Testing Coverage Analysis for Software Component Validation

Raquel Espinoza
San Jose State University

COMPSAC 2005
Edinburgh, Scotland
July 26-28, 2005

Agenda

- Introduction
- Related Work
- Test Model Concept
- Coverage Criteria
- Tool Architecture
- Application Example
- Conclusion
- Recommendations

Introduction

- **Motivation:** support widely used method of incorporating reusable software components to develop larger software systems to reduce development cost, effort and time

- **Problem:** How to adequately test these large software systems prior to release
  - If individual reused software components are not adequately tested, it may be difficult and time consuming to isolate the source of a problem

- **Solution:** Begin testing at the component level to weed out component quality problems and address reuse context issues prior to incorporation

- **Proposed Method:** A model-based coverage analysis method to address adequate testing of a components API in the validation for its reuse

Related Work

- S. H. Edwards define a black-box test model, known as flow graphs, derived from a component's specification and applied analogues to traditional graph coverage techniques to develop a test set generation approach.
- D. Hoffman and P. Strooper define a state-based test model, known as a testgraph, to model a class-under-test (CUT) as a state machine and applies graph traversal techniques for test input generation and develops driver and oracle classes to support test execution.
- S. Beydeda and V. Gruhn define a test model derived from a class specification and implementation and present it as a control flow graph to support structural testing techniques, such as coverage criteria-based techniques, for use in test case generation.
- D. S. Rosenblum defines two formal test models that address adequate unit testing and integration testing of components. These models are defined based on the subdomain-based test adequacy criteria defined by Frankl and Weyuker.

Proposed Method

- A model-based coverage analysis method for a component API to address adequate testing issue in validation for reuse
  - Define a test model that represents a component's API interface
  - Identify coverage criterion for a test model
  - Dynamically analyze and monitor test coverage by evaluating test scenarios to determine percent of coverage provided

Academic Contribution

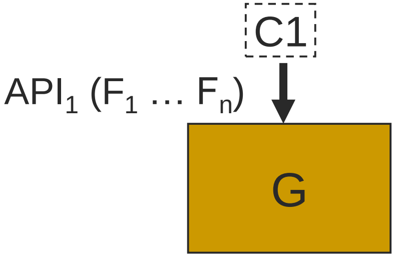
- To establish a dynamic test coverage analysis method that provides identification of higher quality test sets that can offer adequate test coverage at the component level
- To offer a more effective testing method for a components API

Test Model Concept

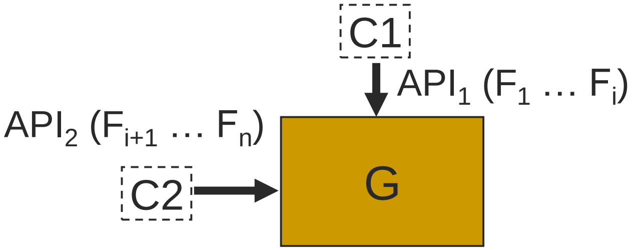
- A test model represents a component’s API interface and semantics that define constraints and usage of the interface
    - Each accessible interface method is defined as a “*node*”
    - Each possible interface method call between a pair of nodes is defined as an edge or “*link*”
    - A link may be “*conditional*” when an accessible element or state of the component represents an operation constraint of the interface method call

Component API Interface

- **Single API**
  - API has a complete set of accessible methods
  - API defines usage by another system component



- **Multiple APIs**
  - Each API has subset of accessible methods
  - Each API defines usage by different system components



Test Model Notation

- Analytical expression: $G = (F,E)$
  - G: test model for a component
  - F: set of all nodes
  - E: set of all links
- F_i is accessible interface method of G
- $E_n = (F_i, F_j)$ describes the method access sequence of F_i then F_j

