The identification of anomalous code measures with conditioned interval metrics

Carlos López, Esperanza Manso, Yania Crespo

GIRO Research Group (Grupo de Investigación en Reutilización y Orientación a Objeto)
Universities of Valladolid and Burgos, Spain

clopezno@ubu.es
{manso,yania}@infor.uva.es
Outline

- Introduction
- Objectives
- Evaluating code tools
- Adaptation of the RefactorIt tool
- Use intervals for code entity
- Conclusions
- Future work
Introduction

Measuring
- Consists of obtaining a numerical value for an attribute of a software product or process
- Part of a process
  - Measurement process defined by Sommerville

Choose measurements to be made
Select components to be assessed
Measure components characteristics
Identify anomalous measurements
Analyze anomalous components
The identification of anomalous code measures with conditioned interval metrics

Introduction

- Identification anomalous measurements
  - Measurement is related with an entity
  - Measure uses a set of software metrics
  - Software metric definitions contain an interpretation of measured value
    - Provides a interval of preferred values
  - How do we identify anomalous measurements of entities?
    - Checking whether a particular measurement is within the range of preferred values
The identification of anomalous code measures with conditioned interval metrics

Introduction

- Our measurement context
  - Software product: Source code
  - Entities: packages, classes, methods
  - Software metrics: Code metrics
    - On packages: Robert Martin, Brito and Abreu.
    - On classes: Chidamber and Kemerer, Lorenz and Kid
    - On methods: McCabe
  - Interpretation of measured value
    - How do we interpret measures values?
    - What intervals of values we have to use?
Introduction

- Problems of interpretation
  - The intervals used to identify anomalous measurements, obtained through empirical experiments, are restricted to the measuring context/software application, thus limiting their use in other contexts/software applications.
  - Even recommended intervals, taken from past measurements in the same context, cannot be used for code entities from different categories (Exception, Test...).
Objectives

- Modify measurement process (continue)
- Add to the knowledge on dependency that the category of the code entity may have in identifying anomalous measurement intervals

Choose measurements to be made

Select components to be assessed

Identify category 1 code entities → Measure category 1 entities

Identify category i code entities → Measure category i entities

Analyze anomalous components

Identify anomalous measurements using specific category thresholds

July 2nd, Málaga, Spain

13th TOOLS Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE 2010)
The identification of anomalous code measures with conditioned interval metrics

Objectives

- Modify measurement process
  - Categories vs. UML stereotypes
    - exception, boundary (system, user and device interface), entity, control, test and utility
    - Define recommended intervals according to the entity category.
  - Adapting our process in a measurement code tool
    - Evaluate code tools
    - Select a tool to adapt
    - Define a prototype
Evaluating code tools

Measurement process needs tools

Characteristics to evaluate tools

- **C1.** Programming language on which the work is done.
- **C2.** Input: binary or source files (binary/source/both).
- **C3.** Number of metrics calculated (C31 Chidamber and Kemerer, C32 Lorenz and Kid, C33 Robert Martin)
- **C4.** Format for exporting results (html/txt/xml/xls).
- **C5.** Graphic indicators or grouping and filtering techniques to analyze results (Yes/No).
- **C6.** Configuration of metrics profiles.
- **C7.** Automatic classification of code entities.
- **C8.** Evaluation of multiple use intervals in the same evaluation.
The identification of anomalous code measures with conditioned interval metrics

Evaluating code tools

<table>
<thead>
<tr>
<th>Tools</th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C31</th>
<th>C32</th>
<th>C33</th>
<th>C4</th>
<th>C5</th>
<th>C6</th>
<th>C7</th>
<th>C8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dependency Finder</td>
<td>java</td>
<td>binary</td>
<td>33</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>html, txt, xml</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>RefactorIt</td>
<td>java</td>
<td>sources</td>
<td>25</td>
<td>5</td>
<td>2</td>
<td>5</td>
<td>html, txt, xml</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>JDepend</td>
<td>java</td>
<td>binary</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>html, txt, xml</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Eclipse Metrics - v1.3.6</td>
<td>java</td>
<td>sources</td>
<td>25</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>xml</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>NDepend</td>
<td>.NET</td>
<td>both</td>
<td>66</td>
<td>6</td>
<td>2</td>
<td>5</td>
<td>html, txt, xml, xls</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SourceMonitor</td>
<td>java, C#, C++, VB</td>
<td>sources</td>
<td>14</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>txt, xml</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
Evaluating tools

