Selecting Components: a Process for Context-Driven Evaluation

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

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

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

Abstract

This paper describes a process for selecting and evaluating candidates for component based software engineering. The process is aimed at developers sourcing components from third party vendors. Component metadata and a formalised specification of the ideal component, including context information, are used to drive the process. This specification is used to shortlist candidate components from commercial repositories and to generate the tests and adaptations for the candidate components. Metrics from each stage of the selection and evaluation process are then combined to compare and rank components for inclusion in the target application. This approach to component selection, using context information and formal methods, helps address issues with component sourcing, selection and testing of third party components.

1. Introduction

Software reuse can help to reduce instances of overtime, over budget and unreliable software. Object-oriented techniques, new programming languages and new software frameworks have engendered a growing industry: Component-Based Software Engineering (CBSE) [1]. There are now thousands of components registered with brokers such as Component Source, who make this software available to the public [2]. To gain the full potential of component based software development, access to and integration of components has to be made as smooth and problem free as possible. Unfortunately, difficulties are encountered when developing software from components, appearing in different stages of the lifecycles of components and component-based systems. Component developers face increased development time for making software that is suited to the general case. Application developers may have difficulty sourcing and selecting candidate components. Use of third party components increases risks for the developer, so a satisfactory level of trust is required. Trust can be improved via certification and/or testing [3]. The context for the component in the target system can result in mismatches and the use of context in testing can improve confidence and trust. The independent development of third party components forces vigilance on the part of the application developer to keep up with new releases, adding to the effort required in testing, risk management and configuration management. Each of these issues must be addressed to improve the uptake and confidence in CBSE.

We have chosen to focus on the acquisition phase, concentrating on the assessment and evaluation of candidate components. Our approach is to provide an automated process for evaluation and testing of candidate components. This improves efficiency and confidence and creates test artifacts that can be reused in system and regression testing.

2. The CdCT Project

The Context-driven Component Testing (CdCT) project aims to address the issues of **sourcing**, **selection** and **evaluation** of software components, with indirect benefits in **testing** and **trust**. The process is driven by a specification of the ideal component and its operating context which provides a foundation for the development of tools and strategies for the intelligent selection of software components. The case study described in this paper follows the scenario of an application developer sourcing, short-listing and evaluating components for application in a specific problem context. We consider the real-world issues encountered when sourcing components from commercial and open source brokers, and how to compare components with varying implementations and levels of documentation. This is a more complex problem than selection from in-house repositories as an organization is able to dictate the level and style of documentation for their software.

Our motivation is to encourage wider use of components by minimising the amount of effort and maximising the benefit gained by application developers in considering third party software: any approach to a broad search for candidate components (i.e. no specific name to search for on the Internet) will be time consuming and likely to return many results that are not relevant. In addition, brokers, such as Component Source [2] and Active-X.COM [4], are restricted to those components that have been registered to their catalogue, so multiple brokers need to be searched.

These calls on developer effort can be partially addressed by providing, through this project, standardised metadata about software components that can then be indexed and matched through existing search engines. This enhances the relevance of the results by only returning information about matching software components (and not assorted document types). A short-listing process is then carried out to tighten the match between possible candidates and the ideal component specification. This is based on metadata and can use multiple passes to produce the desired shortlist of candidates. At this point the developers can be confident that they have a representative set of the available components. Harvested components are then evaluated and ranked to determine the most suitable one(s). This project aims to assist developers in the evaluation phase through the provision of a standardised specification and through strategies and tools for testing and ranking components. Examples of criteria used to rate components for comparison include performance, security, ease of integration, or a combination of these and other indicators.

Components are written for the general case and require contextual information and testing to fully evaluate their suitability to an application [5]. The developer needs to know that the component is not only reliable and meets its specification, but that it is suited to the target system. Certification of components is not sufficient. Our **ideal component specification** includes details of the requirements for the component and aspects of the target system to allow a context-aware evaluation of a component's suitability.