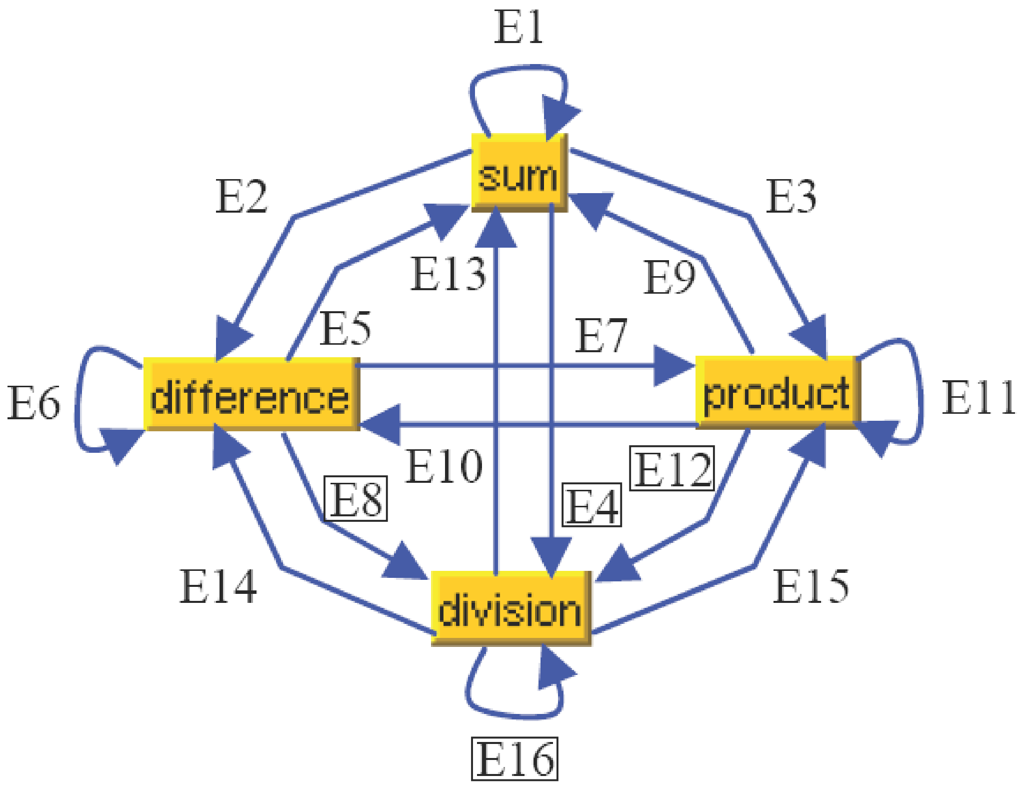
Test Model Representation

- Types
  - Component Functional Access Graph (**CFAG**): visual representation of the test model structure
  - Dynamic Component Functional Access Graph (**D-CFAG**): visual representation of the execution sequence of a test scenario for a given CFAG

- Purpose
  - To assist engineers in defining various test criteria
  - To facilitate automatic test generation
  - To facilitate test coverage analysis

CFAG Example:

