Towards an Effective Component Testing Approach Supported by a CASE Tool

Fernando Raposo da Camara Silva□□ Eduardo Santana de Almeida□ Silvio Romero de Lemos Meira□□

□Federal University of Pernambuco □UFPE□

□□SERPRO – Serviço Federal de Processamento de Dados□ Recife – Brazil

□C□E□S□A□R – Recife Center for Advanced Studies and Systems□ Brazil

frcs@cin□ufpe□br□{esa□□silvio}@cesar□org□br

***Abstract**□ Sometimes information about how to reuse a software component is ineffective or absent□ One of the main challenges of component testing is how can component consumers understand candidate components sufficiently in a way they can check if a given component fulfills its goal□ This is hard though□ because information about component behavior is limited to component consumers□ An approach to reduce the lack of information between component producers and component consumers is presented to improve understandability and support component testing activities□ The approach is covered by a CASE tool integrated in the development environment□*

□□ Introduction

Software componentization is one possible approach to promote software reuse□ As a consequence□ component□based development □CBD□ appeared not only as an action to standardize the construction of components with the premise of software reuse as well as an answer to the claim that CBD allows the reduction of cost and time to market□ while increasing software quality□

The literature contains several work related to CBD methods and approaches□ however□ the main consolidates CBD methods [D'Souza et al□ □□□□□ Atkinson et al□ □□□□ and Chessman et al□ □□□□] are more aware in demonstrating component development as a feasible approach□ while other important activities□ for instance related to quality and more specifically testing□ are sometimes neglected□

According to Apperly [Apperly □□□□]□ in CBD we have a produce□manage□consume process where producers are focused on producing and publishing components to be reused and consumers are concentrated on finding and reusing components to reduce development cost□ Component consumers need to be sure that the component they intend to plug to their systems fulfill their needs□ while component producers try to distribute □as much as possible□ easy to find□ easy to understand and easy to integrate components□

However□ one of the main barriers of CBD is related to component integration□ Component consumers cannot properly conduct tests to ensure that a candidate component does what it is intended to do before deciding upon its integration□ This limitation is a consequence of the limited access to information that is normally available to testers when working on a project totally developed in house □□

In external component integration events like meetings with requirements staff are unrealistic □resources such as functional requirements sheet □sequence diagrams or any type of documentation that describes what the component under test does may not be provided too □In addition □the source code of the component is frequently omitted sometimes due to legal restrictions □□

This lack of information can inhibit CBD and the benefits of software reuse because it increases the effort of component consumers to understand how to use a component in order to test it □Moreover □component producers cannot have its effort to develop a reusable component too increased because activities to produce information to support consumers were added □Thus □it can inhibit the use of CBD to develop reusable software □□

Thus □this paper presents a workflow of activities at the component producer side to support producers to provide information about the component to third party testers □and at component consumer side to aid consumers to understand a component in order to test it before its integration □In addition □these workflows are covered by a CASE tool integrated to the development environment □□

This paper is organized as follows □A motivation for component testing is presented in Section □□□Section □□presents some issues of current component testing approaches and describes the workflow of activities at both sides □component producers and component consumers □Section □□presents detailed description of the workflows and a tool integrated in the development environment to covering the activities of the workflows □A preliminary analysis is presented in Section □□□followed by related works in Section □□□Concluding remarks and future directions are presented in Section □□□

□□Motivation for Component Testing

It is mandatory to provide components with a minimum level of quality to promote software reuse and take the benefits provided by them [Councill □□□□] □According to Szyperski [Szyperski □□□□] □*“Testing of software components □and component based software systems □is possibly the single most demanding aspect of component technology”* □This means component testing is one of key aspects of components quality □□

However □components are heterogeneous by nature □There are open source components □COTS components □components presented in different programming languages □besides components presented with or without source code □In addition □the level and quality of information varies from component to component □□

Testing components does not mean simply to execute tests and correct defects □□While components can be used in different contexts □sometimes the context that the component producer has used to validate its component is different from the component consumer’s one [Gao et al □□□□□] □Component testing is less trivial than traditional COTS testing in which the acceptance criteria of the customer are clearly translated into acceptance criteria of functional and non functional requirements □□

In addition□ according to Krueger [Krueger □□□□]□ the second software reuse truism is □□*For a software reuse technique to be effective□ it must be easier to reuse the artifacts than it is to develop the software from scratch*□□ That truism encourages component consumers to reduce the time spent analyzing if a given component properly fulfills its goals□ however□ this decision can directly impact the quality of the component based systems which consumers intent to build□

