, TFIT and the contextual test metrics (if applicable). In this case study we have usage-based tests, so we generate a score for CX_U and no value for CX_S, CX_R and CX_P (not shown in table). This completes the metrics for the candidates and we add them to the XML file for the candidates in **Step 6**. The full results are in Table II with 10 being a perfect result.

TABLE III
TEST RESULTS FOR SHORTLISTED CANDIDATES

Candidate	NumTests	NumSkipped	NumPassed
C1	12	9	2
C2	12	9	0
C3	12	9	3
C4	12	0	12
C5	12	3	5
C6	12	0	12
C7	12	3	6
C8	0	12	0

The ideal specification is again used for the evaluation of the components in **Step 7**, this time concentrating on the nine functional metrics. The evaluation classifier is generated in the same way as the short-listing classifier, and the candidates are processed. As in Step 2, we may wish to iterate and modify the ideal specification to provide an acceptable list or selection. The recommended action in each iteration is to adjust the values for each of the metrics, as opposed to the dropping of criteria that was done in Step 2. In this case, there were no candidates who achieved the metrics as initially listed. The final values used for the evaluation metrics in this case study were as listed in Table IV.

TABLE IV
REQUIRED VALUES FOR THE EVALUATION METRICS

Iteration	FFIT	FEXS	AEFT	TFIT	TRES	CX_U
Initial values	8	8	6	6	10	8
Final values	8	6	5	6	8	8

After the second pass with the final values, there are two clear recommendations - candidates C4 and C6. Next best were C5 and C7, with the other candidates performing poorly and not being recommended. A possible issue with the top two is that they do not have GPL or equivalent licencing. The software, released by the same company, have commercial (with free trial) and academic licences respectively. Where this is prohibitive, the next ranked candidates would be considered. With the ranking complete, the infomation for the whole evaluation is compiled into a report (**Step 8**) including recommendations along with electronic versions of all artifacts including specifications, classifiers and tests.

V. RELATED WORK

A key issue in Search-based Software Engineering is the need to recast software engineering problems into search-based problems. In structural testing, the problem of test data generation has been recast as a problem of searching for test inputs which will satisfy a specified test adequacy criterion. For example, Harman et al presented a testability transform that transform a code based program with one or more exit statements into a structured branch-covering equivalent program [24]. Antoniol et al [25] have reformulated the problem of project planning in software maintenance as one of task scheduling. Others like Xiao et al [26] have transformed the problem of the test sequence generation based on the Wp method into one of finding the shortest path in the asymmetric travelling salesman problem.

Bagnall et al [19] formulated the next release problem as a variant of the 0-1 knapsack problem where by the objective is to find a set of customers requirements, that falls within a specified budget, which is to be included in the next release. In order to solve this optimisation model, Bagnall et al applied a number of different techniques, including integer programming, greedy algorithms and simulated annealing to data generated from 5 randomly generated artificial projects. Baker et al [27] incorporated concepts first presented by Bagnall et al and described a search-based approach for selection and prioritisation of components in the next release problem using real-world data. The problem was also formulated as a feature subset selection problem; specifically as an optimisation 0-1 knapsack problem with a specified budget K, as the bounding constraint. Simulated annealing and greedy algorithms were used in their approach and it was evaluated using a dataset comprising of 40 candidate components from a telecommunication company. The results obtained from the applications of both greedy and simulated annealing algorithms were compared to those of an expert and was shown to be more superior.

Maxville et al [18] have described some work for shortlisting components that involved the use of Artificial Intelligence (AI) techniques to classify components based on a specified component specification. This specification was used to generate training data which was then used to train a C4.5 classifier and neural network classifier respectively to recognise suitable components. These classifiers were then subsequently used to classify test data from a given component repository in terms of their suitability in the short listing phase of the CdCE process. If the specification of the required component remains unchanged the classifier, once trained, can be used repeatedly to classify new test data as they arise or in another iteration of COTS selection as the product evolved. Positive results from this work led to the automation of the generation of a suite of classifiers representing various combinations of selection criteria. By viewing the criteria, shortlist size and drilling down in to the metadata for candidates on the shortlists, the user can see the impact of their choice of selection criteria and be more confident in the resultant shortlist. This empowers the user and gives more dimension to the data they base their decisions on, as well as making the process less subjective.

VI. Conclusion

The CdCE project has focussed on process development with tool support for the selection of COTS software and components. The aim was for a practical, usable process which would encourage repeatability and documentation in the selection process. The CdCE process uses iteration in deriving a shortlist of candidates, which has tool support for generation of classifiers and the extraction of the metadata shortlists. We have used these tools to modify the process to consider all the possible sets of criteria and generate a suite of classifiers to provide a formation of shortlists for the user to view. As the generation is automated, the overhead is minor but provides more information for the user to decide which criteria (and shortlist) to use. A case study using the process to source an XML editor was shown.

Future work will include improved visualisation of the criteria sets and graph. We will also be providing guidelines for interpreting the patterns and data included in the graphical and other representations of the data. Although the approach has been developed for software selection, it could be applied to dealing with more general multi-criteria selection problems in other domains.