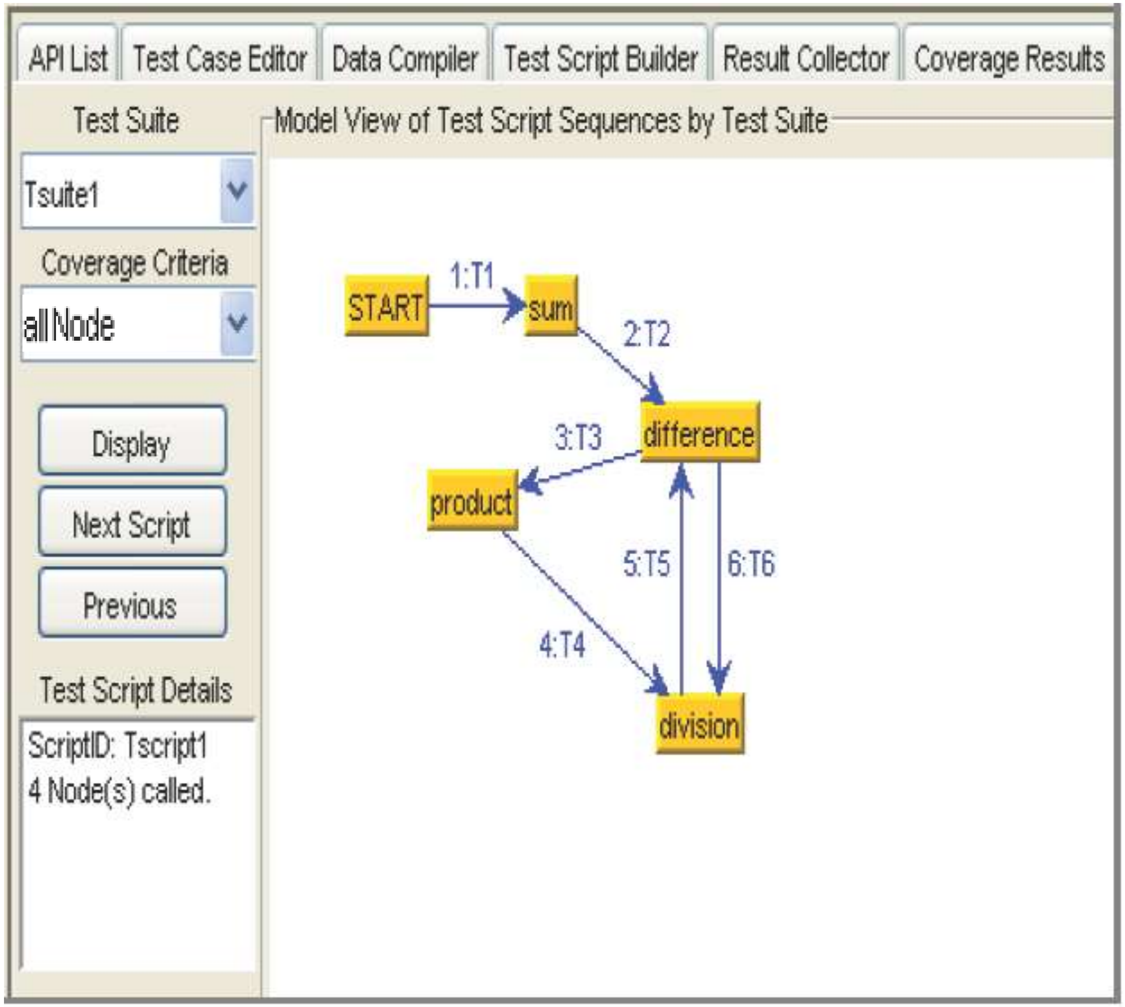
Simple Calculator Component



- Single API
- **Nodes:** sum, product, difference, division
- **Links:** E1 thru E16
- **Conditional links:** E4, E8, E12 and E16 where the condition addresses division by zero
- **Total links:** n^2 where n = number of nodes

D-CFAG Example:
Simple Calculator Component

- One possible test scenario with 6 test cases
- All 4 nodes exercised
- Only 5 out of 16 links exercised



Test Suite: Tsuite1

Coverage Criteria: allNode

Model View of Test Script Sequences by Test Suite

Test Script Details

ScriptID: Tscript1
4 Node(s) called.

Coverage Criteria

- Metric used to assess the effectiveness of a set of tests to exercise the functionality offered by a software component for the purpose of exposing defects
- Coverage criteria target specific characteristics of the software component’s structural or behavioral properties
- Coverage criteria considered:
  - Node coverage criteria for each accessible method in an API interface
  - Link coverage criteria for each link between two nodes
  - Conditional link coverage criteria
  - Path coverage criteria for API access sequences between the nodes

