Quality Prediction for Component-Based Software Development: Techniques and A Generic Environment

Presented by: Cai Xia
Supervisor: Prof. Michael Lyu
Examiners: Prof. Ada Fu
Prof. K.F. Wong

Dec. 17, 2001
Outline

- Introduction
- Technical Background and Related Work
- A Quality Assurance Model for CBSD
- A Generic Quality Assessment Environment: ComPARE
- Experiment and Discussion
- Conclusion
Introduction

- The most promising solution for large-scale, complex and uneasily controlled modern software system is component-based software development (CBSD) approach.
- The concept, process, life cycle and infrastructure of CBSD are different from traditional systems.
- Quality Assurance (QA) is very important for component-based software systems.
Component-based software development (CBSD) is to build software systems using a combination of components.

CBSD aims to encapsulate function in large components that have loose couplings.

A component is a unit of independent deployment in CBSD.

The overall quality of the final system greatly depends on the quality of the selected components.
What is Component-Based Software Development?

Component repository

Component 1

Component 2

Component n

Software systems

select

Commercial Off-the-shelf (COTS) components

assemble
What is A Component?

- A component is an independent and replaceable part of a system that fulfills a clear function
- A component works in the context of a well-defined architecture
- It communicates with other components by the interfaces
System Architecture

Components Layer
- Basic components
- Common components
- Special business components

Application Layer
- App1
- App2
- App3
Problem Statement

- Due to the special feature of CBSD, conventional SQA techniques and methods are uncertain to apply to CBSD.

- We address the investigation of most efficient and effective approach suitable to CBSD.
Our Contributions

- A QA model for CBSD which covers eight main processes.
- A quality assessment environment (ComPARE) to evaluate the quality of components and software systems in CBSD.
- Experiments on applying and comparing different quality predicted techniques to some CBSD programs.
Outline

- Introduction
- Technical Background and Related Work
- A Quality Assurance Model for CBSD
- A Generic Quality Assessment Environment: ComPARE
- Experiment and Discussion
- Conclusion
A framework can be defined as a set of constraints on components and their interactions, and a set of benefits that derive from those constraints.

Three somehow standardized component frameworks: CORBA, COM/DCOM, JavaBeans/EJB.
<table>
<thead>
<tr>
<th></th>
<th>CORBA</th>
<th>EJB</th>
<th>COM/DCOM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Development</strong></td>
<td>Underdeveloped</td>
<td>Emerging</td>
<td>Supported by a wide range of strong development environments</td>
</tr>
<tr>
<td><strong>environment</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Binary</strong></td>
<td>Not binary standards</td>
<td>Based on COM; Java specific</td>
<td>A binary standard for component interaction is the heart of COM</td>
</tr>
<tr>
<td><strong>interfacing</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>standard</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Compatibility &amp;</strong></td>
<td>Strong in standardizing</td>
<td>Portable by Java language spec; but not very compatible.</td>
<td>Not having any concept of source-level standard of standard language binding.</td>
</tr>
<tr>
<td><strong>portability</strong></td>
<td>language bindings; but not so portable</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Modification &amp;</strong></td>
<td>CORBA IDL for defining component interfaces</td>
<td>Not involving IDL files</td>
<td>Microsoft IDL for defining component interfaces</td>
</tr>
<tr>
<td><strong>maintenance</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Services</strong></td>
<td>A full set of standardized services; lack of implementations</td>
<td>Neither standardized nor implemented</td>
<td>Recently supplemented by a number of key services</td>
</tr>
<tr>
<td><strong>provided</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Platform</strong></td>
<td>Platform independent</td>
<td>Platform independent</td>
<td>Platform dependent</td>
</tr>
<tr>
<td><strong>dependency</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Language</strong></td>
<td>Language independent</td>
<td>Language dependent</td>
<td>Language independent</td>
</tr>
<tr>
<td><strong>dependency</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Implementation</strong></td>
<td>Strongest for traditional enterprise computing</td>
<td>Strongest on general Web clients.</td>
<td>Strongest on the traditional desktop applications</td>
</tr>
</tbody>
</table>
Technical Background and Related Work:

**QA Issues**

- How to certify quality of a component?
  - Size
  - complexity
  - reuse frequency
  - reliability

- How to certify quality of a component-based software system?
Life Cycle of A Component

