Specification-based Approach to Select Regression Test Suite to Validate Changed Software

2008 15th Asia-Pacific Software Engineering Conference
SETLabs, Infosys Technologies Ltd.
Bangalore, India.

2012. 10. 12. 박 성 훈
Contents

• Introduction
  – Behavioral slicing
  – Unit of behavior
• Regression test suite selection approach
  – An introduction
• Structuring activity diagrams
  – Brief introduction to activity diagram
  – Introducing unit of behavior
  – The concept of node version number
  – Risk based system testing
• Selection of regression test suite
  – Retrieving information from UCAD
  – Bucketing of different types of nodes
  – Identifying affected paths
  – Finding regression test cases
• Case study
• Conclusion
• Project
Introduction

• Regression test suite selection
  – Utilizes Unified Modeling Language (UML) based Use Case Activity Diagrams (UCAD)

• Use tools
  – Rational Rose
    • Framework and a tool to represent or develop UCAD
  – InFlux™
    • The development of UML analysis models such as use-case activity diagrams named as task-flow diagrams
Introduction

Behavioral slicing

• A given software requirements specifications is first decomposed into a set of use-cases using OOAD best practices

• Each use-case is ‘sliced’ into ‘units of behavior’
Introduction

Unit of behavior

- $\langle \text{UI}, \text{SP}, \text{C}, \text{SO} \rangle$
  - UI : User inputs
  - SP : System process
  - C : A set of conditions on the state of the system
  - SO : System output
Regression test suite selection approach

An introduction

- Based on changes made to software specifications represented as UCADs
Regression test suite selection approach

An introduction

• Structure
  – Behavior slicing using units of behavior
  – Unique node version number
  – Criticality notation

• Identify
  – Affected paths
    • Modification
    • Addition
    • Deletion
    • Shifting

• Select
  – Two type paths
    • Change specifications
    • New specifications
Structuring activity diagrams

Brief introduction to activity diagram

- Essentially six nodes
  - Initial node
  - User action node
  - System processing node
  - System output node
  - Condition node
  - Final node

- Fork and Join
Structuring activity diagrams

Introducing unit of behavior

- Select desired course
- Decompose a complex use case into a set of sub-use cases
- Needs to be validated hence corresponds to a test case or test case step
Structuring activity diagrams

The concept of node version number

- AA : < N.M >
  - AA
  - N
    - The sequence of that node in its type
  - M
    - Version of the node

<table>
<thead>
<tr>
<th>Indicator</th>
<th>Node Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>Initial Node</td>
</tr>
<tr>
<td>UA</td>
<td>User Action Node</td>
</tr>
<tr>
<td>SP</td>
<td>System processing node</td>
</tr>
<tr>
<td>CO</td>
<td>Condition Node</td>
</tr>
<tr>
<td>SO</td>
<td>System output node</td>
</tr>
<tr>
<td>FI</td>
<td>Final Node</td>
</tr>
</tbody>
</table>
Structuring activity diagrams
Risk based system testing

- For risk based system testing
- Description of the nodes
  - [H] : Node with high criticality
  - [M] : Node with medium criticality
  - [L] : Node with low criticality
- The selected test cases are sorted based on their criticality and presented
Selection of regression test suite
Selection of regression test suite

Retrieving information from UCAD

- UCAD developed by InFlux modeler does not come in XML format but in XMI
- Come up with a XML schema for the activity diagram and developed a tool that converts the activity diagram from XMI to XML
- Once a valid input, an activity diagram that adheres to defined XML schema, is passed to the tool, the nodes information is retrieved from the XML file
Selection of regression test suite

Bucketing of different types of nodes

• Affected path Bucketing
  – Added Bucket
  – Deleted Bucket
  – Modified Bucket
  – Unchanged bucket
Selection of regression test suite

Bucketing of different types of nodes

• Comparison algorithm
  – Sorted list of affected path
    – FN\textsubscript{new}
      • First node version number from the new list
    – FN\textsubscript{ori}
      • First node version number from the original list
    – LN\textsubscript{ori}
      • Last node version number from the original list

\begin{verbatim}
let FN\textsubscript{new} = first node version number from the new list
let FN\textsubscript{ori} = first node version number from the original list
let LN\textsubscript{ori} = last node version number from the original list
if the node sequence of FN\textsubscript{new} is equal to the node sequence of LN\textsubscript{ori}
  if the node version of FN\textsubscript{new} is equal to the node version of LN\textsubscript{ori}
    FN\textsubscript{new} is unchanged
    move FN\textsubscript{new} to unchanged bucket
    delete FN\textsubscript{new} from the new list
    delete LN\textsubscript{ori} from the old list
  else
    FN\textsubscript{new} is modified
    move FN\textsubscript{new} to modified bucket
    delete FN\textsubscript{new} from the new list
    delete LN\textsubscript{ori} from the old list
endif
else
  if the node sequence of FN\textsubscript{new} is less than the node sequence of LN\textsubscript{ori}
    let currentPosition = position of FN\textsubscript{ori}
    while currentPosition is not greater than the position of LN\textsubscript{ori} & new list is not empty
      let currentNode = node at currentPosition
      if node sequence of currentNode is equal to the node sequence of FN\textsubscript{ori}
        if the node version of currentNode is equal to the node version of FN\textsubscript{new}
          if FN\textsubscript{new} is unchanged
            currentNode = FN\textsubscript{new}
          endif
          move FN\textsubscript{new} to unchanged bucket
        else
          FN\textsubscript{new} is modified
          move FN\textsubscript{new} to modified bucket
        endif
      endif
      delete currentNode from the original list
      delete currentNode from the original list
    endwhile
  else
    currentNode is deleted
    move currentNode to deleted bucket
    delete currentNode from the original list
  endif
endif
end if
end while loop
move the remaining node in the original list to the deleted bucket
move the remaining node in the new list to added bucket
endif
endif
\end{verbatim}
Selection of regression test suite