Coverage Criterion

- **Node:** achieved for node F_i in G if and only if its adequate test set TF_i has been exercised
- **All-node:** achieved if and only if every node F_i in G has achieved its node coverage criterion
- **Link:** achieved for link $E_n = (F_i,F_j)$ in G if and only if F_i has been exercised at least once using the test set TF_i followed by exercising F_j using the test set TF_j
- **All-link:** achieved if and only if every node E_n in G has achieved its link coverage criterion
- **Condition-link:** achieved for conditional link E_i in G if and only if E_i is exercised with two test cases such that both TRUE and FALSE conditions are tested
- **All-condition link:** achieved if and only if every conditional node E_i in G has achieved its condition-link criterion
- **Path:** achieved for P_k in G where $P_k = (E_i, E_j \ldots E_n)$ if and only if P_k is exercised at least once in a sequence E_i, $E_j \ldots E_n$ by a test script in the test set T for G.
- **Minimum-set path:** achieved if and only if there exists a path set P (for E_i to E_n) which covers all nodes and links reachable from E_i to E_n and is traversed by test scripts in test set T.

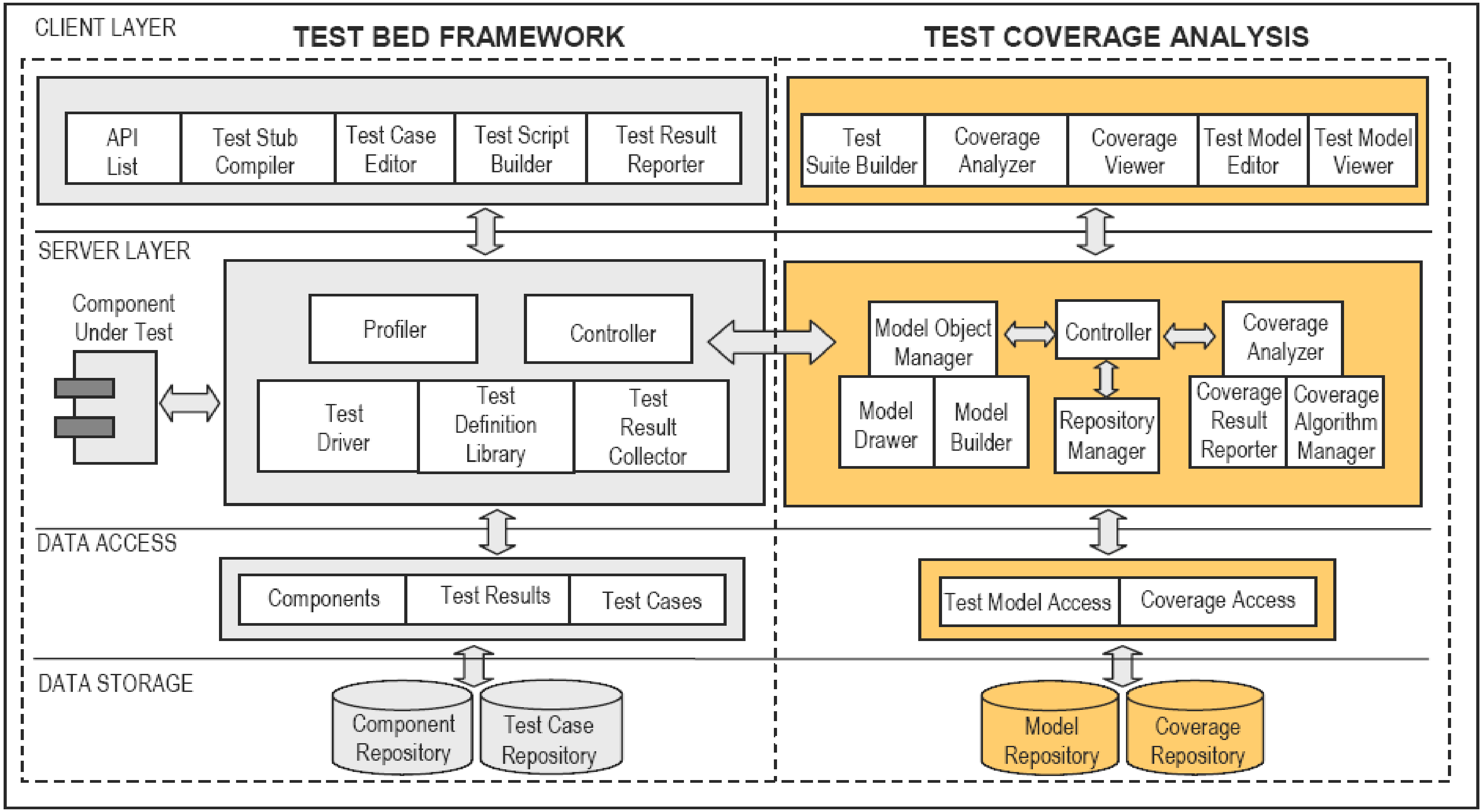
Applying Test Coverage to API

- Component C has a single API interface and black-box test set T with N test scenarios
  - **CFAG** for C is **G = (F, E)**
  - **i-th D-CFAG** for C that represents the i-th test scenario in T from 1 to N is **G**$_i$ **= (F[i], E[i])**