- **Proposal**
  - reject
  - new

- **Under Construction**
  - affirmed for construction
  - change proposal

- **Ready for Distribution**
  - affirmed for delivery
  - new release of component library

- **To be deleted** (do not use)
  - delete

- **Under use**
  - mark for deletion
Life Cycle of CBSD

- Requirements analysis
- Software architecture selection, creation, analysis and evaluation
- Component evaluation, selection and customization
- Integration
- Component-based system testing
- Software maintenance
Technical Background and Related Work:
Quality Prediction Techniques

- Case-Based Reasoning
- Classification Tree Model
- Bayesian Belief Network
- Discriminant Analysis
- Pattern Recognition
Outline

- Introduction
- Technical Background and Related Work
- A Quality Assurance Model for CBSD
- A Generic Quality Assessment Environment: ComPARE
- Experiment and Discussion
- Conclusion
A QA Model for CBSD

- Component
- System
Main Practices

Component
- Requirement Analysis
- Component Development
- Component Certification
- Component Customization

System
- Architecture Design
- System Integration
- System Testing
- System Maintenance
Process Overview:
Component Requirement Analysis

Initiators (Users, Customers, Manager etc.)

Format & Structure
Current URD
Draft User Requirement Documentation (URD)

Requirement Gathering and Definition

Requirement Analysis
Component Requirement Document (CRD)
Structure for naming & Describing

Component Modeling
Updated CRD with model included

Requirement Validation
User Requirement Changes
Current URD

Component Development

Requirement Document Template
Data Dictionary
System Maintenance
Process Overview: Component Development

- **Component Requirement Document**
  - Requirements
  - Draft Component
    - Existing Fault
    - Well-Functional Component
    - Reliable Component
  - Self-Testing (Function)
  - Self-Testing (Reliability)
  - Development Document
    - For Reference
    - Submit
  - Component Certification
Process Overview: Component Certification

System Requirements

Component Outsourcing

Component Functions

Component Testing

Well-Functional Component

Component Selecting

Component fit for the special requirements

Acceptance

Contract Signoffs, Payments

Component Released

Component Development Document

Specific Component Requirements

Component Certification
Process Overview: Component Customization

System Requirements & Other Component Requirements

Component Customization

Component Document
Reject
Component Changed
Component Document
New Component Document
Component Testing
Component fit for the special requirements
Acceptance
Component Document
System Integration
Assemble
System Maintenance
Process Overview: System Architecture Design

Initiators

System Requirement Gathering

Requests for New Systems

Format & Structure

System Requirement Analysis

Current Document

Draft System Requirements Document

System Requirement Document

System Architecture Design

System Specification

System Architecture

System Specification Document

System Testing

System Requirement

System Maintenance

System Integration
Process Overview: System Integration

- System Requirement
  - Requirements for New Systems
  - Architecture
- System Integration
  - Draft System
  - Current Component
- Self-Testing
  - Fault Component
- Component Changing
  - Component Requirement
  - Selecting New Component
- Final System
  - Final System
  - System Integration Document
- Component Certification
- System Maintenance
Process Overview: System Testing
Process Overview:
System Maintenance

- Users
  - Request and Problem Reports
  - Documents, Strategies
  - Support Strategy
  - User Support Agreement
  - Problem Management
    - Change Requests
  - System Maintenance
    - New Version
  - System Testing

All Previous Phases
The Feature of Our QA Model

Compared with other existing models:

- Simple, easy to apply
- Design for local component vendors (small to medium size)
- Focused on development process, according to the life cycle of CBSD
- Not focused on the measure/predict the quality of components/systems
Outline

- Introduction
- Technical Background and Related Work
- A Quality Assurance Model for CBSD
- A Generic Quality Assessment Environment: ComPARE
- Experiment and Discussion
- Conclusion
ComPARE: A Generic Quality Assessment Environment

- Component-based Program Analysis and Reliability Evaluation
- Automates the collection of metrics, the selection of prediction models, the validation of the established models according to fault data collected in the development process
- Integrates & encapsulate the quality control for different processes defined in our QA model
Objective of ComPARE

- To predict the overall quality by using process metrics, static code metrics as well as dynamic metrics.
- To integrate several quality prediction models into one environment and compare the prediction result of different models.
- To define the quality prediction models interactively.
Objective of ComPARE

