Techset Composition Ltd, Salisbury
Doc: {IEE}SEN/Articles/Pagination/SEN57782.3d

Published in IET Software
Received on 31st October 2008
Revised on 8th April 2009
doi: 10.1049/iet-sen.2008.0096

ISSN 1751-8806

Applying a reusable framework for software selection

V. Maxville[1] *J. Armarego*[2] *C.P. Lam*[1]

[1]*Edith Cowan University, Perth, WA, Australia*
[2]*Murdoch University, Perth, WA, Australia*
Q1 *E-mail: maxville@ivec.org*

Abstract: With increasing use of component-based development (CBD), the process for selecting software from repositories is a critical concern for quality systems development. As support for developers blending in-house and 3rd party software, the context-driven component evaluation (CdCE) process provides a three-phase approach to software selection: filtering to a short list, functional evaluation and ranking. The process was developed through iterative experimentation on real-world data. CdCE has tool support to generate classifier models, shortlists and test cases as artefacts that provide for a repeatable, transparent process that can be reused as the system evolves. Although developed for software component selection, the CdCE process framework can be easily modified for other selection tasks by substituting templates, tools, evaluation criteria and/or repositories. In this article the authors describe the CdCE process and its development, the CdCE framework as a reusable pattern for software selection and provide a case study where the process is applied.

1 Introduction

As demand for new software escalates, software reuse is increasing productivity and reducing cost, with over 99% of code executed at the US Department of Defence coming from commercial-off-the-shelf (COTS) products [1]. Those who had previously developed complete systems in-house are moving towards COTS-based development: for example, at the University of Southern California (USC-CSSE) projects were 28% COTS in 1997, but had increased to 60% by 2001 [2]. The primary reason given for this change was 'the large increase in the number of COTS products providing application functions' [2]. Comprehensive repositories for applications and components have become a reality, with the open source software (OSS) site Freshmeat housing over 40 000 OSS projects for download [3]. Each of these repository sites provides a search facility to find suitable software. However, if we have more complex needs, for example trading off various selection criteria between candidates, then we need to consider how to improve our interaction with repositories.

Issues with using COTS components include the black-box nature of the software that restricts evaluation and testing to what can be inferred from the published interfaces. It has been shown that developers working with OSS software components generally do not modify the source code, effectively treating them as COTS [4]. Some differences exist, particularly in licensing, but the other commonly raised issues: the money factor, lack of support and reduced security are considered myths [5]. Recent trends in software engineering, including service-oriented architectures (SOA) [6] and agile software development [7] have been identified as areas where COTS lessons and methods can be applied or integrated. For this reason, research into component-based development (CBD) and component-based software engineering (CBSE) needs to continue to find ways to provide usable solutions for a growing population of developers.

One of the earliest CBD processes was off-the-shelf-option (OTSO) [8], published in 1995. This introduced the use of hierarchies of selection criteria elicited using goal/question/metric (GQM) [9] and evaluated using the analytic hierarchy process (AHP) [10], which was adopted in many subsequent processes including STACE [11] and CAP [12]. In the procurement-oriented requirements engineering (PORE) approach [13], an alternative evaluation method, outranking, was described, with the

authors later publishing a paper outlining the limitations of the commonly used approaches, including AHP and weighted score method (WSM) [14]. In these processes, filtering for candidates was manual using sources such as: 'market surveys, Internet search, product publications and sales promotions, and computer fairs and shows' [8, 11]. Over a decade later, the current practice is to undertake an Internet search or ask an expert [4] and evidence indicates that the use of published processes is low, with only 19% of COTS projects using a formal decision-making method [4]. Selection of components has been shown to have the most impact on risk in CBD [15], but developers are still approaching it in *ad-hoc* ways.

The combination of the limitations of current methods for evaluation, improving repository and discovery options and greater uptake of CBD has led a drive for automated tool support to aid software selection. Ruhe [16] compared ten selection processes published up to 2003 and stated that none of them were ready to be included in a decision support system. Key to automation is the characterisation of components, pioneered by Prieto-Diaz [17] and now more market-driven by metadata utilised by repositories. This characterisation often includes substantial ontologies such as the Troves in Freshmeat [3]. In many cases, software selection relies on a search engine, some search terms and the user's patience to trawl through results. This manual process entrusts the search engine algorithm to order the results and then the users to make a comparison based on their criteria, evaluation and intuition. For those requiring quality in their software selection it is important to have objectivity, repeatability and transparency in the selection process.

The use of AI techniques for retrieving code and software artefacts from repositories includes the application of rough-fuzzy sets [18], neuro-fuzzy search robots [19], fuzzy-subtractive clustering [20] and entropy-based fuzzy *k*-modes [21]. These help to sort the software into clusters based on their descriptions. Other work focuses on providing tool support for searching. Maracatu provides search and retrieval options for software developers based on faceted information including platform, component type and component model [22]. Brou [23] implements a search tool which utilises XML descriptions and XQuery to allow browsing of a repository by field. The context-driven component evaluation (CdCE) process differs from these approaches in that it is a process for filtering (searching and retrieval), evaluation and ranking. We have worked with the intrinsic datatypes and implemented a range of transformations to retain and use the information held in each attribute. The process does not treat all attributes as numbers or as database queries, allowing for more tailored use of information. The Classifier Suite provides a graphical representation of the options available to the developer when tuning the selection criteria. The case study gives an example of the developer choosing two different sets of selection criteria to form a more diverse shortlist for further evaluation. As repositories often have incomplete data, we have also allowed for the developer to decide how to interpret missing data – whether it should be assumed to be a failed match on a value, or is its impact softened to avoid losing relevant candidates.