- Determine set of nodes and links achieving desired coverage using union operation
  - Covered-Node-Set = F[1] U F[2] U … U F[N]
  - Covered-Link-Set = E[1] U E[2] U … U E[N]
  - Uncovered-Node-Set = F – Covered-Node-Set
  - Uncovered-Link-Set = E – Covered-Link-Set

Tool Architecture

- Incorporated concepts into an automated testing application to provide test model generation and coverage analysis



Tool Functionality

- Test coverage analysis
  - Automatic generation of test models (CFAG)
  - Editing of test models
  - Creating test suites (sets of test scenarios)
  - Setup of the coverage analyzer for selected criterion
  - Executing of the coverage analysis
  - Viewing test scenario sequences (D-CFAG)
  - Viewing of coverage results
  - Storing test models and coverage results

- Test-bed framework
  - Loading profiles of component’s API
  - Creating test case and test scripts
  - Executing test scripts on the loaded components
  - Viewing test result

Application Example:
BTree Component

- Single API
- Component includes two classes
  - BSTree: API interface
  - BSTNode: object class

- Test set T containing three test scenarios T_1, T_2 & T_3 generated to exercise in different ways the inserting elements, removing elements and verifying BTree structure after insertion/removal
  - T_1 execute 18 method calls
  - T_2 & T_3 execute 13 method calls

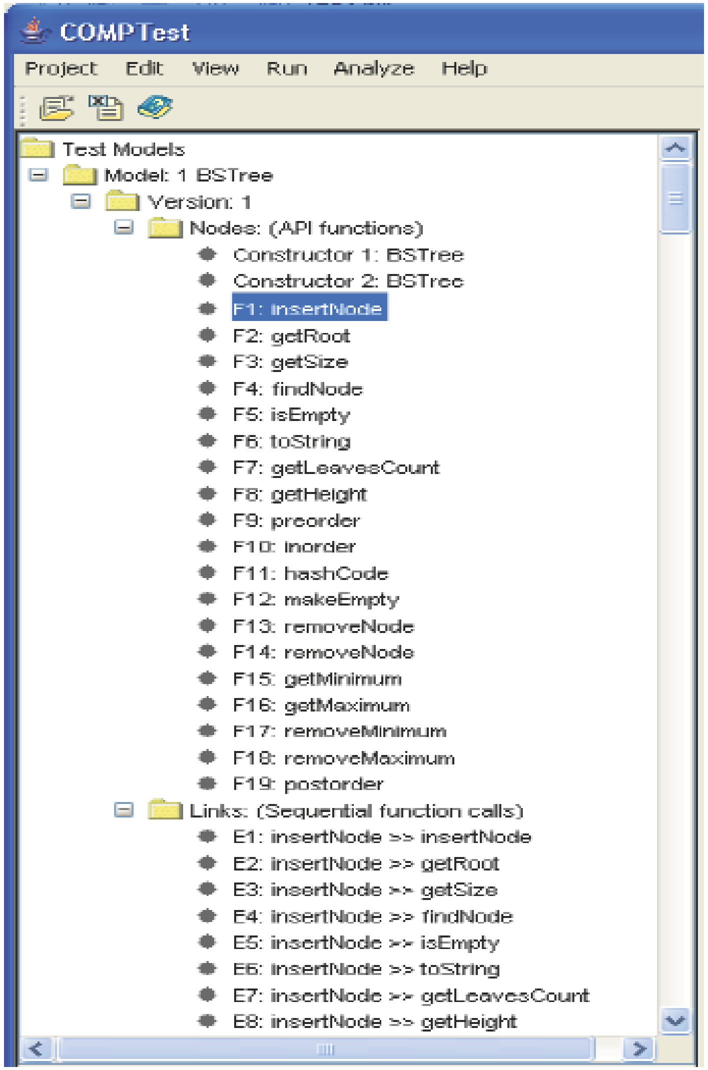
- All test scenarios were analyzed for both node and link coverage