References

[1] M. Aoyama, New Age of Software Development: How Component-Based Software Engineering Changes the Way of Software Development, Proc. 1998 International Workshop on Component-Based Software Engineering, In conjunction with ICSE '98, Apr. 1998, Kyoto.

[2] V. Sai, X. Franch, N. A. M. Maiden: Driving Component Selection through Actor-Oriented Models and Use Cases. ICCBSS 2004: 63-73

[3] S. Comella-Dorda, J. Dean, E. Morris, and P. Oberndorf, A process for COTS software product evaluation. Proceedings of the 1st International Conference on COTS-Based Software System, Orlando, FL. February 4-6, 2002. pp. 86-96. NRC 44925. (LNCS 2255, Springer).

[4] P. K. Lawlis, K. E. Mark, D. A. Thomas, T. Courtheyn: A Formal Process for Evaluating COTS Software Products. IEEE Computer 34(5): 58-63 (2001)

[5] C. Ncube, N. A. Maiden. PORE: Procurement-Oriented Requirements Engineering Method for the Component Based Systems Engineering Development Paradigm". In proceedings of 1999 International Workshop on Component Based Software Engineering.

[6] J. Kontio. OTSO: A Systematic Process for Reusable Component Selection. University of Maryland, CS-TR-3478, 1995.

[7] N. A. M. Maiden, V. Croce, H. Kim, G. Sajeva, S. Topuzidou. SCARLET: Integrated Process and Tool Support for Selecting Software Components. Component-Based Software Quality 2003: 85-98

[8] C. Albert and L. Brownsword. Evolutionary Process for Integrating COTS-Based Systems (EPIC) Building, Fielding, and Supporting Commercial-off-the-Shelf (COTS) Based Solutions, CMU/SEI-2002.

[9] V. Tran, D. Liu and B. Hummel. Component-based systems development: challenges and lessons learned. In Proceedings of the 8th international Workshop on Software Technology and Engineering Practice (STEP '97) (including CASE '97) (July 14 - 18, 1997). STEP. IEEE Computer Society, Washington, DC, 452.

[10] M. Morisio, A. Tsoukis: IusWare: a methodology for the evaluation and selection of software products. IEE Proceedings - Software 144(3): 162-174 (1997)

[11] S.B. Sassi, L.L. Jilani, H.H.B. Ghzala, COTS Characterization Model in a COTS-Based Development Environment, APSEC 2003, 352-361.

[12] 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.

[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] Freshmeat 'Website' [web page]. Accessed 15/12/07, from the WWW: http://freshmeat.net/, 2007

[15] Component Source 'Website' [web page]. Accessed 15/12/07, from the WWW: http://componentsource.com/, 2007

[16] SourceForge 'Website' [web page]. Accessed 15/12/07, from the WWW: http://sourceforge.net/, 2007

[17] V. Maxville. 'Research Website' [web page]. Accessed 26/10/07, from the WWW: http://members.iinet.net.au/ maxville/research/, 2007

[18] V. Maxville, J. Armarego, and C. P. Lam, Intelligent Component Selection, 28th Annual International Computer Software and Applications Conference COMPSAC, Hong Kong, 28-30th September, 2004

[19] A. Bagnall, V. Rayward-Smith, and I. Whittley. The next release problem. Information and Software Technology, 43(14):883–890, Dec. 2001.

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

[21] R.M. Hierons, M. Harman and H. Singh. Automatically generating information from a Z specification to support the Classification Tree Method. Vol 2651 of Lecture Notes in Computer Science, p 388-407. Berlin, Heidelberg: Springer-Verlag, 2003.

[22] 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.

[23] Knowledge Representation for COTS Selection, Postgraduate Electrical Engineering and Computer Science (PEECS) Symposium, Perth, Australia,

[24] M. Harman, L. Hu, R.M. Hierons, J. Wegener, H. Sthamer, A. Baresel, and M. Roper, "Testability Transformation," IEEE Trans. Software Eng., vol. 30, no. 1 pp. 3-16, Jan. 2004.

[25] G. Antoniol, M. Di Penta and M. Harman. Search-Based Techniques Applied to Optimization of Project Planning for a Massive Maintenance Project 21st IEEE International Conference on Software Maintenance (ICSM 2005). September 25th-30th 2005, Budapest, Hungary. Pages 240-249.

[26] J. Xiao, C. P. Lam, H. Li, and J.Wang, Reformulation of the Generation of Conformance Testing Sequences to the Asymmetric Travelling Salesman Problem, 15th International Conference on Genetic and Evolutionary Computation Conference - GECCO 2006, 2006.

[27] P. Baker, M. Harman, K. Steinhofel and A. Skaliotis. Search Based Approaches to Component Selection and Prioritization for the Next Release Problem. In Proceedings of the 22nd IEEE international Conference on Software Maintenance (September 24 - 27, 2006). ICSM. IEEE Computer Society, Washington, DC, 176-185.