As a result□ it is necessary to reduce the heterogeneity of information provided by producers to consumers in order to achieve effective component testing□ also it is necessary to support consumers when understanding and testing candidate components before its integration□ The creation of components hard to find□ complicated to understand□ difficult to adapt and poor testable could lead the reuse culture□ in a long term□ to never be well established□

□□Improving Software Components Testing

□□□□Component Testing Issues

Approaches to test components normally address a specific issue of component testing□ Some of them□ such as [Harrold et al□ □□□□□ Liu et al□ □□□□□ Beydeda et al□ □□□□]□ address the problem of the lack of information between component producers and consumers providing means to improve the quality level of information collected and provided by producers□ while others□ such as [Wang et al□ □□□□□ Bundell et al□ □□□□□ Gao et al□ □□□□□ Atkinson et al□ □□□□□ and Rocha et al□ □□□□] try to promote architectural solutions to construct components with high testability□ Next□ some issues of current component testing approaches are discussed□

Component misunderstanding problem□ According to a recent research on open source component □OSC□ integration by Merilinna et al□ [Merilinna et al□ □□□□]□ "*Primary problems in practice were issues concerning vertical integration and the lack or impreciseness of documentation*"□ This indicates that even if source code cannot be provided□ as well as requirements specification to support testing□ the focus of component testing approaches should be on providing useful information to consumers in a way they can understand how external components behave□ how to test them and how to adapt them to their needs□

Programming overhead of architectural solutions□ There are approaches that rely on easily testable architectures to overcome component testing difficulties such as [Wang et al□ □□□□□ Gao et al□ □□□□□ and Atkinson et al□ □□□□]□ They are founded on software testability that is considered by Voas et al□ [Voas et al□ □□□□] one of the three pieces of the reliability puzzle□ But exclusively architectural solutions have some shortcomings like its increased complexity□ and programming overhead associated□ memory usage issues□ maintainability issues and□ unless older components from repositories were re□ engineered to adapt their architectures to provide testability□ they have applicability limitations□ In addition□ among agile methods adopters□ that according to [Highsmith et al□ □□□□] consider the simplest solution usually the best solution□ complex architecture may find some resistance□

Customer acceptance criteria□ Some Built□in Testing approaches such as [Atkinson et al□ □□□□ and Wang et al□ □□□□] are founded in static data□ with a pre□defined set of test cases□ Consumers may change the input of the pre□defined test cases and check output data but this is not enough to validate a component□ Consumers have their own needs and expectations related to a candidate component □i□e□ this means they must be able to create their own tests to validate components under their own criteria and context of use□ not the one provided by the producer□

Lack of tool support□ Component testing is only part of a workflow of activities that aim at constructing a system with the benefits of software reuse□ Tools like Component Test Bench [Bundell et al□ □□□□] to verify test case pattern and FATESc tool [Teixeira et al□ □□□□] for structural tests are stand alone applications□ This may reduce the possibility to consumers elaborate tests linking the candidate component and other resources like libraries and other components□ However□ currently many development environments□ such as Eclipse□ and Together®□ can concentrate system modeling□ programming interface and testing□ A better integration with development environments would improve productivity on both producer and consumer sides□

□□□ Component Testing Workflow

Based on surveys of component testing [Beydeda et al□ □□□□ and Rehman et al□ □□□□]□ and the issues analyzed previously it can be noticed that the current approaches are not well integrated with the workflow of activities related to component testing□ In general they are focused on a specific issue□ Integration is important because it is the same bridge that connects□ in traditional software development□ the system architects to software programmers□ requirements staff to testers□ However□ this bridge is missing in CBD□ component producers and consumers can be complete independent teams without any communication channel□

Analyzing existent issues of component testing and solutions for component testing covered by existent approaches□ a workflow of activities performed by component stakeholders at both sides is presented followed by a description of each phase□

□□□□ Component Producer Workflow

Figure □□ presents the workflow of activities at component producer side using SADT notation [Ross□ □□□□]□ The workflow is not necessarily a waterfall□ because in practice□ some activities can be executed interleaved or skipped□ whereas others can be fully tool automated to reduce the time spent by producers to provide test information to consumers□

□□ **Collect Information**□ In this stage□ component producers collect resources available that can improve the understanding level of someone that was not involved with the development of the component□ Examples of this type of information can be requirements sheet□ sequence diagrams□ class diagrams or test cases among others□

□□ **Restrict Information**□ At this moment□ an analysis of the information collected is conducted□ If there are any restrictions of information to be provided to consumers□ for instance□ copyright restrictions or confidential data□ the information should be discarded□