CFAG Representation

- BTree interface has 19 accessible methods
    - $G = (F,E)$
    - $F = F_1 \ldots F_{19}$
    - $E = E_1 \ldots E_{361}$

- A tree structure is used to display the CFAG

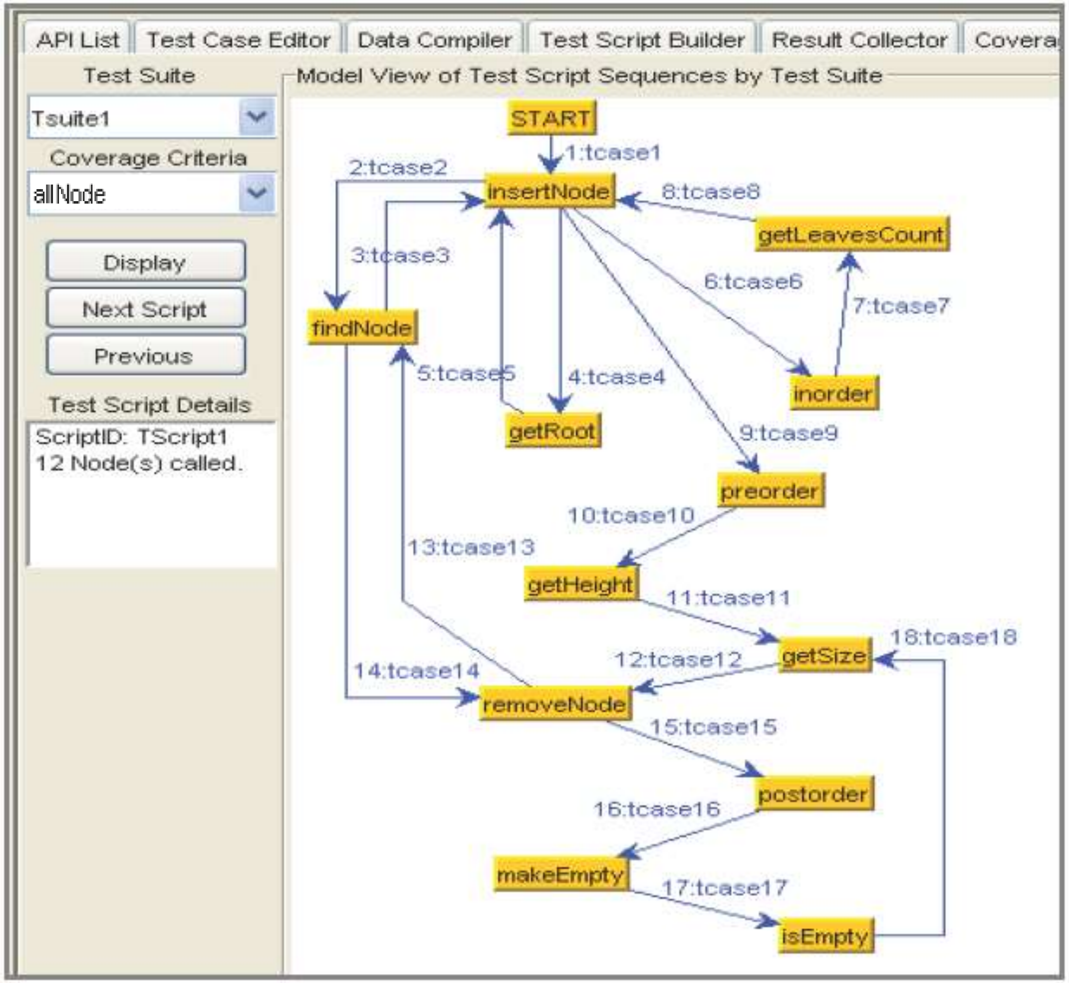
- Selection of a node or link will display more information such as link conditions