In this article we describe the CdCE process, which implements the framework using a specific set of templates, criteria, ontologies and data representations, with classifiers and test generators for the filtering and evaluation. We then look at the CdCE framework and how is can be reused for alternate implementations and selection tasks. In the final section we consider the reuse of the process through the life of an evolving software system.

2 Context-driven component evaluation

The CdCE process was designed to be usable, transparent and repeatable, providing automation of tasks to allow the user to concentrate on the key decisions in software selection. This addresses the issues raised in Section 1 of the need to build new methodologies for OTS selection to reduce risk in CBD. 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 [24]. We concluded that the CdCE selection process needed to include testing of the candidate software in context. Our treatment of context is not only to evaluate the software in the target environment, but also testing with usage profiles, and of performance and reliability requirements.

The driver for the process is the ideal specification, an XML template for describing the requirements for the software component or application. This is an extension of the CdCE template which we use to describe all software under consideration in the process. A filter is required for each repository to convert the entries to the CdCE format and we can then draw in the data for shortlisting. A filter for the Freshmeat repository [3] has been developed using the back-end XML files available at the site. The filter is written in Java and maps between tags for the data model and has a thesaurus to translate terms. The ideal specification and the repository data are the key external inputs to the process.

When developing the tools to support the process, it was clear that we needed to build a pluggable framework to explore implementation options, for example the choice of Z notation [25] for the behavioural specification. As a result, the process allows replacement to suit the user context, while providing a suite of tools which can be used as they are. Information on modifying the process implementation is given in Section 5.

A description of the original CdCE process and implementation appears in [26]. We have applied standards where possible and given a consistent interface of XML

doi: 10.1049/iet-sen.2008.0096

templates as the inputs, outputs and links between activities within the process, which has simplified experimentation and helps to document the process. There have been enhancements and changes to the process, which we include in the following description of the CdCE implementation.

2.1 CdCE process

The process comprises eight steps (Fig. 1), to provide structure and repeatability in OTS selection. The process is driven by an ideal specification in XML, which includes the contextual information required to recognise a suitable candidate. This is defined by the developer who is wishing to source the component or application. Each of the steps is linked through XML files to provide traceability throughout the process. For our recent investigations, we have worked with metadata from the Freshmeat repository [3] which houses OSS projects at various stages of development. We accept the data values provided by Freshmeat as trustworthy, as assessing validity of the data is not the focus of this work.

Step 1 of the process is the definition of requirements through the ideal specification. We use XML and Z notation to describe the functional and non-functional characteristics of the ideal component. The functional or behavioural description does not require a complete Z specification as it is used to direct testing and should define the most important behaviour of the required software. As part of the ideal specification, we have evaluation metrics that consider the performance of the software against specific criteria. These criteria were based on SWEBOK [27] to give coverage of potential areas of focus for testing. This is referred to as context testing and includes 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 1 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).

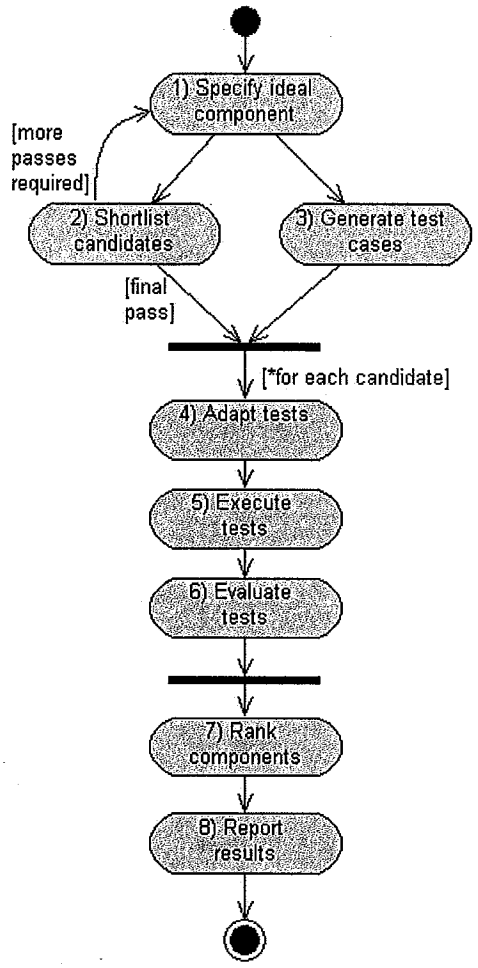

Figure 1 *CdCE process for component selection*

As there may be a need to consider a compromise between the selection criteria where a perfect match is not available, the ideal specification includes attributes on each XML element to indicate whether a criterion is mandatory or non-mandatory. This is where the 'ideal' meets reality in that we may not be able to have a complete match, but can

Table 1 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)

indicate where we can explore dropping non-mandatory criteria. In some cases, there may not be suitable software to match the mandatory criteria, so a re-specification, refactoring of the problem or in-house development may be required.