2.1. Issues in Selecting and Evaluating Components

Selection and evaluation of components requires metrics and a process to elicit values for those criteria. The following sections discuss the basis for our approach which aims to improve confidence and assist with component selection and testing.

2.1.1. Component Selection

Once a developer commits to using CBSE, information about available 3rd party software is required. At this point, constraints of the project should be known and can be used to reduce the number of contending components. However, there is no standard for the documentation of a component. This creates difficulties when trying to compare components based on the vendor's "shipping information". Thus the selection of a component becomes complex and time consuming, resulting in fewer components being considered for the application. Broker sites can assist the selection process by providing customer rating systems, however, these orderings are based on the requirements of other organisations.

Once there is a range of components to match a given requirement, we then need to select the "best set" of components for the task at hand. This process may be optimised to select components displaying particular properties, for example, performance, security or ease of integration. One approach to component selection is K-BACEE – a system for selecting the best ensemble of components based on a weighting of their performance against certain criteria [6]. Other work takes the context of the component into account when using an in-house repository [7]. This project addresses the specification of components, and how to automate the selection process to allow consideration of a larger range of components from external repositories.

2.1.2. Component Trust

Approaches to improving component trust include certification, contracts, and self testing components.

Most of these rely on dynamic testing of components by the developer, user or a third party organisation. Software certification has many flavours. For some, it should be done by certifying the organisation and its processes, implying a high quality product results from high quality processes. This is advocated by users of the Capability Maturity Model (CMM) developed by the SEI [8] and in a more general sense, the ISO 9000 standard. Others see this as a flawed argument and advocate that the product should still undergo full testing before it can be certified. Voas [9] puts forward a model for certification to take place remotely, after deployment. Aimed at mass-market software, the independent certification in this model is carried out by Software Certification Laboratories (SCLs) which receive testing data from actual users, providing access to higher numbers of test results using real world data. An alternative, aimed at the smaller end of the market, is to ship test certificates with the component [10]. The application developer can then run and verify the results of the included tests against the component to make their own assessment. A draw-back of certification is that it ignores the target context, testing the component in isolation. This provides confidence that a component meets its specification, but application developers will still need to thoroughly test the component in their environment.

2.1.3. Testing Components

We see testing of third party components by application developers as unavoidable. Research into predictability of assemblies of components highlights the increased importance being given to the target context of the component [11]. With components, the testing process is made more complex by the variety of environments that a component may be expected to execute. Weyuker [5] discusses the problem of testing software components to take their context, or target operating environment, into account. When testing a component for reuse in a new or changed environment, it is important to prioritise testing based on expected usage. Beyond unit, integration and system testing, organisations can carry out feature testing and load testing (including performance, stability, stress and reliability testing). These more specialised tests take the context of the component into account and will vary between applications of the same component.

Third party components pose special difficulties in testing. Developers of components and certification bodies may make use of white box testing techniques as they have access to the component's source code [10]. However, application developers can only expect an executable and associated documentation to work from. This dictates that component testing (from the user perspective) is limited to black box techniques [12]. An issue for component users is the amount of documentation available. If we have a specification of the required functionality and/or interfaces we can work from metadata [13], Unified Modelling Language (UML) [14] or Assertion Definition Language (ADL) [15]. These can form the basis for specification-based testing. Test generation options are quite limited if only provided with interface descriptions. Partition information for input variables can aid in the selection of test data to exercise various scenarios. Behavioural information can provide partitioning information and also assist in developing meaningful sequences of method calls and some oracle functionality. We view specification-based testing using formal specifications of interface and behaviour to be the most appropriate option for testing third party components. In this project we use metadata for short-listing components, and combine it with formal specifications of the component for test generation and evaluation.

Automatic test case generation is important in reducing the effort required for the evaluation of third party components. We have chosen the Z notation from the available formal methods, noting previous research using Z for automated test generation, available tools and our familiarity with the language. Z notation has successfully been applied to automated test generation, and is well suited to describing component interfaces and behaviour. The approaches to automated testing using Z have three main strategies: to use Z for specification and test generation [16], to start with another model (e.g. finite state machine (FSM)) and convert to Z [17], or to specify with Z, then convert to a state machine for test case generation [18]. Techniques from other model (state) based specification languages can be applied to Z, (e.g. VDM), and if the Z specification is converted to a state machine, we can draw on FSM and graph theory-based approaches to test generation.