Test Models
Model: 1 BSTree
Version: 1
Nodes: (API functions)
Constructor 1: BSTree
Constructor 2: BSTree
F1: insertNode
F2: getRoot
F3: getSize
F4: findNode
F5: isEmpty
F6: toString
F7: getLeavesCount
F8: getHeight
F9: preorder
F10: inorder
F11: hashCode
F12: makeEmpty
F13: removeNode
F14: removeNode
F15: getMinimum
F16: getMaximum
F17: removeMinimum
F18: removeMaximum
F19: postorder
Links: (Sequential function calls)
E1: insertNode >> insertNode
E2: insertNode >> getRoot
E3: insertNode >> getSize
E4: insertNode >> findNode
E5: insertNode >> isEmpty
E6: insertNode >> toString
E7: insertNode >> getLeavesCount
E8: insertNode >> getHeight

D-CFAG Representation

- $G_{T1} = (F[T_1], E[T_1])$ (SHOWN)
  - 18 calls to 12 nodes
  - Node coverage 55%
  - Link coverage 4.5%
- $G_{T2} = (F[T_2], E[T_2])$
  - 13 calls to 8 nodes
  - Node coverage 45%
  - Link coverage 2.9%
- $G_{T3} = (F[T_3], E[T_3])$
  - 13 calls to 11 nodes
  - Node coverage 55%
  - Link coverage 3.2%