- Criteria to select a tool
  - Open source
  - Measure Java source code
  - High number of metrics that implement each set considered

- These criteria leave RefactorIt and Eclipse Metrics as two possible candidates

- The final selection criterion is based on the
  - functionality offered RefactorIt on the metric profile management
  - user interface
  - provide a catalog of refactorings which assist in the maintenance process
The identification of anomalous code measures with conditioned interval metrics

Adaptation of the RefactorIt tool

- Introduction
- Objectives
- Evaluating code tools
- Adaptation of the RefactorIt tool
- Use intervals for code entity
- Conclusions
- Future work

Measurement process on RefactorIt

1. Select component to be assessed
2. Define recommended thresholds
3. Identify anomalous measurement

July 2nd, Málaga, Spain
Adaptation of the RefactorIt tool

- The new requirements to adapt a new process
  - Open classification of code entities
    - Which categories of code entities do we want to consider?
    - Can we define new categories?
  - Classification of entities in the categories considered
    - How do we classify code entities?
    - Can any functionality of the tool assist to inspector?
      - Classification by entity groupings (by inspector)
      - Automatic classification (by algorithms)
  - The adaptation of RefactorIt is named RefactorItUBU
Adaptation of the RefactorIt tool

- RefactorItUBU open classification of code entities
  - Defining a configuration file from which the different categories under consideration are extracted (/refactorit_ubu/estereotipos.csv)
    - By defect standard UML stereotypes
    - Unknown, Exception, Interface, Control, Entity, Test and Utility
  - This classification will be used in two later activities
    - when the use/preferred interval of each metric is defined
    - when the measurement of the component is carried out
The identification of anomalous code measures with conditioned interval metrics

Adaptation of the RefactorIt tool

- RefactorIt tool
- UBU open classification of code entities
- when the use interval of each metric is defined

Introduction
Objectives
Evaluating code tools
Adaptation of the RefactorIt tool
Use intervals for code entity
Conclusions
Future work
The identification of anomalous code measures with conditioned interval metrics

Adaptation of the RefactorIt tool

- RefactorIt tUBU open classification of code entities
  - when the measurement of the component is carried out

<table>
<thead>
<tr>
<th>Target</th>
<th>STR</th>
<th>WMC</th>
<th>RFC</th>
<th>DIT</th>
<th>NOC</th>
<th>LCOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unknown</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lanSimulation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LANSimulation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Network</td>
<td></td>
<td></td>
<td>9</td>
<td>25</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>lanSimulation.internals</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Node</td>
<td></td>
<td></td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Packet</td>
<td></td>
<td></td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>lanSimulation.tests</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LANTests</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LANTests$PreconditionViolation</td>
<td></td>
<td></td>
<td>38</td>
<td>39</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>LANTests$PreconditionViolation</td>
<td></td>
<td></td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>
Adaptation of the RefactorIt tool

- **RefactorItUbu** - Classification by entity groupings by inspector
  - The physical grouping structures are
    - packages, which contain packages and classes
    - classes, which contain methods.
  - The application of a category on a grouping structure is propagated to the rest of the components
  - Example
    - An application with a logical grouping marked by a three-layered architecture could be classified by indicating the category of the three packages that contain the superior levels
The identification of anomalous code measures with conditioned interval metrics

Adaptation of the RefactorIt tool

- RefactorItUbu - Automatic classification by algorithms
  - Based on entities named conventions
    - For instance, the code entities whose name contains the literal strings “interface”, “gui”, “form”, etc. usually belong to the category “graphic interface”
  - The inspector can define a configuration file where categories are associated a strings
    - Configuration file format
      - `<package convention, category package, class convention, category class>`
      - `<“ui”,”Interface”,”listener”,”Control”>`
      - `<“test”,”Test”,”Exception”,”Exception”>`
RefactorIt TUBU needs new data
- Default categories considered: UML standard stereotypes
  - e₁ exception
  - e₂ interface
  - e₃ entity
  - e₄ control
  - e₅ test
  - e₆ utility
  - UnKnown

For each metric in RefactorIt T
- Define category use/preferred intervals
- Example: McCabe complexity metric VG
  - VGe₁ [1,2], VGe₂ [1,2], VGe₃ [1,2], VGe₄ [1,3], VGe₅ [1,1], VGe₆ [1,3]
The identification of anomalous code measures with conditioned interval metrics

Use intervals for code entity

How we get category use/preferred intervals?