□ http□□□www□eclipse□org□

□ http□□□www□borland□com□us□products□together□

Analyze Information This phase represents the tasks related to component producers to consolidate the data from previous activity This may compound the application of tools to extract specific data elaboration of component usage manuals creation of generic test cases or implementation of some testable infrastructure to be provided to consumers

Publish Information Corresponds to the act of attaching to the component the information to be provided to consumers or publishing somewhere else the information that third party testers may use to validate the component

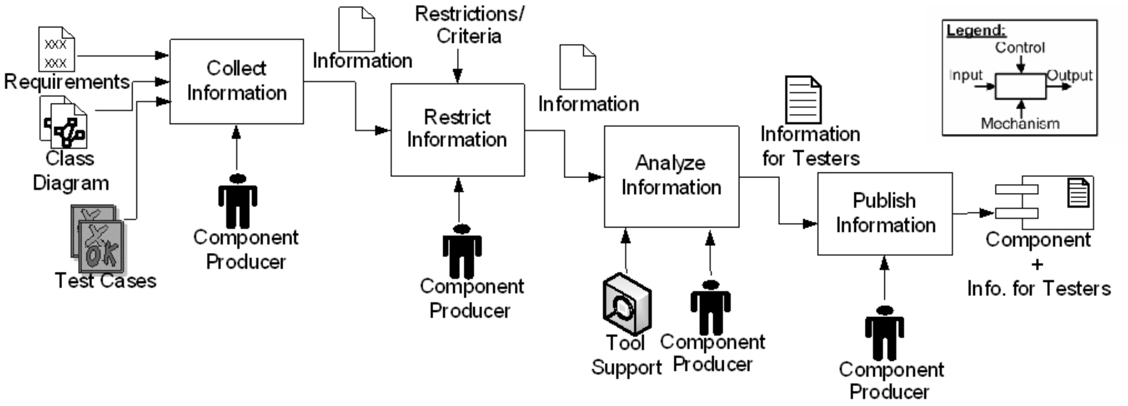


Figure – Workflow of Activities at Component Producer Side

Component Consumer Workflow

Figure presents the workflow of activities at component consumer side Some activities can be executed simultaneously and not exclusively by testers and others can be fully tool automated tool supported to reduce the effort of component consumers

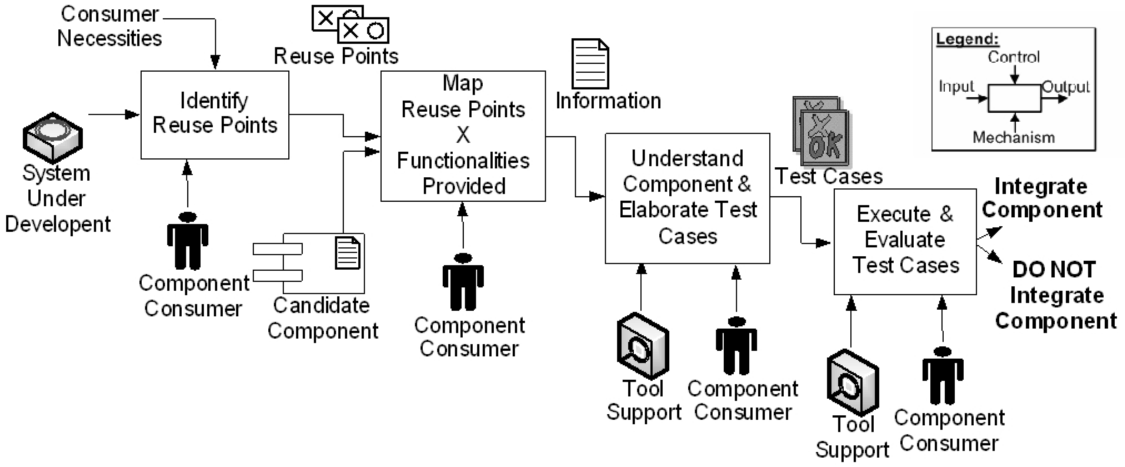


Figure – Workflow of Activities at Component Consumer Side

Identify Reuse Points In this phase component testers identify where the system under development ends and where the component under test will be plugged to the system For instance for an on line store system after customers complete registration an e mail is sent to them with password and other information to confirm the provided data□The reuse point identified will be the point to invoke an email component to send the e□mail after user registration□□

□□**Map Reuse Point to Component Functionality**□At this moment□a mapping of the selected reuse points of the system under development and functionalities provided by the candidate component will be performed□Detailed component description provided from producer is a key aspect because they will help consumers to check if the candidate component has chances of fulfilling its needs□□