Given the ideal specification from Step 1, Step 2 takes that specification and generates a model for the classifier. We use the C4.5 algorithm within the Weka machine learning tools [28] to generate the classifiers as it provides human-readable decision trees. A first pass with the classifier and all criteria included (mandatory and non-mandatory) is helpful to highlight any issues with the repository data against the requirements. In some cases there can be a high level of missing data on one or more criteria, resulting in no matches, which will either require loosening that criterion or abandoning the search (with that repository).

Once the ideal specification has been 'tuned' through iteration back to Step 1, we can utilise the automation provided by the CdCE tools and generate a Classifier Suite [29]. The Suite provides a graphical interpretation of the results of various combinations of non-mandatory criteria, shown in Fig. 2. Each set of criteria is identified with a label, for example S10, S23. These labels indicate the set's level in the lattice, and also the number of criteria that have been dropped – S23 is at level 2, and has had one criterion dropped. The criteria are represented as letters, in this case A to E. Using the Classifier Suite, the user can explore the impact of various criteria and drill-down to the shortlists. This is a major decision point in the process and we provide condensed information for the expert to make their choice of shortlist(s) to take into the evaluation phase. Another example of a Classifier Suite graph is given in Fig. 6.

In parallel with Step 2, we have abstract tests generated from the behavioural specification in Step 3. These are based on equivalence partitioning of the operations in the Z specification and currently use a combinatorial generation

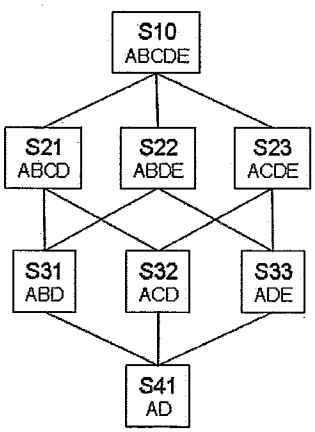

Figure 2 *Graph representing a series of sets, classifiers and subsequent shortlists*

{*A, D*} are mandatory and {*B, C, E*} are non-mandatory

of partition sets for test cases. Using the same abstract tests on all candidates allows for a meaningful comparison between candidates. These abstract tests are adapted to each candidate in Step 4, which also produces input for calculating metrics for functional fit and adaptation effort (FFIT, FEXS, AEFT and TFIT in Table 1). The tests are provided as abstract XML and need to be ported to the local harness/environment, then executed in Step 5, or can be executed manually.

The raw test results are used in Step 6 to calculate the remaining metrics where appropriate – test result (TRES) plus the context metrics CX_P, CX_R, CX_S and CX_U. A second classifier is generated in Step 7 to process the results of the short-listed candidates. Using a similar approach to Step 2, we can iterate to tune the required metrics or generate a Classifier Suite. It is also possible that the functional specification is too strict and needs to 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.

With all the results available from each step of the process, Step 8 collates the data to provide a report which includes 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. The reporting and most of the data processing have been automated, achieving our aim to free up the expert user to concentrate on the key decisions while providing additional information to assist them.

2.2 Development through experimentation

The project underlying the CdCE process aims to develop and evaluate strategies for the intelligent selection of software components. This work has explored solutions through the application of the spiral development model (SDM) [30], where each iteration has focused on approaches and enhancements to aspects of the process: each phase considered options, implemented one or more of them, evaluated the results, then planned for the next iteration.

Spiral 1 developed the specification for the OTS components based on use cases, industry and organisational standards. In Spiral 2, the process was drafted and applied to real-world data, manually collected from five online repositories with a combined listing of over 87 500 software items [26]. Four iterations were able to provide a shortlist of nine candidates through the manual application of selection criteria. WSM was used for the ranking in [26], while AHP was also investigated. With nine candidates, the pairwise comparisons between each candidate on each criterion showed that the AHP quickly became taxing on expert time and would not be able to scale.

Responding to the experience of manually applying the process in Spiral 2, we looked into a range of AI

doi: 10.1049/iet-sen.2008.0096

techniques that could be suitable in Spiral 3. The first experiments were proof of concept for applying AI techniques including expert systems, machine learning (C4.5) and artificial neural networks (ANN) [31]. Expert systems were discarded because of the need to code them up for each selection task – we did not anticipate having the reuse to justify the effort. Using generated and then real-world data, we found that C4.5 and ANN were able to be trained to classify the component data at accuracies over 90% [32]. The choice was made to continue with C4.5 as its error rate was lower and, more importantly, it provides a decision tree which aids transparency in decision making.

Spiral 4 concentrated on enhancing the data representation and experimenting with data transformations to tease out a much as possible from each attribute. The experiments used 33 262 entries in the Freshmeat repository (at that time) with five attribute types and five data transformations. The transformations allowed the comparison of increasingly complex and tailored processing of the attributes, with Transformation 1 (T1) being equivalent to a Boolean SQL query and Transformation 5 (T5) utilising the ontology hierarchy and distance matrices. The case study focused on shortlisting and found that T5 identified five candidates, while T3 returned two and T1 returned 1 [33], indicating that a Boolean SQL query would have been far inferior to the T5 transformation.

As a result of developing hand-drawn graphs to help visualise experimental data, the Classifier Suite was implemented as a tool to assist in understanding the data generated in the shortlisting process [29]. The data from three previous case studies were used to explore the value of the tool in informing the selection process. It was found that the Classfier Suite was effective in assisting the user in choosing an appropriate set of criteria, aided in the reasoning about the choice and let them view more options than would have been possible otherwise.