- To display quality of components by different categories
- To validate reliability models defined by user against real failure data
- To show the source code with potential problems at line-level granularity
- To adopt commercial tools in accessing software data related to quality attributes
Architecture of ComPARE

- Metrics Computation
- Criteria Selection
- Model Definition
- Model Validation
- Case Base
- System Architecture
- Result Display
- Failure Data
- Candidate Components
Combination of Metrics & Models

Models
- BBN
- CBR
- Tree

Process Metrics
- Time
- LOC

Static Metrics
- CR
- Effort
- CC

Dynamic Metrics
- NOC
- Heap
- Call Graph
- Coverage

Sub-metrics
Quality Control Methods

Existing Software Quality Assurance (SQA) techniques and methods have explored to measure or control the quality of software systems or process.

- Management/process control
- Software testing
- Software metrics
- Quality prediction techniques
Quality Assessment Techniques

- Software metrics:
  - Process metrics
  - Static code metrics
  - Dynamic metrics

- Quality prediction model:
  - Classification tree model
  - Case-based reasoning method
  - Bayesian Belief Network
## Progress and Dynamic Metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>Time spent from the design to the delivery (months)</td>
</tr>
<tr>
<td>Effort</td>
<td>The total human resources used (man*month)</td>
</tr>
<tr>
<td>Change Report</td>
<td>Number of faults found in the development</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Case Coverage</td>
<td>The coverage of the source code when executing the given test cases. It may help to design effective test cases.</td>
</tr>
<tr>
<td>Call Graph metrics</td>
<td>The relationships between the methods, including method time (the amount of time the method spent in execution), method object count (the number of objects created during the method execution) and number of calls (how many times each method is called in you application).</td>
</tr>
<tr>
<td>Heap metrics</td>
<td>Number of live instances of a particular class/package, and the memory used by each live instance.</td>
</tr>
</tbody>
</table>
# Static Code Metrics

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Lines of Code (LOC)</strong></td>
<td>Number of lines in the components including the statements, the blank lines of code, the lines of commentary, and the lines consisting only of syntax such as block delimiters.</td>
</tr>
<tr>
<td><strong>Cyclomatic Complexity (CC)</strong></td>
<td>A measure of the control flow complexity of a method or constructor. It counts the number of branches in the body of the method, defined by the number of WHILE statements, IF statements, FOR statements, and CASE statements.</td>
</tr>
<tr>
<td><strong>Number of Attributes (NA)</strong></td>
<td>Number of fields declared in the class or interface.</td>
</tr>
<tr>
<td><strong>Number Of Classes (NOC)</strong></td>
<td>Number of classes or interfaces that are declared. This is usually 1, but nested class declarations will increase this number.</td>
</tr>
<tr>
<td><strong>Depth of Inheritance Tree (DIT)</strong></td>
<td>Length of inheritance path between the current class and the base class.</td>
</tr>
<tr>
<td><strong>Depth of Interface Extension Tree (DIET)</strong></td>
<td>The path between the current interface and the base interface.</td>
</tr>
<tr>
<td><strong>Data Abstraction Coupling (DAC)</strong></td>
<td>Number of reference types that are used in the field declarations of the class or interface.</td>
</tr>
<tr>
<td><strong>Fan Out (FANOUT)</strong></td>
<td>Number of reference types that are used in field declarations, formal parameters, return types, throws declarations, and local variables.</td>
</tr>
<tr>
<td><strong>Coupling between Objects (CO)</strong></td>
<td>Number of reference types that are used in field declarations, formal parameters, return types, throws declarations, local variables and also types from which field and method selections are made.</td>
</tr>
<tr>
<td><strong>Method Calls Input/Output (MCI/MCO)</strong></td>
<td>Number of calls to/from a method. It helps to analyze the coupling between methods.</td>
</tr>
<tr>
<td><strong>Lack of Cohesion Of Methods (LCOM)</strong></td>
<td>For each pair of methods in the class, the set of fields each of them accesses is determined. If they have disjoint sets of field accesses then increase the count P by one. If they share at least one field access then increase Q by one. After considering each pair of methods, ( LCOM = (P &gt; Q)^2 (P - Q) \cdot 0 )</td>
</tr>
</tbody>
</table>
Quality Prediction Techniques

