A Simple and Effective Measure for Complex Low-Level Dependencies

Dirk Beyer, Ashgan Fararooy

Simon Fraser University
University of Passau
int a, b;

void swap () {
    a += b;
    b = a - b;
    a -= b;
}

int a, b;

void swap () {
    int temp = a;
    a = b;
    b = temp;
}
Comparison

void swap () {
    a += b;
    b = a - b;
    a -= b;
}

void swap () {
    int temp = a;
    a = b;
    b = temp;
}

<table>
<thead>
<tr>
<th>Measure / Function</th>
<th>swap (Left)</th>
<th>swap (Right)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lines of Code</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Cyclomatic Complexity</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DepDegree</td>
<td>6</td>
<td>3</td>
</tr>
</tbody>
</table>
void swap () {
    a += b;
    b = a - b;
    a -= b;
}

void swap () {
    a += b;
    b = a - b;
    a -= b;
}

Dependencies

'a' init def
'b' init def

a += b;
void swap () {
    a += b;
    b = a - b;
    a -= b;
}

\begin{dependency}
\begin{deptext}
\node[init] {a' init def}\node {a += b;};
\node[init] {b' init def}\node {b = a - b;};
\end{deptext}
\end{dependency}
void swap () {
    a += b;
    b = a - b;
    a -= b;
}

dependencies

'a' init def
'b' init def

a += b;
'b' init def

b = a - b;
a += b;
a -= b;
Definition

- Reaching Definitions

\[ rd_G : B \times X \rightarrow 2^B \]
\[ rd_G(b_u, x) = \{ b_d \in B | b_u \text{ uses } x; b_d \text{ defines } x; b_d \rightsquigarrow b_u \} \]

\( G = (B, F) \) : Control-flow graph

- \( B \) : Set of all program operations
- \( X \) : Set of all program variables
Definition

Use-Def Graph

\[ S_G = (B, E) \]
\[ (b_u, b_d) \in E \iff \exists x \in X: b_d \in rd_G(b_u, x) \]

\[ G = (B, F) : \text{Control Flow Graph} \]

\( B \) : Set of All Program Operations

\( X \) : Set of All Program Variables
Use-Def Graphs for Swap

\[
\begin{align*}
'a' \text{ init def} & \rightarrow a + = b; \\
a + = b; & \rightarrow b = a - b; \\
b = a - b; & \rightarrow a - = b; \\
'a' \text{ init def} & \rightarrow \text{int temp} = a; \\
\text{int temp} = a; & \rightarrow b = \text{temp}; \\
b = \text{temp}; & \rightarrow a = b;
\end{align*}
\]
Definition

• **DepDegree**

  • For a single operation of the program

\[
\text{dd}_G : B \to \mathbb{N} \\
\text{dd}_G(b) = |\{b' \in B | (b, b') \in E\}| \\
S_G = (B, E) : \text{Use-def graph} \\
B : \text{Set of all program operations} \\
E : \text{Set of edges in the use-def graph}
\]
Definition

• **DepDegree**

  • For a function of the program

\[
dd_G : G \rightarrow \mathbb{N}
\]

\[
\dd(G) = \sum_{b \in B} \dd_G(b) = |E|
\]

\(G\) : Set of control-flow graphs (functions)

\(S_G = (B, E)\) : Use-def graph

\(G = (B, F)\) : Control-flow graph
Use-Def Graphs for Swap

1. 'a' init def
2. 'b' init def
3. a += b;
4. b = a - b;
5. a -= b;

1. 'a' init def
2. 'b' init def
3. int temp = a;
4. b = temp;
5. a = b;
Properties

Dep-Degree Indicator

- Simple
  - Based on well-known concepts
  - Straight-forward calculation

- Flexible
  - Applicable to all imperative programming languages

- Scalable
  - Applies to well-structured and unstructured code

- Independent
  - Complementing other indicators

- Automatic
  - Based on the facts present in the program code only
Evaluation

● Comparative Assessments
  ● Compared alternative implementations
  ● Compared DepDegree with other widely used indicators
  ● Evaluated refactored code
  ● Compared different revisions of open source software
Evaluation

- Eclipse Plug-in (DepDigger)
  - Automatic calculation of DepDegree
  - Highlighting operations based on DepDegree values

```java
public static int fool(int n, int k)
{
    // Initialize 'arr' as an array
    // of length 'n+1' filled with ones
    int[] arr = arrayOfOnes(n + 1);

    for (int i = 0; i <= n; i++)
    {
        int temp = arr[0];
        for (int j = 1; j < i; j++)
        {
            arr[j] = arr[j] + temp;
            temp = arr[j] - temp;
        }
    }

    return arr[k];
}
```
Thank You

Questions?