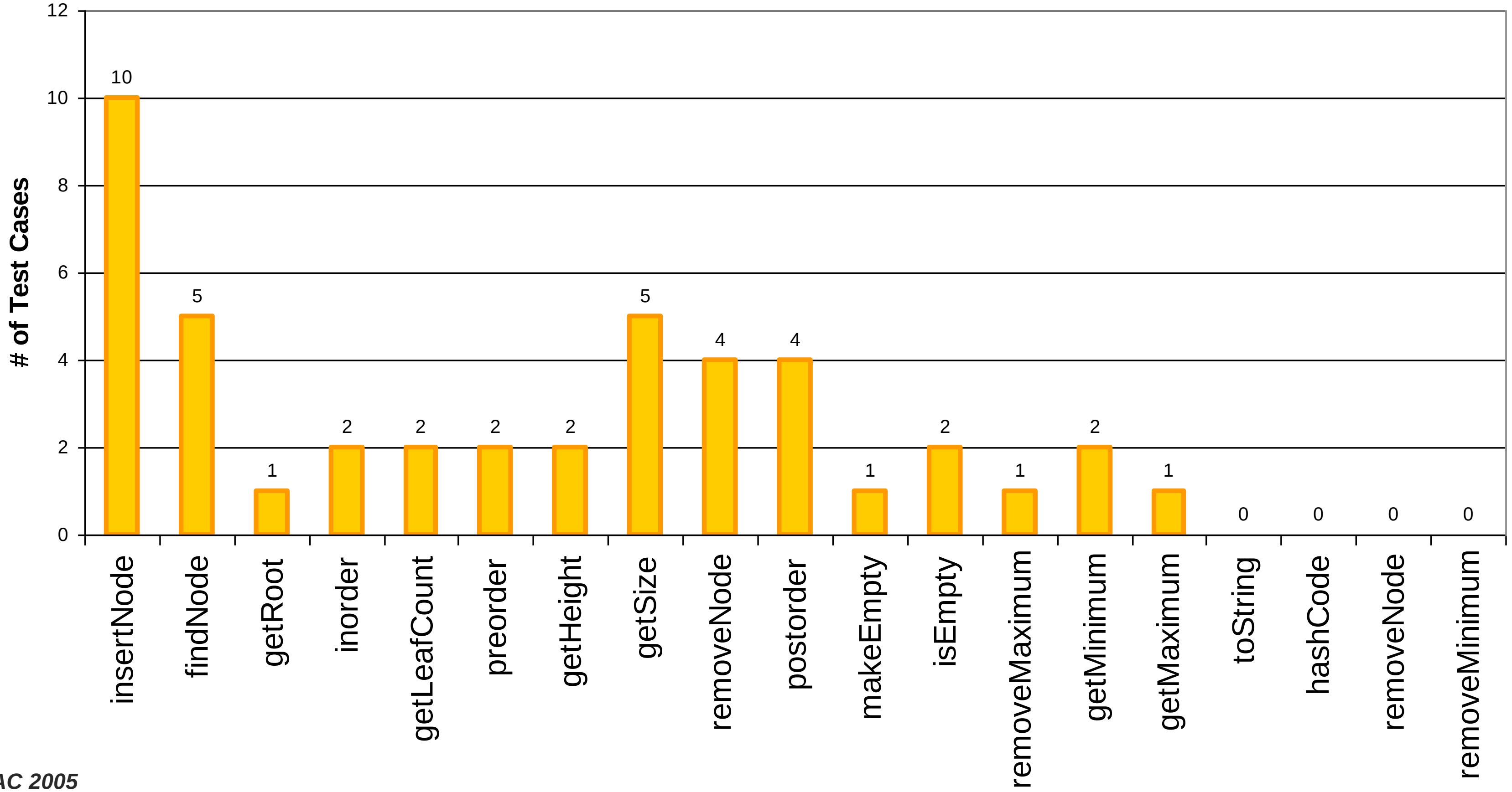
Test Suite: Tsuite1

Coverage Criteria: allNode

Test Script Details:
ScriptID: TScript1
12 Node(s) called.

Coverage Analysis Result:
Node Coverage Criterion

- Covered-Node-Set = F[T_1] U F[T_2] U F[T_3]
- 3 test scenarios provided a total node coverage of **78.9%**



Coverage Analysis Result:
Link Coverage Criterion

- Covered-Link-Set = $E[T_1] \cup E[T_2] \cup E[T_3]$
- 3 test scenarios provided total link coverage of **9.4%**

API Methods	insertNode	findNode	getRoot	inorder	getLeafCount	preorder	getHeight	getSize	removeNode	postorder	makeEmpty	isEmpty	removeMaximum	removeMinimum	getMinimum	getMaximum	toString	hashCode	removeNode
insertNode		1	1		1	1										1			
findNode	3												1						
getRoot	1																		
inorder	2																		
getLeafCount				1			1												
preorder	1					1													
getHeight		1				1													
getSize	1				1		1					2							
removeNode		1		1				1		1									
postorder	1							1	1						1				
makeEmpty										1									
isEmpty									1		1								
emoveMaximum										1									
emoveMinimum																			
getMinimum	1							1											
getMaximum															1				
toString																			
hashCode																			
removeNode																			

Conclusion

- A basic framework is proposed for generating test models (CFAG and D-CFAG) based on a component's API interface and developing a set of coverage criteria for the models
- Challenges for this method:
  - Developing a test model that sufficiently represents an component API
  - Displaying the test model and coverage results in a comprehensive manner
  - Defining adequate coverage criterion for the test model
  - Defining a test set that provides adequate test coverage
- This method provides revealing coverage metrics by facilitating the identification of higher quality test sets that can offer adequate test coverage
- It is a first step towards defining a more effective component-level unit testing method for a component's API

Recommendations and Future Work

- Develop coverage criteria that identify high risk areas that should be tested
- Develop test models and test coverage methods to address adequate testing for component reuse at the component integration and system level
- Future research efforts will be dedicated to:
  - Develop a systematic solution for a minimum path-set
  - Apply current research to components with different types of interfaces and components with multiple APIs
  - Extent the test models to represent component interaction patterns and develop related coverage criteria