Throughout this project we have put a priority on working with real-world data and considering the usability of the process and tools. The experiments and case studies have informed the investigation and feedback from working through case studies has been a major guide in choice of strategies to explore.

3 Reusable framework

The CdCE process as described above is an implementation of a more abstract selection framework. Through the ideal specification, the process allows reuse on a task level. Substituting a different set of requirements through a modified ideal specification re-wires the process to select any type of software. Beyond this task-level reuse, we can also switch many aspects of the implementation of the process. Specifically, this may include any or all of the items in Table 2.

Table 2 Replaceable elements of the CdCE process framework

template*	metrics*
ontology*	thesaurus*
distance matrix*	classifier (or alternate tool within Weka)
model generator	filter (i.e. replace classifier and model generator)
behavioural specification language	test generator
data representations (attribute types)	data transformations
repository*	

Each activity in the process has clear input and output items, allowing for the independent substitution of many of these and/or the tools used to implement the steps. Through the development of the process, each item has undergone experimentation and update and been shown to be replaceable. Clearly, changes to the XML template for the component specification would have some effect throughout the process; however, the strict use of object-oriented programming techniques contains much of the impact. Adding new entries in the specification requires an attribute type to be assigned, and this relationship is held in another XML file used by almost all of the programs. Similarly, the items with an * are all implemented as XML files which can be substituted through changes to parameter files and/or minor changes to the code.

The ideal specification was developed to be compliant with Dublin Core [34], a standard for the description of electronic resources. With substitution of aspects of the implementation, the CdCE framework could be applied to a wide variety of resource discovery tasks.

To explain what would be involved in re-applying the framework, we now look at revising attribute types, ontology and transformations in more detail.

3.1 Attribute types

In deciding on the datatypes, we focused on the way the textual data could be handled. Data may be structured or unstructured, affecting the processing and matching techniques that can be applied [35]. Structured data are deterministic, and is quite straightforward to match and transform (e.g. simple text, date and numeric data). Unstructured data may require surrogates to be made to represent the data (word lists for textual data) or auxiliary representations of the content can be developed (e.g. classification trees and ontologies). Comparing these categories with the specification template, we identified four data types: numeric, date, text and descriptive text (longText). There were two clear categories of text: those that could have

standardised values (ontology), and those that were less predictable (freeText). FreeText entries include the software name and the developer name. Ontology attributes are based on a restricted vocabulary which uses a thesaurus to standardise terms. The terms are organised into hierarchies for each criterion to represent the relationships between the terms. LongText entries are longer descriptions which are dealt with using information retrieval techniques such as removal of stop words and punctuation, and calculations based on recall/relevance calculations. Word stemming was considered, but has not been implemented.

To implement additional attribute types, the new attribute is created as a subclass of the abstract attribute class. All interaction with the attributes in the developed tools is in terms of the attribute class, thus there is a series of methods that need to be implemented to suit the attribute concerned. Perhaps the most effort would be required to interpret the data transformations for a new type. This would be based on the existing transformations which begin as the equivalent of Boolean comparisons and progressively utilise more of the inherent information to be found in an attribute, for example ontology attributes make use of the hierarchy in the ontology. It can also be helpful to work with variations on the transformations to explore how best to transform the new datatype. This was the approach taken with ontology attributes.

3.2 Ontology implementation

Using an ontology, and creating matching attribute types, provides a way to determine the closeness/similarity of terms, and provide levels of abstraction. For attributes such as maturity, the ontology's distance matrix can also resolve whether a value satisfies a requirement. For example, if the maturity requirement is for 'stable', then 'stable' and 'production' should both be satisfactory.

Table 3 Freshmeat Trove mappings

Freshmeat attribute	Number of elements	CdCE attribute
development status	6	devStatus
environment	25	standard
audience	6	–
licence	57	licence
network environment	3	standard
operating system	32	operatingSystem
programming language	62	devLanguage
topic	344	subject
translations	36	language

We chose to reuse an established ontology, developed for the Freshmeat website [3]. The Freshmeat Trove categories include nine attributes, which we map into the corresponding CdCE attributes (Table 3). The Trove enumerates the vocabulary for the attributes, as well as the hierarchy of terms. Fig. 3 shows the hierarchy for operatingSystem. The ontology data are extracted into the CdCE ontology format (Fig. 4), creating a separate tree structure for each of the ontology attributes in the CdCE specification. The ontology was recently updated with new data from the Freshmeat Trove. The process of