Z also assists in the checking of test results. Test oracle functionality can be provided through the Z specification or by using a schema compiler to generate executable code from Z.

In this project, we try to assist the application developer in selecting and evaluating component by facilitating the testing of multiple candidate components. This testing is able to include site specific requirements through context schemas in the ideal component specification (see table 1).

2.1.4. Specifying Components.

The CdCT project has developed a data model for the specification of components. Requirements for the data model included that it should contain data useful for component selection and attempt to adhere to current and foreseeable trends and standards in the documentation of electronic resources. The attributes identified for each component had to allow for standardised searching as well as provide for a technical specification to facilitate the later testing and evaluation of components. We have used the Dublin Core metadata standard [19] for generalised fields in the component specification, such as developer and support contact details.

Table 1. Context related operation schemas

Name	Attributes
CX_values	individual constraints group constraints
CX_probability	individual values combinations of values method calls combinations of method calls
CX_sequence	values method calls
CX_frequency	method calls
CX_response	maximum response time
CX_critical	individual values combinations of values method calls combinations of method calls
CX_environment	required environment variables facility to change environment variable values model of interfacing methods

More specific information is required for the evaluation of components. A recent workshop on CBSE processes found difficulty in finding consensus on what a component specification should include [20]. The discussion concluded that the specification should have descriptions of public interfaces, and little more could be expected. Our data model obviously needs to hold technical information about the interfaces offered by a component. As we are working towards automated test generation and execution to assist in the decision making process, we also require information about the component's behaviour. A formalized representation of the component is able to convey both interface and behavioural information. Using Z notation allows the behaviour of the component to be included in the specification, useful for test generation and oracle purposes. Options for context information include usage profiles, required response times and critical interfaces/values. We have developed **context Z schemas** to focus the test generation on areas that are important for the application under development. These schemas add environment information to the interface and behaviour information in the Z specification and are detailed in [21].

2.1.5. Representing the Specification

The CdCT data model needs to satisfy the multiple needs of component developers, brokers and users. Broking and searching are assisted by metadata and adherence to general standards for describing electronic resources (Dublin Core). We have followed metadata conventions and kept less technical information in a shallow tree for easy access. More sophisticated searching and evaluation is facilitated by the context metadata and the technical description, which also feed into the test generation process. Technical information about the component is held in a separate tree of the data model. To improve interoperability, we have chosen to implement the data model in eXtensible Markup Language (XML) and related W3C standards [22].

The schema developed allows for the specification of interfaces using specialised XML tags or within the tags that contain the Z specification for the component. Context information is recorded between the metadata (e.g. platform requirements) and in specialised Z operation schemas (e.g. usage profiles). The Z specification is currently encoded using LaTeX [23], which is compatible with current Z tools.

The CdCT project has necessitated the development of a suite of schemas and XSL transformations to coordinate data and results and to allow traceability of information throughout the selection and evaluation process. The user interface manages the documents for a given acquisition project through XML related tools and standards. The user always has the option of viewing and editing the underlying text files, or import/exporting files for use with other applications.

3. The CdCT Process

The overall process followed for selecting components follows the activity diagram (Fig. 1). The first step **(1)** is to define the problem and the requirements for the component. A formal description of the component behaviour and its context information is recorded in Z notation. The next task **(2)** is to create a short-list of candidate components. This is based on the **ideal component specification** and may take a number of passes to get the desired number of candidates. The candidate components are then specified in a complementary format to the ideal component and an **adaptation model** is used to map the required interface to that provided by each candidate component. Test generation **(3)** is based on the ideal component specification to provide a consistent set of tests to run against all of the candidate components. Steps 2 and 3 are independent and can be carried out in any order, or in parallel.