Empirical study

Select 10 projects from SourceForge (size=296,752 LOC)
Select and group metrics to evaluate from RefactorIt
  - package, class, method
  - size, documentation, coupling, inheritance, structural complexity, abstraction, cohesion and design principles
Measure the projects selected
Classify entity code in categories (e₁, e₂, e₃...)
Create intervals per category
  - Analyze data/measurements
  - Calculate lower threshold as quartiles Q₁ (percentile 25)
  - Calculate higher threshold as quartiles Q₃ (percentile 75)
The identification of anomalous code measures with conditioned interval metrics

Use intervals for code entity

- **Some examples of intervals**
  - **Weighted Methods per Class (WMC)**
  - Chidamber and Kemerer
  - Class metric
  - Complexity metric
  - RefactorIt preferred/use intervals [1, 50] ***

- **Our empirical intervals (see figure)**

<table>
<thead>
<tr>
<th>Target</th>
<th>STR</th>
<th>WMC</th>
</tr>
</thead>
<tbody>
<tr>
<td>lanSimulation</td>
<td>Unknown</td>
<td>9</td>
</tr>
<tr>
<td>LANSimulation</td>
<td>Unknown</td>
<td>2</td>
</tr>
<tr>
<td>Network</td>
<td>Unknown</td>
<td>2</td>
</tr>
<tr>
<td>lanSimulation.internals</td>
<td>Unknown</td>
<td>2</td>
</tr>
<tr>
<td>Node</td>
<td>Unknown</td>
<td>2</td>
</tr>
<tr>
<td>Packet</td>
<td>Unknown</td>
<td>2</td>
</tr>
<tr>
<td>lanSimulation.tests</td>
<td>Unknown</td>
<td>2</td>
</tr>
<tr>
<td>LANTests</td>
<td>Unknown</td>
<td>2</td>
</tr>
<tr>
<td>LANTests$PreconditionViolationTestCases</td>
<td>Unknown</td>
<td>2</td>
</tr>
</tbody>
</table>

- **e1 exception** [1.75, 4.00]
- **e2 interface** [4.00, 16.00]
- **e3 entity** [5.00, 25.00]
- **e4 control** [3.00, 16.25]
- **e5 test** [2.00, 8.25]
- **e6 utility** [4.00, 22.00]
- **UnKnown** [1, 50] *** special category
Conclusions

- Measurement process is modified
  - Incorporating the inspector’s/evaluator’s knowledge of the code entity classification
  - Exception, Interface, Control, Entity, Test and Utility, Unknown
- A use interval has been proposed for each metric and category of the classification
- The result of adapting the new code entity measurement process to the RefactorIt tool is presented
- Our conclusion is that it is necessary to pursue research into the field opened up by this case study
The identification of anomalous code measures with conditioned interval metrics

Future work

- Replicate the case study with other sets of projects
  - Refine and compare the proposed use intervals
- Validate the proposed measurement process
  - The use intervals of anomalous measurements are more accurate in the new scenario
- Validate, our measurement process proposal, with others sets of software metrics
- The labelling of code entities, as they sometimes do not belong to only one stereotype/category
  - We propose elaborating a new fuzzy classification
The improvements achieved in this work depend on the new task of classification of entities according to the stereotypes/categories code considered. In this sense, it is necessary to carry out experiments that help to validate the consistency of classification by experts.
Thank you for your attention

Carlos López, Esperanza Manso, Yania Crespo

GIRO Research Group (Grupo de Investigación en Reutilización y Orientación a Objeto)
Universities of Valladolid and Burgos, Spain

clopezno@ubu.es
{manso,yania}@infor.uva.es