□□**Understand Component and Elaborate Test Cases**□In this phase□component testers should understand candidate component usage□method call dependencies and pre□conditions of operation□Based on this understanding□testers will analyze necessities of some glue code or stubs to create tests to validate basic functionalities of the candidate component and tests with their own acceptance criteria to check integration with the system under development□□

□□**Execute and Evaluate Test Cases**□The test cases created are executed and the results are evaluated□□

□□Towards an Effective Component Testing Approach

In order to overcome the difficulties of component testing we suggest an integrated approach incorporated to component development focused on both sides□It covers the activities performed by component producers to prepare a component to be tested elsewhere□and also supports component testers at consumer side to understand the candidate component and analyze it against consumer's acceptance criteria□□

A tool focused on reducing the effort of both stakeholders for testing components□component producers and consumers□and integrated to the development environment was implemented too□It covers activities from the defined component test workflows at both sides and applies a combination of techniques□automatic□computer□assisted and manual□to reduce programming overheads□Next sections will describe in detail how to support component testing activities□□

□□□Providing Information to Component Consumer Tester

Current component testing approaches present a variety of information types that can be captured by component producers for further testing□[Beydeda et al□□□□□] proposed finite state machine representation□[Teixeira et al□□□□□] capture data for coverage analysis and [Wang et al□□□□□] create parameterized test cases□However□before consumers are able to create their own test cases to validate a component they must understand their functionalities□dependencies among interface method calls and usage assumptions□This demonstrates that the abstraction level provided to consumers should be at functionality level in spite of test case level□□

Our tool named PUG□*Producer Usage Generator*□is an Eclipse plug□in that accepts as input source code snippets to capture information about functionalities of the component□Its strategy is to capture as much as possible information producers are able to provide to consumers combining an algorithm to capture usage information and functionality descriptions provided by producers□ The information collected is consolidated in a standard format XML file that can be attached to the component before its publishing□Its usage has three steps that can be associated to the three last steps of the presented workflow □Figure □□□at component producer side □the first two □we consider to be performed manually □□Next □how the workflow at component producer side is covered by the tool is presented □□

i□□Usage Capture□PUG tool parses methods from provided code snippets and uses an algorithm similar to program slicing approach □as described by Harrold et al □[Harrold et al □□□□□] □and component retrospectors as presented by Liu et al □[Liu et al □□□□□] □It extracts functionality descriptions from JAVADOC □dependencies among variables and correct order of method calls to accomplish a functionality of a component □It always skips unrepresentative usage commands like `System□out□println` □or local variable definition □for instance □□

To reduce programming overhead □we suggest producers to reuse high level component unit tests with little adjustments in case they were created during component development □High level unit tests are those conceived while producers are developing the component to validate general functionalities from component's main interface □not the ones to test `getXXX□□` and `setXXX□□` methods of a single class □□

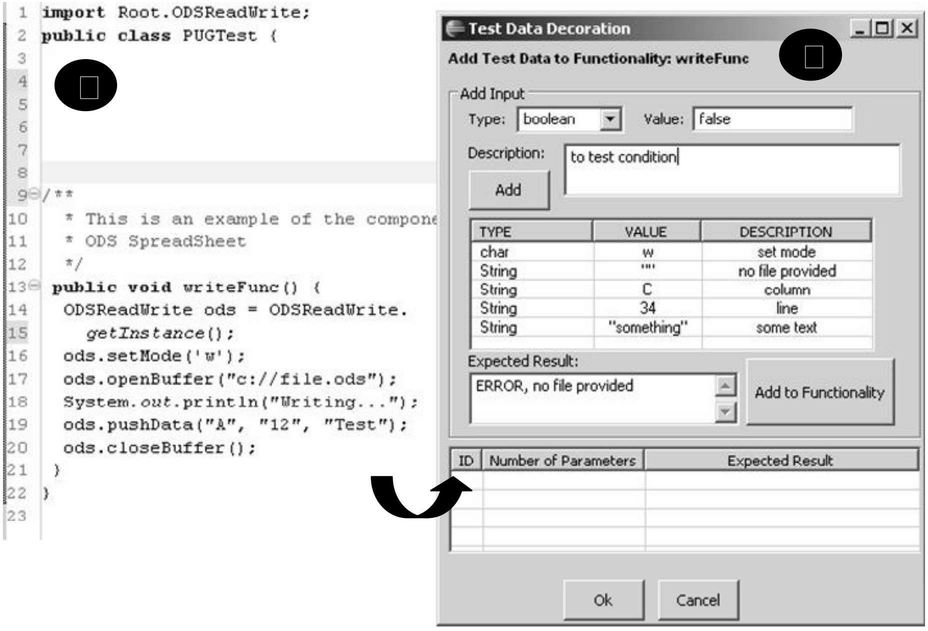
ii□□Additional Information Decoration□□After some functionality usages were captured by the algorithm □the tool enables producers to decorate the captured functionalities with additional information in an input form □This information is data that affects the state of the functionalities and can be used by consumers to understand the behavior of the component and create their own test cases □For instance □in an email component □□for `sendSimpleMessage` functionality □an input address such as "`test@@address`" is classified as "incorrect usage" that posses the message □ "INVALID EMAIL ADDRESS" as expected result □This additional information is optional to producers □however □according to IEEE Standard for Software Test Documentation [IEEE Std □□□□□□□] □textual descriptions are generally provided to testers □□