Once the candidate components and the tests are defined, a series of activities is carried out for each candidate. The test sets from step 3 are adapted to the individual component **(4)**, based on the adaptation model from step 2. These tests are then executed and the results recorded **(5)**. At this point the test results and the information from the selection process are combined to evaluate each component **(6)**.

The components can then be ranked based on their test results and other suitability and context information **(7)**. The developer has control over the weighting of each recorded metric and can prioritise risk factors as suits the project or organisation. The results of the entire process are then summarised to produce a report **(8)** advising of the most suitable components, the reason for the choice, and the adaptations required for integrating the component into the target system.

4. Case Study

The purpose of this case study was to explore the feasibility of the CdCT process. We applied the process to a real-world problem to identify issues associated with each step of the process and the specifications, strategies and metrics required. We explore the selection of a component to provide scientific calculation functionality to the target system. The system provides the interface for the user to enter the information to set up the calculation, with the calculator component carrying out back-end calculations.

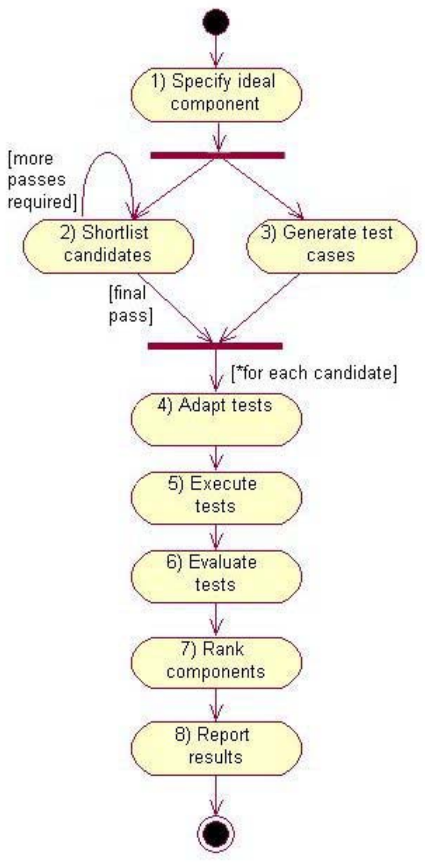
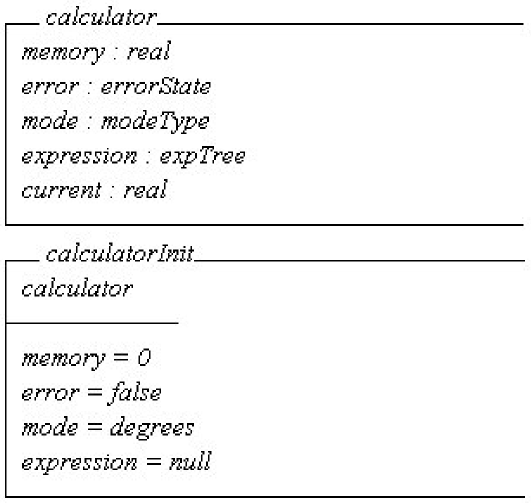


Fig. 1. Activity diagram for CdCT process

Step 1 : Specification of Ideal Component

Crucial to this selection process is the clear definition of the required component. This is done by providing a description of the ideal component, including context information.

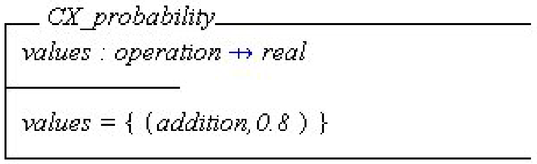
Context information recorded in the ideal specification includes the platform, programming language (desirable), memory usage (disk and RAM), required functionality and context information. Below are the Z schemas for the state and initialisation of the ideal component, and are based on published specifications for a simple calculator and generic trees [24].



```
calculator
memory : real
error : errorState
mode : modeType
expression : expTree
current : real

calculatorInit
calculator
memory = 0
error = false
mode = degrees
expression = null
```

Fig. 2. State and initialization schemas



```
CX_probability
values : operation ⇸ real
values = { (addition, 0.8 ) }
```

Fig. 3. Context schema – CX_probability

Mandatory operations are included in the specification and are indicators of the required interfaces. Additional information about the behaviour of the component is used for test generation and oracle functions, without implying that the candidate components have to use the same logic.