- Classification Tree Model
  - classify the candidate components into different quality categories by constructing a tree structure

![Classification Tree Model Diagram]
Quality Prediction Techniques

- Case-Based Reasoning
  - A CBR classifier uses previous “similar” cases as the basis for the prediction. case base.
  - The candidate component that has a similar structure to the components in the case base will inherit a similar quality level.
  - Euclidean distance, z-score standardization, no weighting scheme, nearest neighbor.
Quality Prediction Techniques

**Bayesian Network**
- a graphical network that represents probabilistic relationships among variables
- enable reasoning under uncertainty
- The foundation of Bayesian networks is the following theorem known as Bayes’ Theorem:

\[
P(H|E,c) = \frac{P(H|c)P(E|H,c)}{P(E|c)}
\]

where H, E, c are independent events, P is the probability of such event under certain circumstances
Prototype

- GUI of ComPARE for metrics, criteria and tree model
Prototype

- GUI of ComPARE for prediction display, risky source code and result statistics
Outline

- Introduction
- Technical Background and Related Work
- A Quality Assurance Model for CBSD
- A Generic Quality Assessment Environment: ComPARE
- Experiment and Discussion
- Conclusion
Experiment: Objective

- Apply various existing quality prediction models to component-based programs to see if they are applicable
- Evaluate/validate the prediction results to CBSD
- Investigate the relationship between metrics and quality indicator
Experiment: Data Description

- Real life project --- Soccer Club Management System
- A distributed system for multiple clients to access a Soccer Team Management Server for 10 different operations
- CORBA platform
- 18 set of programs by different teams
- 57 test cases are designed: 2 test cases for each operation: one for normal operation and the other for exception handling.
## Experiment: Data Description