iii□Information Publishing□After usage information and additional information were collected □the data is consolidated in a XML file called *Usage Manifest* that the PUG tool generates in an area specified by the producer □XML is used by other approaches to format the data provided to consumers □However □PUG tool has the advantage to generate the file automatically whereas other approaches such as [Bundell et al □□□□□] suggest creating the XML file using a XML editor □Finally □producers can attach the *Usage Manifest* file to the component object code □for instance □the □*jar* file □before publishing the component □□

Figure □□□presents how component producers can capture information to be provided to consumers □□The source code snippet □at Figure □□□mark □□□specifies the usage of a component that has a functionality to write a string of characters in a cell of an *OpenOffice* spreadsheet file □This snippet is a JAVA method inside a JAVA class from where the algorithm captures JAVADOC as description of `writeFunc` functionality and the ordered sequence of method calls to successfully execute a write operation in a cell □The `println` command will be skipped by the algorithm as well as any other non necessary information □□

After the algorithm is processed □a window □at Figure □□□mark □□□appears to let producers decorate the functionality captured with possible input groups and its expected results It is possible to associate more than one input group to a functionality and type value and description to each input

The flow is concluded with the tool generating a `usage manifest xml` file which producers can attach to its component before publishing to market or open source repositories



```
import Root.ODSReadWrite;
public class PUGTest {





/**
 * This is an example of the compone
 * ODS SpreadSheet
 */
public void writeFunc() {
  ODSReadWrite ods = ODSReadWrite.
    getInstance();
  ods.setMode('w');
  ods.openBuffer("c://file.ods");
  System.out.println("Writing...");
  ods.pushData("A", "12", "Test");
  ods.closeBuffer();
 }
}
```

Add Test Data to Functionality: writeFunc

Add Input

Type: boolean

Value: false

Description: to test condition

TYPE	VALUE	DESCRIPTION
char	w	set mode
String	""	no file provided
String	C	column
String	34	line
String	"something"	some text

Expected Result: ERROR, no file provided

ID	Number of Parameters	Expected Result

Figure – Preparation of a Component to be Tested Elsewhere

Supporting Component Consumer Tester

Approaches to directly support consumers to test external components are presented from static built in tests [Wang et al] to frameworks such as [Bertolino et al] to execute a fixed test suite with consumer acceptance criteria over different candidate components In some cases they can contribute to validate a candidate component but on the other hand they are focused solely at specific activities such as like test case creation and test execution but not the whole process

According to Andrews et al [Andrews et al] component testers must understand how a component behaves in order to test it [Boehm et al] stressed that understanding software components is a specialized case of software understanding It is specialized since the objective is not to understand the code itself but how the component works in high level how it can be re used and how it needs to be adapted

In order to support component consumers to understand and test a component and assuming they have received a candidate component with a *Usage Manifest* attached a tool integrated with Eclipse development environment was implemented The tool called CTP *Component Test Perspective* combines visual and analytical support to guide testers at component consumer side to validate a component

CTP covers steps of the defined workflow of activities at component consumer side It has features to register reuse points of the system under development a pattern verifier to check if a test case under construction contains the commands to accomplish a functionality a visual representation of the functionalities provided in the usage manifest functionalities to improve component understanding level and code assist support to facilitate when testers are creating integration and basic test cases to validate a candidate component Next the usage of CTP is described in detail

i Reuse Points Identification The place in a system under development where consumers should plug external components can already be clearly specified in a form of internal interfaces Thus only some glue code can be necessary to invocate functionalities of the candidate component However given the current diversity of CBD methods these reuse points may be missing or not clearly defined yet They can vary from functional requirements described in natural language to use cases that were elaborated but not realized This issue can impact on the quality of the test cases created specifically those focused on testing the integration of the component and the target system To support the identification of reuse points that consumers should conduct CTP provides means to users register the exact reuse points that should be covered by external candidate components