The application developer may have usage profiles or other information to guide test generation towards important functionality or input values. This information is recorded in predefined context schemas (Table 1) that are understood by the test generator. An example is given below, where the most common mathematical operation is known to be addition. Usage based tests can then be generated with a bias to addition whenever a mathematical operation is required.

Step 2 : Short-listing and Specification of Candidates

Information supplied for components, and software in general, is quite variable, making automated short-listing difficult for the time being. In some situations, automated selection may be possible for particular repositories, usually in-house software libraries for organisational use. We look at a real-world scenario, using commercial software repositories and component brokers. The selection process was carried out manually, but enacted using well-defined selection rules to simulate an automated process. The targeted maximum number of candidates is 7±2 for the manual short-listing process [25]. Automation will allow for these limits to be increased or removed, with the user having the option to set the number of candidate components returned.

The short-listing process gives the application developer a structured and repeatable approach to sourcing components. This saves time, allows for a wider search for candidates (through automation) and clear, traceable reasoning for selections made.

Table 2. Short-listing results

Site Code	Total available entries	First Pass	Second Pass	Third Pass	Final Pass
I	8000+ components	2	1/2	1/1	1/1
II	533 components	2	0/2	-	-
III	12,212 projects	113	16/113	7/16	7/7
IV	36725 projects	173	36/173	4/36	4/4
V	30,000+ titles	67	11/67	3/11	0/3
	~87,500 listings	**357**	**64/357**	**16/64**	**12/16**

The initial task was to select repositories to search for components. Five web sites were chosen, each offering access to software descriptions and implementations. Two of the sites were specifically component brokers, two were foundries for open source projects and one offers a large selection of freeware and shareware applications. The criteria for each pass of the selection process are taken from the ideal component metadata, focusing on the description and the environment/platform requirements. The first pass identified any software that included "calculator" in the description. This very broad criterion gave an indication of the number of possibilities the search would need to consider. A total of 357 software possibilities were returned for the five sites. The process could then use any part of the ideal component specification to reduce the number of values returned. Table 2 shows the number of possibilities remaining after each pass. The criteria for the second pass focused on ways to easily rule out a large proportion of the components based on metadata in the ideal and available components. The sweep criteria for each pass are given in Table 3. The second pass used criteria A-F to reduce the possibilities from 357 to 64. The third pass introduced criteria X-Z and also found some software that failed A-F upon further investigation. This resulted in sixteen matches. The final pass came as a result of gathering information to fully specify the candidates in CdCT format. The information uncovered in downloading and reviewing the software and its documentation exposed three possibilities that failed the selection criteria already used. There were also four duplicates, resulting in nine candidate components to fully specify and take through the evaluation process.

Table 3. Short-listing criteria

Code	Selection Criteria:
A	Description includes "scientific"
B	Is a "calculator application or component"
C	Not specific to X11 (or GTK/Gnome/KDE/Motif)
D	Not a specific calculator emulator
E	Environment is (Windows or O/S independent)
F	Has description
X	Has scientific functionality (not just basic arithmetic functions)
Y	Has a programmable interface (not just mouse/GUI)
Z	Has files released/can access web pages

Step 3 : Test Generation

The generation of tests for the selection and evaluation process is based on the Z specification for the ideal component and its context, and current strategies as described earlier. We follow an approach similar to [26] and [17], where specifications are transformed to disjunctive normal form, from which operations are generated to represent each partition of the input space.

For this case study, the tests were generated manually. The four categories of test cases target the functional areas: memory sequences, mode sequences, expression sequences and expression+memory sequences. These were pulled from the variables in the Z state schema and provide full coverage of the ideal component operations. Each variable had two partitions - valid and invalid, resulting in the pairs of test cases (e.g. 1 and 1a). These tests served to provide a quick assessment of the available functionality. Values from each partition were then substituted through a random test data generation process. Full descriptions of the tests are available in the case study documentation [21].