<table>
<thead>
<tr>
<th>Team</th>
<th>TLOC</th>
<th>CLOC</th>
<th>SLOC</th>
<th>CClass</th>
<th>CMethod</th>
<th>SClass</th>
<th>SMethod</th>
<th>Fail</th>
<th>Maybe</th>
<th>R</th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2</td>
<td>1129</td>
<td>613</td>
<td>516</td>
<td>3</td>
<td>15</td>
<td>5</td>
<td>26</td>
<td>7</td>
<td>6</td>
<td>0.77</td>
<td>0.88</td>
</tr>
<tr>
<td>P3</td>
<td>1874</td>
<td>1023</td>
<td>851</td>
<td>3</td>
<td>23</td>
<td>5</td>
<td>62</td>
<td>3</td>
<td>6</td>
<td>0.84</td>
<td>0.95</td>
</tr>
<tr>
<td>P4</td>
<td>1309</td>
<td>409</td>
<td>900</td>
<td>3</td>
<td>12</td>
<td>1</td>
<td>23</td>
<td>3</td>
<td>12</td>
<td>0.74</td>
<td>0.95</td>
</tr>
<tr>
<td>P5</td>
<td>2843</td>
<td>1344</td>
<td>1499</td>
<td>4</td>
<td>26</td>
<td>1</td>
<td>25</td>
<td>2</td>
<td>1</td>
<td>0.95</td>
<td>0.96</td>
</tr>
<tr>
<td>P6</td>
<td>1315</td>
<td>420</td>
<td>895</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>39</td>
<td>13</td>
<td>10</td>
<td>0.60</td>
<td>0.77</td>
</tr>
<tr>
<td>P7</td>
<td>2674</td>
<td>1827</td>
<td>847</td>
<td>3</td>
<td>17</td>
<td>5</td>
<td>35</td>
<td>3</td>
<td>14</td>
<td>0.70</td>
<td>0.95</td>
</tr>
<tr>
<td>P8</td>
<td>1520</td>
<td>734</td>
<td>786</td>
<td>3</td>
<td>24</td>
<td>4</td>
<td>30</td>
<td>1</td>
<td>6</td>
<td>0.88</td>
<td>0.98</td>
</tr>
<tr>
<td>P9</td>
<td>2121</td>
<td>1181</td>
<td>940</td>
<td>4</td>
<td>22</td>
<td>3</td>
<td>43</td>
<td>4</td>
<td>2</td>
<td>0.89</td>
<td>0.93</td>
</tr>
<tr>
<td>P10</td>
<td>1352</td>
<td>498</td>
<td>854</td>
<td>3</td>
<td>12</td>
<td>5</td>
<td>41</td>
<td>2</td>
<td>2</td>
<td>0.93</td>
<td>0.96</td>
</tr>
<tr>
<td>P11</td>
<td>563</td>
<td>190</td>
<td>373</td>
<td>3</td>
<td>12</td>
<td>3</td>
<td>20</td>
<td>6</td>
<td>3</td>
<td>0.84</td>
<td>0.89</td>
</tr>
<tr>
<td>P12</td>
<td>5695</td>
<td>4641</td>
<td>1054</td>
<td>14</td>
<td>166</td>
<td>5</td>
<td>32</td>
<td>1</td>
<td>4</td>
<td>0.91</td>
<td>0.98</td>
</tr>
<tr>
<td>P13</td>
<td>2602</td>
<td>1587</td>
<td>1015</td>
<td>3</td>
<td>27</td>
<td>3</td>
<td>32</td>
<td>17</td>
<td>19</td>
<td>0.37</td>
<td>0.70</td>
</tr>
<tr>
<td>P14</td>
<td>1994</td>
<td>873</td>
<td>1121</td>
<td>4</td>
<td>12</td>
<td>5</td>
<td>39</td>
<td>4</td>
<td>6</td>
<td>0.82</td>
<td>0.93</td>
</tr>
<tr>
<td>P15</td>
<td>714</td>
<td>348</td>
<td>366</td>
<td>4</td>
<td>11</td>
<td>4</td>
<td>33</td>
<td>2</td>
<td>5</td>
<td>0.88</td>
<td>0.96</td>
</tr>
<tr>
<td>P16</td>
<td>1676</td>
<td>925</td>
<td>751</td>
<td>3</td>
<td>3</td>
<td>23</td>
<td>44</td>
<td>30</td>
<td>0</td>
<td>0.47</td>
<td>0.47</td>
</tr>
<tr>
<td>P17</td>
<td>1288</td>
<td>933</td>
<td>355</td>
<td>6</td>
<td>25</td>
<td>5</td>
<td>35</td>
<td>3</td>
<td>3</td>
<td>0.89</td>
<td>0.95</td>
</tr>
<tr>
<td>P18</td>
<td>1731</td>
<td>814</td>
<td>917</td>
<td>3</td>
<td>12</td>
<td>3</td>
<td>20</td>
<td>4</td>
<td>9</td>
<td>0.77</td>
<td>0.93</td>
</tr>
<tr>
<td>P19</td>
<td>1900</td>
<td>930</td>
<td>970</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>20</td>
<td>35</td>
<td>1</td>
<td>0.37</td>
<td>0.39</td>
</tr>
</tbody>
</table>
Experiment: Data Description

- TLOC: the total length of whole program;
- CLOC: lines of codes in client program;
- SLOC: lines of codes in server program;
- CClass: number of classes in client program;
- CMethod: number of methods in client program;
- SClass: number of classes in server program;
- SMethod: number of methods in server program;
Experiment: Data Description

- Fail: the number of test cases that the program fails to pass

- Maybe: the number of test cases, which are designed to raise exceptions, can not apply to the program because the client side of the program deliberately forbids it.

- R: pass rate, defined by \[ R_j = \frac{P_j}{C} \]

- R1: pass rate 2, defined by \[ R1_j = \frac{P_j + M_j}{C} \]

C is the total number of test cases applied to the programs (i.e., 57); Pj is the number of “Pass” cases for program j, Pj = C – Fail – Maybe; Mj is the number of “Maybe” cases for program j.
Experiment: Procedures

- Collect metrics of all programs: Metamata & JProbe
- Design test cases, use test results as indicator of quality
- Apply on different models
- Validate the prediction results against test results
Experiment: Modeling Methodology

- Classification Tree Modeling
  - CART: Classification and Regression Trees

- Bayesian Belief Network
  - Hugin system
CART

- Splitting Rules: all possible splits
- Choosing a split: GINI, gwoing, ordered twoing, class probability, least squares, least absolute deviation
- Stopping Rules: too few cases
- Cross Validation: 1/10 for smaller datasets and cases
CART: Tree Structure

```
CMETHOD < 7

TLOC < 1495.5
1

TLOC < 638.5
2

TLOC < 638.5
3

TLOC < 2758.5

CMETHOD < 26

SLOC < 908.5
8

TLOC < 921.5
4

TLOC < 1208.5
5

TLOC < 1208.5
6
```