In addition CTP allows users to register their expectations related to each reuse point These expectations can be restrictions that the functionality of the component to be plugged should address or non functional requirements the candidate component must realize These restrictions are important to aid the definition of expected results of the test cases created In Figure □□ is presented how consumers can register reuse points of the system and expectations that external functionalities should cover of each reuse point

ii Mapping Reuse Points to Functionalities Provided After the parts of the system where external components can be plugged are identified an association to functionalities of the candidate component that probably addresses the reuse points identified should be conducted This is important for two reasons The first reason is that generally not all the functionalities of a candidate component are used by consumers but a subset from the total of functionalities provided The mapping identifies the functionalities of the candidate component that should the tested to check if they work as designed from the functionalities that should be discarded to be tested For instance in an email component if consumers just want a component to send simple text messages only `sendSimpleMessage` should be tested whereas a functionality to send emails with attachments for instance `sendMessageWithAttachment` should be ignored The second reason is that the mapping identifies interactions between the system and candidate components This can help consumers to identify future integration tests

CTP tool supports as depicted in Figure □□ the mapping of what is provided by the candidate component to requirements of the system under development In addition types of strategy to test a functionality of the candidate component can be registered

iii Component Understanding and Test Case Creation Next consumers should understand how candidate components work to avoid component misuse problem According to [Liu et al □□□□] component misuse problem happens when a component is used in a way different from what the component producer expects For instance if a hypothetic component χ has methods ***a*** □□ and ***b*** □□ in its main interface and the invocation of method ***b*** □□ reads a variable that must be set by method ***a*** □□ beforehand this is a case of dependency between ***a*** and ***b*** If the consumer is unaware of that dependency the lack of clarity may lead to component failure Eventually component consumer will reject component ***χ*** The information provided from producers about the component is fundamental to reduce the lack of information between component producers and consumers

After component usage is internalized by consumers they should elaborate test cases to validate the provided functionalities and the interaction between the system and the candidate component

CTP tool uses the *Usage Manifest* provided within the candidate component to support component consumers to understand the functionalities provided and to elaborate test cases Testers can access information about a component by accessing features that provide **Component Information** to access different types of information about the component including its main interface and functionalities descriptions **Test Template** to be presented to a test template of the functionalities and **Test Data** to access the input data if the producers have decorated the *Usage Manifest* with additional data

In addition CTP provides pattern verification to check if the set of method calls that represent correct usage of a functionality is present in the test case under construction This functionality is useful to support testers when many method calls are required to be executed in some sequence due to variable or context dependencies Also when creating integration tests it helps testers to ensure the adherence of the test case to the functionality under test

Test case creation is also aided by code assist support Code assist is present in many development environments It reduces the typing analytical effort of programmers providing suggestions of probable next commands to be written However one issue of content assists is the generality of the suggestions provided normally in lists with many items CTP manipulates Eclipse’s content assist to provide better suggestions based on the functionality selected in the tree view

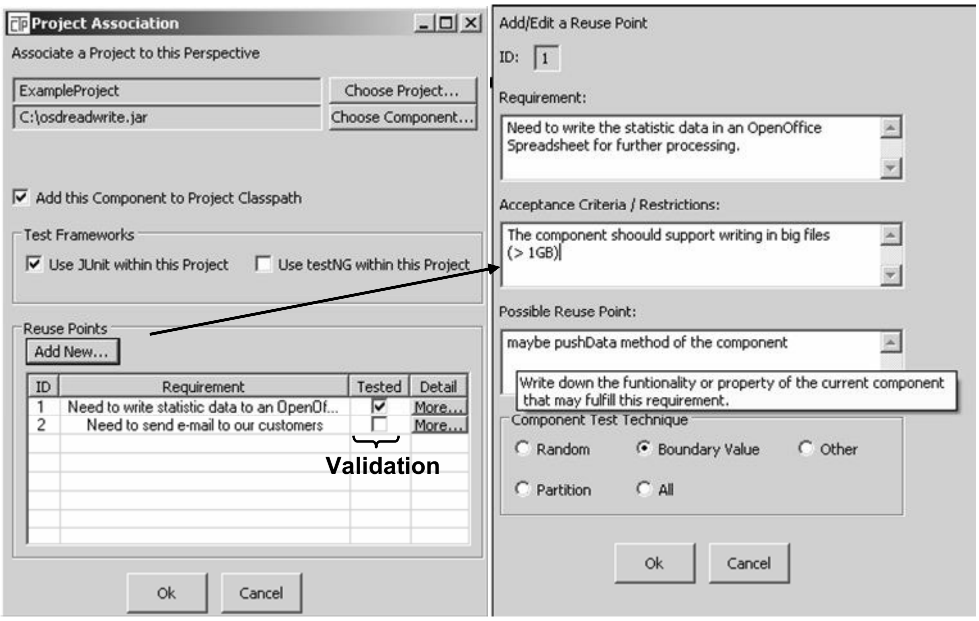
iv **Test Case Execution and Evaluation** Finally consumers should execute the elaborated test cases and perform an analysis of the results The results can indicate that the candidate component fulfills the quality standards of consumers and can be considered to be integrated to the system or on the contrary can be discarded