Test generation is the main application for the formalised specification of the ideal component. This specification of the required behaviour has many rewards, not only for generating tests to evaluate the component, but also for testing updates to components and to test the target system during development. Only the ideal component needs to be specified formally. From this specification we generate abstract test cases which are adapted for execution against each candidate component.

Step 4 : Test Adaptation

Test adaptation is carried out by combining the test sets from **step 3** with the adaptations from **step 2**. The result is a set of tests for each candidate component that exercise identical functionality and data for consistent tests across all components. The adaptations in this case were syntactical - there is a clear mapping between the generated tests and the actual transcript.

Test adaptation is currently a manual process and has two options for automation: to produce a set of inputs to suit the candidate component or to produce wrapper code to sit between the tester and the component. The second option would provide a useful head start in integrating the component and further investigation will be needed before making the decision.

Step 5 : Test Execution

We concentrate on abstract descriptions of the test cases to allow for variation in test environments. Future work will automate test execution using a test harness and the AGEDIS format for describing test sets and results [27]. The AGEDIS project is developing XML based test documents to aid portability of test scripts across programming languages and environments.

The tests were run manually against each of the components. The possible results were **pass**, **fail** or **not applicable** (where the functionality was not present). The test suite did not expose any failures - all the candidates passed the tests, where the functionality was available in the component.

Step 6 : Evaluation of Results

This step converts the raw test execution results, selection criteria and adaptation information to create a picture of each candidate's suitability. The test execution results are converted into a score indicating the performance of the component against functional and usage based testing (#tests passed/#tests in total). The selection process also provides useful information about the component's suitability, based on comparison with the ideal component specification. Another facet affecting the suitability of a component is the effort required to adapt the component to its target system. These pieces of information from steps 2 and 5 are now collated for each component, ready for ranking in **step 7**.

Table 4. Component metrics and ratings

Metric	**Result**	**Score (s)**	**Conversion**	**Rating**
% Features available	75%	7	s	7
Excess Features (# categories)	12	10	10-s	0
% Interfaces needing adaptation	100%	10	10-s	0
Maturity (years since first release)	>1	1	2x s	2
Maturity (stability)	4 (Beta)	4	2x s	8
Cost	free	0	10-s	10
Test results	75%	7	s	7
Simple Total				**34**
Simple Percentage				**49%**

An example of the case study results for a component is given in Table 5. Following the approach of Solberg and Dahl [28], the rating for each metric is a conversion from the raw results to a value in the 0-10 range. Higher ratings indicate component features are more suited to requirements. A simple indicator of the overall performance of the candidate component is given by summing the ratings or calculating the percentage of rating points achieved.

Step 7 : Ranking of Candidates

Given the ratings for each component against each metric, it is possible to compare and rank the components. By default, the CdCT process considers all metrics equally. There is a facility to add weightings to each metric to suit a particular project, or an organisation's quality or standards requirements. For example, an organisation may decide that the risk of immature/unstable software or of excess functionality is of high importance. It would then increase the weightings of these metrics to increase their effect on the rankings. A cutoff value may also be used for high priority metrics, disqualifying the candidate from the rankings. A simple weighted score sum [28] is used to determine the result for each component:

$$result = w1*s1 + w2*s2 + w3*s3 + \ldots + w7*s7$$

We investigated a number of combinations of weightings. Table 5 shows how the overall scores for three components are affected by varying the weightings of the metrics. In this case, C3 consistently achieves the highest scores

Table 5. Results of component ranking

Weighting Pattern	**C2**	**C3**	**C9**
Default - all equal	33	**44**	34
Tests have triple value, all else = 1	37	**64**	48
Features has triple value, all else = 1	37	**64**	48
Adaptation has triple value, all else = 1	33	**44**	34
Maturity has triple value, all else = 1	57	**64**	54
Risk factors tripled, all else = 1 (maturity, adaptation and excess features)	71	**72**	54

Step 8 : Report on Results

Once the candidates have been ranked, the resulting information is presented as a report on the process and advice on the most suitable component(s) for the problem being addressed. The report includes features and shortcomings of the component(s), and the adaptation required to integrate the component into the target system. It can also provide information on the short-listing process and criteria used for selecting candidates. It serves as the justification of the choice of component for inclusion in the system documentation.