52
### CART: Node Information

<table>
<thead>
<tr>
<th>Parent Node</th>
<th>Wgt Count</th>
<th>Count</th>
<th>Median</th>
<th>MeanAbsDev</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.00</td>
<td>1</td>
<td>13.000</td>
<td>0.000</td>
<td>17.000</td>
</tr>
<tr>
<td>2</td>
<td>2.00</td>
<td>2</td>
<td>35.000</td>
<td>2.500</td>
<td>17.000</td>
</tr>
<tr>
<td>3</td>
<td>1.00</td>
<td>1</td>
<td>6.000</td>
<td>0.000</td>
<td>6.333</td>
</tr>
<tr>
<td>4</td>
<td>1.00</td>
<td>1</td>
<td>2.000</td>
<td>0.000</td>
<td>2.500</td>
</tr>
<tr>
<td>5</td>
<td>1.00</td>
<td>1</td>
<td>7.000</td>
<td>0.000</td>
<td>4.000</td>
</tr>
<tr>
<td>6</td>
<td>6.00</td>
<td>6</td>
<td>3.000</td>
<td>0.500</td>
<td>4.000</td>
</tr>
<tr>
<td>7</td>
<td>3.00</td>
<td>3</td>
<td>4.000</td>
<td>0.000</td>
<td>3.000</td>
</tr>
<tr>
<td>8</td>
<td>1.00</td>
<td>1</td>
<td>17.000</td>
<td>0.000</td>
<td>14.000</td>
</tr>
<tr>
<td>9</td>
<td>2.00</td>
<td>2</td>
<td>2.000</td>
<td>0.500</td>
<td>8.000</td>
</tr>
</tbody>
</table>
# CART: Variable Importance

<table>
<thead>
<tr>
<th>Metrics</th>
<th>Relative Importance</th>
<th>Number Of Categories</th>
<th>Minimum Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMETHOD</td>
<td>100.000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLOC</td>
<td>45.161</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCLASS</td>
<td>43.548</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLOC</td>
<td>33.871</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SLOC</td>
<td>4.839</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SMETHOD</td>
<td>0.000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCLASS</td>
<td>0.000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

N of the learning sample = 18
### CART: Result Analysis

<table>
<thead>
<tr>
<th>Terminal Node</th>
<th>Mean Faults</th>
<th>CM offense</th>
<th>TLOC</th>
<th>SLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2</td>
<td>7~26</td>
<td>638.5~921.5</td>
<td>&lt;=908.5</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>&gt;7</td>
<td>&lt;=638.5</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>7~26</td>
<td>1208.5~2758.5</td>
<td>&lt;=908.5</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>7~26</td>
<td>638.5~921.5</td>
<td>&gt;908.5</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>&gt;7</td>
<td>&lt;=638.5</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>7~26</td>
<td>638.5~921.5</td>
<td>&lt;=908.5</td>
</tr>
<tr>
<td>1</td>
<td>13</td>
<td>&lt;=7</td>
<td>&lt;=1495.5</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>17</td>
<td>&gt;26</td>
<td>638.5~921.5</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>35</td>
<td>&lt;=7</td>
<td>&gt;1495.5</td>
<td>-</td>
</tr>
</tbody>
</table>
Hugin Explorer System

- Construct model-based decision support systems in domains characterized by inherent uncertainty.
- Support Bayesian belief networks and their extension influence diagrams.
- Define both discrete nodes and continuous nodes.
Hugin: Influence Diagram
Hugin: Probability Description
Hugin: Propagation

- The sum propagation shows the true probability of state of nodes with the total summation 1.

- For the max propagation, if a state of a node belongs to the most probable configuration it is given the value 100, all other states are given the relative value of the probability of the most probable configuration they are found in compared to the most probable configuration.
Hugin: Propagation

- Using max propagation instead of sum propagation, we can find the probability of the most likely combination of states under the assumption that the entered evidence holds. In each node, a state having the value 100.00 belongs to a most likely combination of states.
Hugin: Run Result (sum prop.)