Bucketing of different types of nodes

- The node sequence of $\text{FN}_{\text{new}}$ is equal to the node sequence of $\text{LN}_{\text{ori}}$
- The node sequence of $\text{FN}_{\text{new}}$ is less than the node sequence of $\text{LN}_{\text{ori}}$
- The node sequence of $\text{FN}_{\text{new}}$ is greater than the node sequence of $\text{LN}_{\text{ori}}$
Selection of regression test suite
Bucketing of different types of nodes

If
The node version of $\text{FN}_{\text{new}}$ is equal to the node version of $\text{LN}_{\text{ori}}$
Move $\text{FN}_{\text{new}}$ to *unchanged bucket*
Delete $\text{FN}_{\text{new}}$ from the new list
Delete $\text{LN}_{\text{ori}}$ from the old list

Else
Move $\text{FN}_{\text{new}}$ to *modified bucket*
Delete $\text{FN}_{\text{new}}$ from the new list
Delete $\text{LN}_{\text{ori}}$ from the old list

Move the remaining node in the new list to *added bucket*
Move the remaining node in the old list to the *deleted bucket*
Selection of regression test suite

Bucketing of different types of nodes

Let $\text{currentPosition} = \text{position of } F_{\text{ori}}$

While $\text{currentPosition}$ is not greater than the position of $L_{\text{ori}}$ && new list is not empty

Let $\text{currentNode} = \text{node at } \text{currentPosition}$

IF node sequence of $\text{currentNode}$ is equal to the node sequence of $F_{\text{new}}$

IF the node version of $\text{currentNode}$ is equal to the node version of $F_{\text{new}}$

$\text{currentNode} = F_{\text{new}}$

Move $F_{\text{new}}$ to $\text{unchanged bucket}$

ELSE

Move $F_{\text{new}}$ to $\text{modified bucket}$

END IF

Delete $F_{\text{new}}$ from the new list
Delete $\text{currentNode}$ from the original list

ELSE

Move $\text{currentNode}$ to deleted bucket
Delete $\text{currentNode}$ from the original list

END IF

Increment $\text{currentPosition}$

END While Loop

Move the remaining node in the original list to the $\text{deleted bucket}$
Move the remaining node in the new list to the $\text{added bucket}$
Selection of regression test suite

Bucketing of different types of nodes

Move everything in the new list to added bucket
Move everything in the original list to the deleted bucket
Selection of regression test suite

Identifying affected paths

• Modified and Added Node
  – Modified or an added node is visited are the affected paths

• Deleted Node
  – Both the preceding and succeeding nodes visited are the affected paths
Selection of regression test suite

Finding regression test cases

• The affected paths and unchanged paths are identified
  – This behavior has to be re-validated
  – Belong to affected paths are the ones that need to be re-executed
  – All affected paths will appear in the order of criticality
Case study

- Retail system
  - 22 user action nodes
  - 22 system output nodes
  - 12 condition nodes
  - 4 final nodes
  - 342 test cases by original diagram
  - 250 test cases by new diagram
## Case study

**Affected Paths**

<table>
<thead>
<tr>
<th>Scenario /Test cases :</th>
<th>Original ID</th>
<th>Description</th>
<th>New ID</th>
<th>Description</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter the card = Card inserted properly? = Yes &gt; Valid PIN = yes -&gt; Cash withdrawl? = No -&gt; Both PINs Match = no -&gt;</td>
<td>Original ID</td>
<td>Description</td>
<td>New ID</td>
<td>Description</td>
<td>Status</td>
</tr>
</tbody>
</table>

**Criticality:** Low

**Preconditions:**

**Description** | **Test Steps** | **Input** | **Conditions** | **Expected Output**
--- | --- | --- | --- | ---
1. Enter the card | Card inserted properly? = Yes | 1. Display a message to enter PIN
2. Enter the PIN | Valid PIN = yes | 1. Display a message to confirm PIN
3. Enters the operation to be performed | | 1. Display a message to confirm transaction
4. Presses to Change the PIN | Cash withdrawl? = No | 1. Display a message to confirm cash withdrawal
5. Enter the old PIN | | 1. Display a message to confirm old PIN
6. Enter the new PIN | | 1. Display a message to confirm new PIN

**Postconditions:**
Conclusions

• requirements based regression testing approach provides an effective means of change detection at user requirements level and selection of regression test suit based on changes made to an activity diagram
• Unit of behavior, version number
• Algorithm to identify easily the changes made to the activity diagram
• Selection of regression test suite
Conclusions

• Currently the node version number for each node has to be manually typed in along with the description at the time the node is created
Project

• Proposed approach
  – For OOAD
    • Utilizes UML based UCAD
  – Some changes are needed
Project

Structure

Use SRA document
- Use case
- UCAD
- Unit of behavior

Create XML schema and XML document
- Information from UCAD
- Version number
- Criticality
Project
Identify

Development
  Sorting Algorithm
  Comparison Algorithm

Make result document
Project
Select

Refer to Identify result document

Affected paths based test suite

Criticality based test suite classification