4.1.1. Case Study Observations

The short-listing process exposed wide variation among vendors in terms of total software titles, candidates returned and the documentation provided. These differences will be recorded in a knowledge base to allow them to be taken advantage of. For example, site IV ordered the results by project activity. This meant that after the first 44 projects, the usual reason for failing the third pass was that there were no files available for the project. By taking the activity metric into account, it would have been possible to reduce the short-listing effort by 75%.

Sites I-IV had significant amounts of information about each piece of software. Site V had little documentation and meant that the developer web site had to be accessed for each possibility in the third pass. In terms of usefulness as an incoming component, sites I and II are targeted to the component market, and tend to be better documented. Unfortunately they did not have many components matching the case study criteria. Sites III and IV are aimed at encouraging open source development. Their projects may not be stable, but there are good options for reuse as interfaces are accessible. There are a large number of projects, varying in maturity and level of documentation. Site V mainly provides standalone applications, so little information about integrating the software is available. The site does offer a large number of titles and the available software may prove suitable for integration into other component-based systems.

5. Conclusion and Future Work

This paper has outlined our process for selecting and evaluating third party components. Use of third party components is hindered by such issues as how to source, select and test candidate components. Application developers need to be confident that they have the most suitable component for their system. This approach is aimed at developers sourcing third party components from external repositories. Such components come with varying levels of documentation. Our process provides a systematic approach for sourcing and selecting components. Automation of the process will save time, allow for a wider field of components to be considered, and gives traceable reasons for any choices made.

The application developer provides a full specification of the ideal component for use in the selection and evaluation process. This pushes most of the (specification) effort to the application developer, along with the flexibility and control of including context information and prioritising selection and evaluation criteria. We use specification-based test generation from the formal specification using Z notation. By including context information in the process, we are able to address issues with component testing.

An important benefit of generating tests from the ideal component specification is that the candidate components are then tested using the same test cases, allowing for a meaningful comparison of results. This is similar to the well-established approach to conformance testing using test suites. Beyond the component, the test cases can be used for testing the target system. In addition, the tests may be reused for regression testing of new components, or regenerated from an updated ideal component specification.

Future work is to carry out further case studies to gauge the suitability of the process to more complex components. We are planning for a highly automated process, which would be aided by a level of standardisation in component specifications. To further aid interoperability, we will adopt the ZML standard being developed for representing Z in XML [29]. Important focus areas will be the exploration of test generation techniques, evaluation metrics (e.g. performance) and the approach to combining the metrics for ranking. We are confident that the CdCT project will contribute to the use of third party components by adding confidence to the selection, sourcing and testing phases of development.

Glossary

application developer - an organisation or person who makes use of third party components in the development of new systems and applications.
component - "binary units of independent production, acquisition, and deployment that interact to form a functioning system" [1]

component broker - an organisation offering access (and purchase) of software components
component developer (user) - an organisation or person who develops components for application developers to integrate into new systems.

References

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

[2] Component Source. "Commercial Website." Accessed 5/6/03, from the World Wide Web: http://www.componentsource.com/, 2003

[3] J. Stafford and K. Wallnau, "Is Third Party Certification Necessary?". In: Proceedings of the 4th ICSE Workshop on Component-Based Software Engineering: Component Certification and System Prediction. Portland, Oregon USA May 3-4, 2003

[4] Active-X.COM. "Commercial Website." Accessed 3/6/03, from the World Wide Web: http://www.active-x.com/, 2003

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

[6] R. C. Seacord, D. Mundie and S. Boonsiri, "K-BACEE: Knowledge-Based Automated Component Ensemble Evaluation". In 27th Euromicro Conference 2001: A Net Odyssey (euromicro'01), September 04 - 06, 2001, Warsaw, Poland

[7] C. J. Fidge, "Contextual matching of software library components". In P. Strooper and P. Muenchaisri, editors, Asia-Pacific Software Engineering Conference, pages 297-306. IEEE Computer Society Press, 2002.