To execute test cases and evaluate results existing frameworks such as JUnit and testNG can be applied with CTP completing all the workflow activities related to component testing

Preliminary Analysis

It is a consensus that quality plays an essential role to stimulate software reuse and attack the “*Not Invented Here*” syndrome However there is no defined standard that tests components and assures its quality

To analyze the efficiency and effectiveness of the proposed workflows in conjunction with the tools we have an on going experiment As preliminary results we can enumerate the following



Project Association

Associate a Project to this Perspective

ExampleProject — Choose Project...

C:\osdreadwrite.jar — Choose Component...

☑ Add this Component to Project Classpath

Test Frameworks

☑ Use JUnit within this Project ☐ Use testNG within this Project

Reuse Points

Add New...

ID	Requirement	Tested	Detail
1	Need to write statistic data to an OpenOf...	☑	More...
2	Need to send e-mail to our customers	☐	More...

Ok Cancel

Add/Edit a Reuse Point

ID: 1

Requirement: Need to write the statistic data in an OpenOffice Spreadsheet for further processing.

Acceptance Criteria / Restrictions: The component shoould support writing in big files (> 1GB)

Possible Reuse Point: maybe pushData method of the component

Write down the funtionality or property of the current component that may fulfill this requirement.

Component Test Technique

○ Random ● Boundary Value ○ Other

○ Partition ○ All

Ok Cancel

Figure – Linking a Candidate Component to a JAVA Project

- The main advantage of our approach in comparison with other approaches is to cover the activities related to testing both component stakeholders producers and consumers
- Candidate components can be selected in a repository for instance by natural language descriptions However its specificities will only arise when mapping its provided functionalities to consumer necessities and then creating integration tests This activity is valuable information that is covered by the workflows and is not explicitly addressed by other approaches
- The proposed workflows are architecture independent Thus they can be applied independently of programming language or CBD process adopted
- Unlike other component test approaches based on self testability where most of the effort of testing is concentrated on consumer to implement self testable components our approach can be applied to components older than our approach legacy components
- Although information about the component can be provided in interface specifications or user manuals the approach is focused on providing minimal resources to support component testing independently of any other information that can be provided because other information may be out dated or incorrect
- Another benefit in comparison with other approaches is the use of tools at both sides to aid stakeholders This can reduce the effort of producers to prepare a component to be tested elsewhere and also reduce the effort of consumers to create integration tests checking how candidate components and the system under development work together

Related Work

A number of researches presented approaches attacking the problems related to component testing One possible classification of component test approaches as suggested by [Beydeda et al □□□□] is to classify approaches in two categories related to how they take into account the lack of information between component producers and consumers □

Approaches focused on the causes of lack of information try to minimize the dependence of component consumers on the information provided by component producers □ In this way the general strategy behind is to aggregate valuable information to the component in order to facilitate test activities at component consumer side □ [Liu et al □□□□] proposed an algorithm to capture information related to the usage of the component inside source code □ Similarly □ [Harrold et al □□□□] suggested the use of tools to capture component summary information like program dependencies among its statements □ information about exceptions handling that can help consumers to elaborate test cases and data flow information and to measure test suite coverage □

On the other hand □ approaches aiming at the effects of lack of information to support testing at component consumer side try to increase component testability by adding executable test cases that are built in the component together with the normal functions □ or try to equip the component with a specific testable architecture that allows component consumers to easily execute test cases □ [Wang et al □□□□] presented the *Built in Test* approach that is based on the construction of test cases inside component source code as additional functionalities □ In the same way □ [Gao et al □□□□] proposed the testable beans approach □ A testable bean has two parts □ one containing the functionalities that can be reused and another supporting component testing □

Analyzing current approaches □ it can be noticed that current approaches only address specific activities of the workflow related to component testing such as information collection or test case execution not the whole workflow starting at component producers and ending at component consumers □

□ Concluding Remarks and Future Directions

Theoretically □ a component has the known benefit of reliability because the use of a component in several systems increases the chance of errors being detected and strengthens confidence in that component □ However □ after examples of reuse with catastrophic results such as Ariane project □ [Jézequel et al □□□□] □ component consumers must have confidence in the component they plan to reuse □ Nowadays □ the reuse community is active □ and focused on solving this issue □