<table>
<thead>
<tr>
<th>ClientClass</th>
<th>88.00 1 - 5</th>
<th>6.00 5 - 10</th>
<th>6.00 10 - 15</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClientMethod</td>
<td>17.00 1 - 10</td>
<td>78.00 10 - 50</td>
<td>50 - 100</td>
</tr>
<tr>
<td>ServerClass</td>
<td>94.00 1 - 5</td>
<td>5 - 10</td>
<td>6.00 10 - 15</td>
</tr>
<tr>
<td>ServerMethod</td>
<td>1 - 10</td>
<td>94.00 10 - 50</td>
<td>50 - 100</td>
</tr>
<tr>
<td>TLOC</td>
<td>0 - 0.5</td>
<td>11.00 0.5 - 1</td>
<td>61.00 1 - 2</td>
</tr>
<tr>
<td>TestResult</td>
<td>35.78 0 - 5</td>
<td>23.11 5 - 10</td>
<td>20.45 10 - 20</td>
</tr>
<tr>
<td>clientLOC</td>
<td>28.00 0 - 0.5</td>
<td>38.00 0.5 - 1</td>
<td>28.00 1 - 2</td>
</tr>
<tr>
<td>serverLOC</td>
<td>17.00 0 - 0.5</td>
<td>61.00 0.5 - 1</td>
<td>22.00 1 - 2</td>
</tr>
<tr>
<td>LOC</td>
<td>91.27 1 - 5</td>
<td>4.93 5 - 10</td>
<td>4.19 10 - 15</td>
</tr>
<tr>
<td>LOC</td>
<td>9.93 1 - 10</td>
<td>96.98 10 - 50</td>
<td>3.49 100 - 200</td>
</tr>
<tr>
<td>LOC</td>
<td>95.89 1 - 5</td>
<td>5.04 50 - 100</td>
<td>1 - 10</td>
</tr>
<tr>
<td>LOC</td>
<td>1 - 10</td>
<td>94.96 10 - 50</td>
<td>5.04 100 - 200</td>
</tr>
<tr>
<td>LOC</td>
<td>0 - 0.5</td>
<td>7.91 0.5 - 1</td>
<td>67.18 1 - 2</td>
</tr>
<tr>
<td>LOC</td>
<td>100.00 0 - 5</td>
<td>5 - 10</td>
<td>10 - 20</td>
</tr>
<tr>
<td>LOC</td>
<td>32.67 0 - 0.5</td>
<td>37.98 0.5 - 1</td>
<td>26.76 1 - 2</td>
</tr>
<tr>
<td>LOC</td>
<td>12.44 0 - 0.5</td>
<td>65.13 0.5 - 1</td>
<td>22.43 1 - 2</td>
</tr>
<tr>
<td>LOC</td>
<td>87.29 1 - 5</td>
<td>6.31 5 - 10</td>
<td>6.49 10 - 15</td>
</tr>
<tr>
<td>LOC</td>
<td>14.75 1 - 10</td>
<td>73.84 10 - 50</td>
<td>5.41 100 - 200</td>
</tr>
<tr>
<td>LOC</td>
<td>93.84 1 - 5</td>
<td>5 - 10</td>
<td>8.36 10 - 15</td>
</tr>
<tr>
<td>LOC</td>
<td>93.39 10 - 50</td>
<td>6.05 50 - 100</td>
<td>100 - 200</td>
</tr>
<tr>
<td>LOC</td>
<td>12.24 0.5 - 1</td>
<td>60.83 1 - 2</td>
<td>20.44 2 - 5</td>
</tr>
<tr>
<td>LOC</td>
<td>100.00 0 - 5</td>
<td>5 - 10</td>
<td>10 - 20</td>
</tr>
<tr>
<td>LOC</td>
<td>22.31 0 - 0.5</td>
<td>44.71 0.5 - 1</td>
<td>26.50 1 - 2</td>
</tr>
<tr>
<td>LOC</td>
<td>18.56 0 - 0.5</td>
<td>61.88 0.5 - 1</td>
<td>19.56 1 - 2</td>
</tr>
</tbody>
</table>
### Hugin: Run Result (max prop.)