[8] Capability Maturity Model for Software (SW-CMM) [web page]. Accessed 12/4/2003, from the World Wide Web: http://www.sei.cmu.edu/cmm/cmm.html

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

[10] J. Morris, G. Lee, C. Parker, G. A. Bundell, and C. P. Lam, "Software Component Certification," *IEEE Computer*, 2001.

[11] 6th International Conference on Software Engineering (ICSE) Workshop on Component-Based Software Engineering: Automated Reasoning and Prediction. Portland, Oregon USA May 3-4, 2003

[12] G. J. Myers, *The Art of Software Testing*. New York: John Wiley & Sons, 1979.

[13] A. Orso, M. J. Harrold and D. Rosenblum, "Component Metadata for Software Engineering Tasks", In *Proc. 2nd International Workshop on Engineering Distributed Objects*, Davis, CA, November 2000.

[14] H. Yoon, B. Choi and J. Jeon, "A UML Based Test Model for Component Integration Test", *Workshop on Software Architecture and component, Japan*, 1999

[15] S. Sankar and R. Hayes, "Specifying and Testing Software Components using ADL", *Technical Report, Sun Microsystems*, April 1994

[16] H-M. Horcher, and E. Mikk "Test Automation using Z Specifications" In: B. Buth, R. Berghammer, J. Peleska(eds): "Tools for System Development and verification", Workshop, Bremen, Germany. BISS Monographs, Shaker Verlag,1996.

[17] S. Burton. "Automated Testing From Z Specifications," Accessed 27/5/02, from the World Wide Web: http://citeseer.nj.nec.com/burton00automated.html, 2000

[18] R. M. Hierons, S. Sadeghipour, and H. Singh, "Testing a System specified using Statecharts and Z", Information and Software Technology, 43 2, pp. 137-149.

[19] Dublin Core Metadata Initiative. "DCMI Website." Accessed 20/12/01, from the World Wide Web: http://www.dublincore.org/, 2001

[20] 7th International Conference on Software Reuse (ICSR7) Workshop on Component-based Software Development Processes Austin, Texas, April 15-19, 2002 http://www.idt.mdh.se/CBprocesses/

[21] V. Maxville. "research home page," [web page]. Accessed 6/9/03, from the World Wide Web: http://www.scis.ecu.edu.au/research/PhD/vmaxvill/, 2003

[22] World Wide Web Consortium. "homepage," [Web page]. Accessed 20/12/01, from the World Wide Web: http://www.w3.org/, 2001.

[23] J. M. Spivey, "A Guide to the Zed Style Option" ftp://ftp.comlab.ox.ac.uk/pub/Zforum/zguide.ps.Z, 1990

[24] R. Barden, S. Stepney, and D. Cooper, "Z in Practice", Prentice Hall, London. 1994

[25] G. A. Miller, "The magical number seven plus or minus two: Some limits on our capacity for processing information". Psychological Review, 63, 81-9, 1956

[26] J. Dick and A. Faivre, "Automating the Generation and Sequencing of Test Cases from Model-Based Specifications," presented at FME '93: Industrial-Strength Formal Methods, 1993.

[27] AGEDIS, "Automated Generation and Execution of Test Suites for Distributed Component-based Software", Accessed 20/11/02, from the World Wide Web: http://www.agedis.de/

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

[29] M. Utting, I. Toyn, Jing Sun, A. Martin, Jin Song Dong, N. Daley, and D. Currie, "ZML: XML Support for Standard Z". In: The 3rd International Conference of B and Z Users (ZB2003), 4-6 June 2003, Turku, Finland.