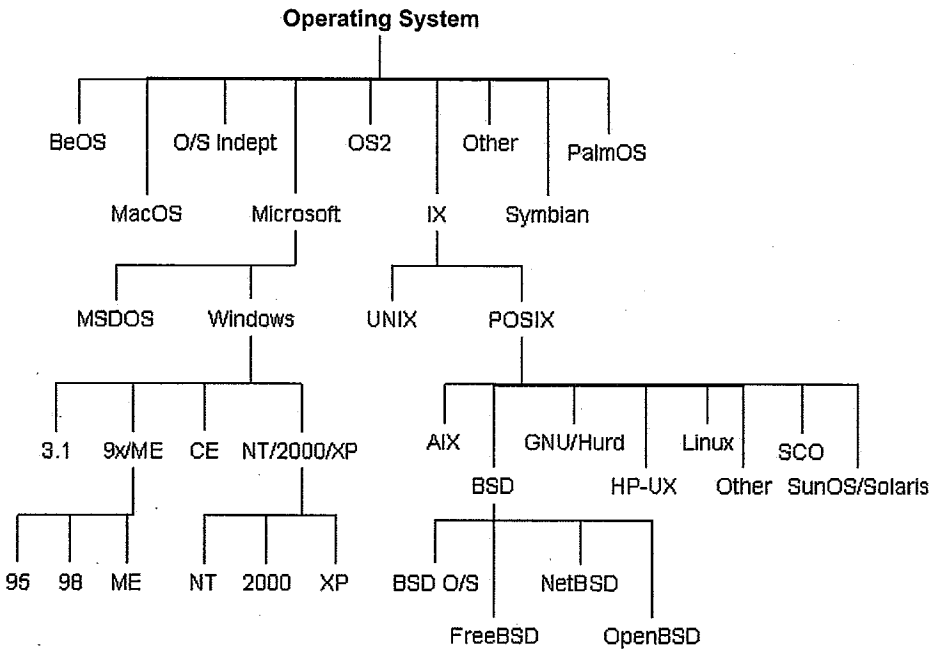

Figure 3 *Operating system hierarchy*

doi: 10.1049/iet-sen.2008.0096

```
<ontology>
    ...
    <descriminator>
        <id>9</id>
        <name>Development Status :: 3 - Alpha</name>
        <mapsTo>3 - Alpha</mapsTo>
        <parent_id>6</parent_id>
        <root_id>6</root_id>
        <field>devStatus</field>
    </descriminator>
    <descriminator>
        <id>10</id>
        <name>Development Status :: 4 - Beta</name>
        <mapsTo>4 - Beta</mapsTo>
        <parent_id>6</parent_id>
        <root_id>6</root_id>
        <field>devStatus</field>
    </descriminator>
    ...
</ontology>
```

Figure 4 *Ontology XML file*

updating the ontology involved accessing the XML backend to Freshmeat, then regenerating the thesaurus and ontology files. Attributes that have a distance matrix (described in Section 2.4) need corresponding new entries to be added into the matrices. The updated files are then linked to the CdCE programs via parameter files. A similar process could be followed to substitute an alternative ontology for use with the CdCE process.

3.3 Data transformations

In [31], the CdCE classifier worked with a Boolean representation of the match between requirements and observed values. This produced a decision tree based on Boolean values and the test data were also Boolean. At that point the classifier was equivalent to a database query. To explore and extend the utilisation of data, we worked on the representation of the 'closeness' of two values. These can be transformed between types, for example numeric values can be discretised into ordinal or enumerated types, which allow more useful information to be retained than a basic Boolean output. We use the ideal specification to determine the transformed value for each attribute within each instance. The transformations we have applied are given in Table 4.

Each attribute type needs to have a different interpretation of the required calculation for the selected transformation (T1–T5). The transformations are implemented easily using object orientation. We currently apply the same transformation across all attributes, but have the facility to select different transformations at the attribute level. The transformations for ontology attributes are now described in detail.

Using an ontology attribute as an example, we can show how each transformation is treated differently (Table 5). The first transformation is a simple string compare – match/no match. The second transformation is a loosening of the criteria, and checks for whether the required string is a substring of the given value. Transformations 3–5 make

Table 4 Transformations used

Trans.	Description
T1	simple match on value or range, equivalent to [31]
T2	as above, with a loosening to accept 'close' values
T3	converts the comparison to a score between 0 and 10.
T4	similar to T3, but uses a different calculation for longText
T5	same as T3, but ontology attributes can be abstracted

Table 5 Ontology attribute transformations

Trans.	Calculation	Output type
1	if matched = true else false	(true, false, −999)
2	if matched = true else if substring matched = border else = false	(true, false, border, −999)
3–4	distance(have, require)	real
5	distance(abstract(have, level), abstract(require, level))	real

use of a distance matrix to calculate similarities. The matrix includes all combinations of values for each attribute, with the indexed element (i, j) being the score out of 10 if we have value i and want value j. The matrices are not normally symmetrical and have been generated manually. Table 6 gives the distance matrix for maturity. The distance is used as-is for Transformations 3 and 4. In Transformation 5, the values are abstracted by travelling up through the ontology tree to the designated level in the hierarchy. The distance calculation is then based on the abstracted value. In the operatingSystem hierarchy (Fig. 3), if the ontologies are drawn up to level 2, then all POSIX-based operating systems would be matches for each other. Thus Transformation 5 can be used to abstract the selection process for all ontology attributes. We have found that Transformation 5 gives better results than Transformations 1–4 as it captures suitable candidates that would otherwise be missed.

To extend on the transformations, the attribute superclass would need to be updated to include more transformation entries. Then within each of the attribute types, there are a small number of methods that work with the transformations within a case statement. These case statements need to have new entries for each new transformation index. Once this is done the transformations can be used by indicating them in the parameter files.