- **ClientClass**
  - 100.00 1-5
  - 2.58 5-10
  - 2.31 10-15

- **ClientMethod**
  - 29.55 1-10
  - 100.00 10-50
  - 50-100
  - 2.17 100-200

- **ServerClass**
  - 100.00 1-5
  - 5-10
  - 2.11 10-15

- **ServerMethod**
  - 1-10
  - 100.00 10-50
  - 6.38 50-100
  - 100-200

- **TLOC**
  - 0-0.5
  - 6.12 0.5-1
  - 100.00 1-2
  - 36.07 2-5
  - 3.34 5-10

- **TestResult**
  - 100.00 0-5
  - 67.86 5-10
  - 25.00 10-20
  - 29.56 20-40

- **clientLOC**
  - 100.00 0-0.5
  - 67.86 0.5-1
  - 36.07 1-2
  - 5.36 2-5
  - 5-10

- **serverLOC**
  - 9.46 0-0.5
  - 100.00 0.5-1
  - 48.95 1-2
  - 2-5
  - 5-10

- **ClientClass**
  - 100.00 1-5
  - 2.58 5-10
  - 2.31 10-15

- **ClientMethod**
  - 5.43 1-10
  - 100.00 10-50
  - 50-100
  - 2.17 100-200

- **ServerClass**
  - 100.00 1-5
  - 5-10
  - 2.17 10-15

- **ServerMethod**
  - 1-10
  - 100.00 10-50
  - 6.38 50-100
  - 100-200

- **TLOC**
  - 0-0.5
  - 6.12 0.5-1
  - 100.00 1-2
  - 36.07 2-5
  - 3.34 5-10

- **TestResult**
  - 100.00 0-5
  - 5-10
  - 10-20
  - 20-40

- **clientLOC**
  - 13.29 0-0.5
  - 67.86 0.5-1
  - 36.07 1-2
  - 5.36 2-5
  - 5-10

- **serverLOC**
  - 13.29 0-0.5
  - 100.00 0.5-1
  - 48.95 1-2
  - 2-5
  - 5-10
## Hugin: Result Analysis

<table>
<thead>
<tr>
<th>TestResult</th>
<th>CCLASS</th>
<th>CMethod</th>
<th>SCLASS</th>
<th>SMethod</th>
<th>TLOC</th>
<th>CLOC</th>
<th>SLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-5</td>
<td>1-5</td>
<td>10-50</td>
<td>1-5</td>
<td>10-50</td>
<td>1-2K</td>
<td>0-0.5K</td>
<td>0.5-1K</td>
</tr>
<tr>
<td>5-10</td>
<td>1-5</td>
<td>10-50</td>
<td>1-5</td>
<td>10-50</td>
<td>1-2L</td>
<td>0.5-1K</td>
<td>0.5-1K</td>
</tr>
</tbody>
</table>
## Comparison

<table>
<thead>
<tr>
<th>Modeling</th>
<th>Advantage</th>
<th>Disadvantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classification Tree</td>
<td>Very accurate if the learning sample is large enough</td>
<td>Need large learning data and data description</td>
</tr>
<tr>
<td>Bayesian Belief Network</td>
<td>Can suggest the best combination of metrics for the faults in a specific range</td>
<td>Need expert acknowledge in a specific domain to construct a correct influence diagram</td>
</tr>
</tbody>
</table>
Discussion

- For testing result between 0-5, the range of CMetho, TLOC and SLOC is very close in the two modeling methods.
- For our experiment, the learning data set is limited to 18 teams.
- The prediction results will be more accurate and representative if the learning data set is larger.
Discussion

- If more learning data and more metrics are available, the results will be more complex and hard to analysis.
- This will raise the need for an automatic and thorough modeling and analysis environment to integrate and encapsulate such operations. That’s exactly what ComPARE aims at.
Discussion

- Case-based reasoning is not applied in our experiment because the lack of tool, yet it can be simulated by the results of the classification tree.

- Dynamic metric is not collected because of the complex and confliction of the CORBA platform and existing metric-collected tool.
Conclusion

Problem: conventional SQA techniques are not applicable to CBSD.

We propose a QA model for CBSD which covers eight main processes.

We propose an environment to integrate and investigate most efficient and effective approach suitable to CBSD.
Conclusion

- Experiments of applying and comparing different quality predicted techniques to some CBSD programs have been done.
- Not enough component-based software programs and results collected for our experiment
- Validation/evaluation of different models should be done if learning data set is large enough
Thank you!