Even so □ it is clear that □ to evaluate components properly and guarantee reliability □ component testers has tremendous difficulties of developing test suites for candidate components due to the lack of low level understanding □ According to Weyuker [Weyuker □□□□] □ necessary insights are not available and lack of access to needed artifacts prevents certain types of testing □

We have presented in details two workflows that describe necessary activities to be conducted by producers to prepare a component to be tested by third party □ and the activities performed by component consumers to elaborate and execute test cases to support de decision of integrating candidate components to a system under

□ RESAFE' □□ International Workshop on Software Reuse and Safety development In addition tools integrated in the development environment were developed to support both component producers and consumers to accomplish the presented workflows

Beyond our preliminary analysis our work is not finished yet An experiment is being developed to measure the impact when applying the proposed workflows combined with the tools at both sides

Acknowledgements

This work is sponsored by Brazilian Agency CNPq process number

References

Andrews A Ghosh and S Choi E “A Model for Understanding Software Components” th IEEE International Conference on Software Maintenance pp

Apperly H “Component Industry Metaphor In Component Based Software Engineering Putting the Pieces Together” Addison Wesley pp

Atkinson C and Gross H G “Built In Contract Testing in Model Driven Component Based Development” th International Conference on Software Reuse

Atkinson C Bayer J Laitenberger O and Zettel J “Component Based Software Engineering The KobrA Approach” First Software Product Line Conference SPLC Kluwer International Series in Software Engineering and Computer Science pp

Bertolino A and Polini A “A Framework for Component Deployment Testing” th International Conference on Software Engineering pp

Beydeda S and Gruhn V “State of The Art in Testing Components” rd International Conference on Quality Software pp

Beydeda S and Gruhn V “An Integrated Testing Technique for Component Based Software” ACS IEEE International Conference on Computer Systems and Applications pp

Boehm B W Brown J R Kaspar H Lipow M MacLeod G J and Merritt M J “Characteristics of Software Quality” North Holland Publishing Company

Bundell A Lee G Morris J and Park K “A Software Component Verification Tool” International Conference on Software Methods and Tools pp

Chessman J and Daniels J “UML Components A Simple Process for Specifying Component Based Software” Addison Wesley Publishing Co pp

Councill W T “Third Part Testing and the Quality of Software Components” IEEE Software Vol No pp

D’Souza D F and Wills A C “Objects Components and Frameworks With UML The Catalysis Approach” Addison Wesley Publishing Company pp

Gao J Gupta K Gupta S and Shim S “On Building Testable Software Components” st Int Conference on COTS Based Software Systems pp

Gao J Tsao J H S and Wu Y “Testing and Quality Assurance for Component Based Software” Artech House pp

Harrold M J Liang D and Sinha S “An Approach to Analyzing and Testing Component Based Systems” st International Conference on Software Engineering pp

Highsmith J and Cockburn A “Agile Software Development The Business of Innovation” Computer vol no pp

IEEE Std IEEE Standard for Software Test Documentation

Jézequel J M and Meyer B “Design by Contract The Lessons of Ariane” Computer vol no pp

Krueger C W “Software Reuse” ACM Computing Surveys vol no pp

Liu C and Richardson D “Towards Discovery Specification and Verification of Component Usage” th IEEE International Conference on Automated Software Engineering pp

Merilinna J and Matinlassi M “State of the Art and Practice of Open Source Component Integration” nd EUROMICRO Conference on Software Engineering and Advanced Applications SEAA pp

Rehman M J Jabeen F Bertolino A and Polini A “Software Component Integration Testing A Survey” Journal of Software Testing Verification and Reliability STVR Vol No June pp

Rocha C R and Martins E “A Method for Model Based Test Harness Generation for Component Testing” Journal of the Brazilian Computer Society v pp

Ross D T “Structured Analysis SA A Language for Communicating Ideas” IEEE Transaction on Software Engineering Vol No pp

Szyperski C “Component Software Beyond Object Oriented Programming” Addison Wesley pp

Teixeria V S Delamaro M E and Vincenzi A M R “FATESc A Tool to Support Component Structural Tests” In Portuguese I Brazilian Symposium on Software Components Architectures and Reuse pp

Voas J M and Miller K W “Software Testability The New Verification” IEEE Software vol no pp

Wang Y King G and Wickburg H “A Method for Built in Tests in Component Based Software Maintenance” rd European Conference on Software Maintenance and Reengineering pp

Weyuker E J “Testing Component Based Software A Cautionary Tale” IEEE Software Vol No pp