Table 6 Distance matrix for maturity

	Want					
Have	Plan	Pre-alpha	Alpha	Beta	Prod'n/ stable	Mature
planning	10	8	6	4	2	0
pre-alpha	10	10	8	6	4	2
alpha	10	10	10	8	6	4
beta	10	10	10	10	8	6
prod/ stable	10	10	10	10	10	8
mature	10	10	10	10	10	10

4 Case study

We now demonstrate the use of the CdCE process and tools to source an XML editor from the Freshmeat repository. The ideal specification is recorded in XML (Fig. 5) and states the requirements and our alphabetic handles to reference each of the criteria. In summary, the description (A) and detailed description (B) are 'XML editor', written in 'Java' (E) to run on a 'Linux' platform (F). We are looking for a 'Mature' product (D) with a recent update (G), and would like a 'GNU General Public Licence' (C) or similar. Values were defined for 'memory' and 'disk space' limits, and for each of the relevant metrics. We flag two criteria as mandatory – 'detail' and 'devLanguage' as other individual criteria can be loosened, but we consider these to be the core attributes.

We begin Step 2 using the full, most restrictive ideal specification and Transformation 5 (see Section 5.3). Previous investigations have indicated that this transformation provides improved short lists as semantically similar results are captured through the use of distance measures and ontologies for abstraction of some text attributes. This first run results in an empty shortlist, and the statistical tools in Weka show that there is a high level of missing data on the memory and diskSpace attributes. We remove these from the ideal specification and generate a suite of classifiers for the combinations of sets of criteria (Fig. 6). To reduce the size of the graph, the date criteria (G) is overlayed in the figure. In set S10, the 1:0 represents one item returned with date omitted, and no items returned if it is included.

The aim when working with these sets is to gain the highest relevance of results while not having too many candidates on the shortlist, so we look through Fig. 6 to find 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. This reasoning about the short lists is assisted by the tools, but requires some interpretation. An example of the reasoning used with a 32 set Classifier Suite follows:

doi: 10.1049/iet-sen.2008.0096

```
                    <?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/" >
A - Non-mandatory         <dc:description  type="mandatory">XML editor</dc:description>
B - Mandatory             <dc:detail type="mandatory">XML editor</dc:detail>
C - Non-mandatory         <swv:licence type="mandatory">GNU General Public License (GPL)</swv:licence>
D - Non-mandatory         <swv:devStatus type="mandatory">6 - Mature</swv:devStatus>
G - Non-mandatory         <dc:date type="mandatory" min="01-01-2005" max="31-12-2007">31-12-2007</dc:date>
                          <swv:technical>
E - Mandatory                 <swv:devLanguage type="mandatory">Java</swv:devLanguage>
F - Non-mandatory             <swv:operatingSystem type="mandatory">Linux</swv:operatingSystem>
                              <swv:systemRequirements>
   Missing                       <swv:memory type="mandatory" min="15" max="50">20</swv:memory>
   Missing                       <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>
```

Figure 5 *Ideal specification for XML editor case study*

We have two criteria that are similar: description and detailed_description, and may be redundant is some cases. Drilling down through the Classifier Suite graph to the metadata we find 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. Two of the remaining 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 repository have taught us that date and devStatus are important for ensuring active and mature projects. As S21 does not include date, we look for nodes including this criterion combined 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 choose S35 to maximise the range of items on the shortlist. We thus accept two four item short-lists – S21-date, S35 + date, giving a total of eight candidates with these changes.

Step 3 generates a test suite of 12 test cases, concentrating on loading and validating XML documents and taking in test data (XML files) from the specification. We provide an XSLT transformation to convert the XML test cases in HTML forms to record the results for each candidate. The shortlisted candidates are downloaded in Step 4 and a manual comparison is made on the differences between the available and required interfaces. We now refer to the candidates as C1–C8 for ease of reference and to de-identify the software. Through this step we are able to 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 7. Candidate C8 was not available for download as it is now a commercial product with no trial version.

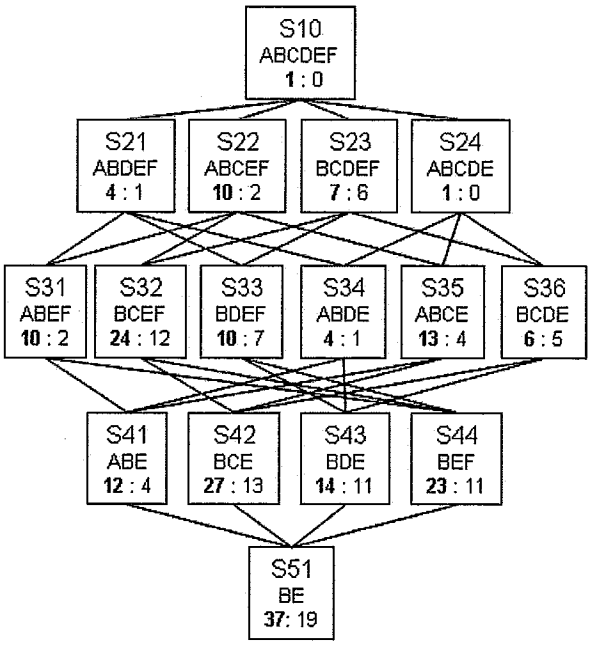

Figure 6 *Graph representation of case study shortlists*
Date (G) are overlayed on the graph as the right-hand shortlist count

Given the adaptation information from Step 4, we transform the abstract tests to executable tests in Step 5. In this case study the tests are manually executed and results recorded in HTML forms. Each candidate is tested in Step 6 and the results are collated, generating metrics for TRES, TFIT and the contextual test metrics (if applicable). In this case study we have usage-based tests, so we only generate a score for CX_U. 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 7.

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. The initial values for the metrics exceeded the performance of all of the candidates, so they were loosened by changing the values. After the second pass with the classifier, 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. With the ranking complete, the information for the whole evaluation is compiled into a report (Step 8) including recommendations along with electronic versions of all artefacts including specifications, classifiers and tests.

4.1 Evolution

The work done in the CdCE project has aimed to allow the automation of a generic approach to evaluating software. The first level of reuse is thus the process, associated tools and

Table 7 Complete evaluation for shortlisted candidates

Cand.	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

Ten is a perfect result

procedures. These are documented in [31]. The generated tests can also be used for regression and system testing. As the process includes determination of the adaptation model, that can assist the integration of the selected component. For a particular selection task, and related/subsequent selections, we can reuse some or all of the artefacts of the process. The ideal specification is the foundation artefact of the process. When considering the repetition and evolution of selection tasks, we have four significant reusable artefacts, the shortlisting classifier, the classifier suite, the test cases and the evaluation classifier.

We now consider at a hypothetical extension of the case study. A few months later, it may be that the developer supplying the XML editor has gone out of business. Without on-going support, the software puts the organisation at risk. The developer could have used the next best option from the original selection results, but they decide to update the search from the repository which reuses the short-listing classifier, Classifier Suite, abstract test cases and the evaluation classifier. For our simulation of this situation, we have injected instances into the repository to represent new software projects. When we rerun the classifiers from the original search over the repository we confirm that they pick up the new items and the process is completed to locate a new XML editor.

Six months later, the stakeholders decide that the XML editor should have style sheet functionality evaluated. Extra schemas are added to the formal specification to address the new requirements. We are able to reuse the short-listing classifier, the Classifier Suite and the evaluation classifier. The re-selection process can now begin. The short-listing classifier is reused and produces an updated set of candidates equivalent to those in the first run. New abstract tests are generated which include tests of the style sheet functions. Some adaptation models may be reused from last time as many candidates are on both short-lists. The tests are executed using the harness from the initial selection. Evaluation of the test results gives guidance in

doi: 10.1049/iet-sen.2008.0096

choosing the replacement software, and the developer can be confident that the same requirements have been met through the use of artefacts from the initial selection process.

5 Conclusion

The design of the CdCE process was intended to provide flexibility and reuse. While this is clearly possible at the task and system evolution level, we can also view the process as an implementation of a more abstract framework for selection. The framework describes the three-phase process of filtering, evaluating and ranking items, which can then be populated with tools and procedures for selecting software or other resources. We can then use this as a pattern to provide consistency and repeatability in many tasks that need to extract and select items from repositories. It also makes it possible to concentrate on particular parts of the process, as we have in exploring the filtering and short-listing activity (Step 2).

Throughout the development of the CdCE process, real-world data and experimentation have been used. The Freshmeat dataset is large and diverse which presents issues such as missing data and immature or dormant projects. This has influenced the development of the process and made it more robust and relevant for real-world application.

6 References

[1] BASILI V.R., BOEHM B.W.: 'COTS-based systems top 10 list', *IEEE Computer*, 2001, **34**, (5), pp. 91–93

[2] BOEHM B., PORT D., YANG Y., BHUTA J., ABTS C.: 'Composable process elements for developing COTS-based applications'. Proc. 2003 Int. Symp. Empirical Software Engineering (ISESEO03), pp. 8–17

[3] Freshmeat: 'Website' from the WWW: http://freshmeat.net/, [web page] accessed 31 April 2007

[4] LI J., *ET AL.*: 'Validation of new theses on off-the-shelf component based development'. IEEE METRICS, 2005, p. 26

[5] DI GIACOMO P.: 'COTS and open source software components: are they really different on the battlefield?' in FRANCH X., PORT D. (EDS.): ICCBSS 2005, 2005, (*LNCS*, **3412**), pp. 301–310

[6] ANDERSON W.: 'What COTS and software reuse teach us about SOA'. ICCBSS, Sixth Int. IEEE Conf. Commercial-off-the-Shelf (COTS)-Based Software Systems (ICCBSS'07), 2007, pp. 141–149

[7] NAVARRETE F., BOTELLA P., FRANCH X.: 'How Agile COTS selection methods are (and can be)?'. EUROMICRO-SEAA, 2005, pp. 160–167

[8] KONTIO J. OTSO: 'A systematic process for reusable component selection'. CS-TR-3478, University of Maryland, 1995

[9] BASILI V., ROMBACH H.: 'Tailoring the software process to project goals and environments'. Proc. ICSE87, 1987, pp. 345–357

[10] SAATY T.L.: 'The analytical hierarchy process' (McGraw-Hill, New York, 1990)

[11] KUNDA D., BROOKS L.: 'Applying social-technical approach for COTS selection' in BROOKS L., KIMBLE C. (EDS.): UKAIS 99, Proc., April 1999. ISBN: 0-07-709558-8

[12] OCHS M.: 'Using software risk management for deriving method requirements for risk mitigation in COTS assessment and selection'. SEKE 2003, pp. 639–646

[13] NCUBE C., MAIDEN N.A.: 'PORE: procurement-oriented requirements engineering method for the component based systems engineering development paradigm'. Proc. 1999 Int. Workshop on Component Based Software Engineering

[14] NCUBE C., DEAN J.C.: 'The limitations of current decision-making techniques in the procurement of COTS software components'. ICCBSS 2002, pp. 176–187

[15] PORT D., CHEN S.: 'Assessing COTS assessment: how much is enough?'. ICCBSS 2004, pp. 183–198

[16] RUHE G.: 'Intelligent support for selection of COTS products', *Web Web-Servi. Database Syst.*, 2002, pp. 34–45

[17] PRIETO-DIAZ R.: 'Implementing faceted classification for software reuse', *Commun. ACM*, 1991, **34**, pp. 88–97

[18] RAO D.V., SARMA V.V.S.: 'A rough:fuzzy approach for retrieval of candidate components for software reuse', *Pattern Recognit. Lett.*, 2003, **26**, (6)

[19] KUO Y.-H., HSU J.-P., HORNG M.-F.: 'Neuro-fuzzy based search robot for software components', *Int. J. Arti. Intell, Tools*, 1999, **8**, (2), pp. 119–135

[20] NAKKRASAE S., SOPHATSATHIT P., EDWARDS W.R.: 'Fuzzy subtractive clustering based indexing approach for software components classification', *Int. J. Comput. Inf. Sci.*, 2004, **5**, (1)

[21] STYLIANOU C., ANDREOU A.S.: 'A hybrid software component clustering and retrieval scheme using an entropy-based fuzzy k-modes algorithm'. Proc. 19th IEEE Int. Conf. Tools with Artificial Intelligence, 29–31 October 2007, vol. 1, ICTAI, Washington, DC, pp. 202–209

[22] GARCIA V., *ET AL.*: 'From specication to experimentation: a software component search engine architecture' in GORTON *ET AL.* (EDS.): CBSE 2006, 2006, (*LNCS* **4063**), p. 8297

[23] BROU K.: 'Querying of open source programs libraries: an approach based on a XML metadata repository'. IEEE SITIS, pp. 100–106

[24] WEYUKER E.: 'Testing component-based software: a cautionary tale', *IEEE Soft*, 1998, **15**, (5), pp. 54–59

[25] SPIVEY J.M.: 'The Z notation: a reference manual' (Prentice Hall international series in computer science', 1992, 2nd edn.)

[26] MAXVILLE V., LAM C.P., ARMAREGO J.: 'Selecting components: a process for context-driven evaluation'. Proc. 10th Asia-Pacific Software Engineering Conf., Chiang Mai, Thailand, 10–12 December 2003

[27] Institute of Electrical and Electronics Engineers, Inc.: 'Guide to the software engineering body of knowledge' (New York, NY, USA, 2001), available online http://www.swebok.org

[28] QUINLAN J.R.: 'C4.5: programs for machine learning' (Morgan Kaufmann, San Francisco, 1993)

[29] MAXVILLE V., LAM C.P., ARMAREGO J.: 'Supporting component selection with a suite of classifiers'. 2008 IEEE World Congress on Computational Intelligence, Hong Kong, 1–6 June 2008

[30] BOEHM B.: 'Anchoring the software process', *IEEE Softw.*, 1996, **13**, pp. 73–82

[31] MAXVILLE V., LAM C.P., ARMAREGO J.: 'Learning to select software components'. Proc. 16th Conf. Software Engineering – Knowledge Engineering (SEKE), Banff, Canada, 20–24 June 2004

[32] MAXVILLE V., ARMAREGO J., LAM C.P.: 'Intelligent component selection'. 28th Annual Int. Computer Software and Applications Conf. COMPSAC, Hong Kong, 28–30 September 2004

[33] MAXVILLE V.: 'Knowledge representation for COTS selection'. Postgraduate Electrical Engineering and Computing Symp. (PEECS), Perth, Australia

[34] Dublin Core Metadata Initiative: Website http://dublincore.org/, 2005

[35] SANCHEZ J.A., PROAL C., MALDONADO F.: 'Supporting structured, semi-structured and unstructured data in digital libraries'. Proc. Mexican Int. Conf. Computer Science (ENC 2004), 2004

doi: 10.1049/iet-sen.2008.0096

SEN57782

Author Queries

V. Maxville, J. Armarego, C.P. Lam

Q1 There is a mismatch in the corresponding author email address between the cover sheet and the doc file. We have followed the one given in the cover sheet. Please check.

Q2 Please provide other two authors name instead of Et al in Refs. [4, 22].

Q3 Pl. provide vol. no. to ref. [16].

Q4 Pl. provide page range to refs. [18, 20].

Ms. Valerie Maxville
School of Computer and Information Science
Edith Cowan University
2 Bradford St
Mt Lawley Western Australia
Australia

Reference: Item Number 0096L Ref. Id: SEN-2008-0096
Applying a Reusable Framework for Software Selection

IET– PROOFS

Please check the enclosed proofs carefully. The ultimate responsibility for the accuracy of the information contained within the paper lies with the author.

However, only essential corrections should be made at this stage and the proofs should not be seen as an opportunity to revise the paper.

Please return all corrections to this office by the date on the proof. Late corrections may not be included.