Automatic Segmentation of Method Code into Meaningful Blocks to Improve Readability

Xiaoran Wang, Lori Pollock, K. Vijay-Shanker
Computer and Information Sciences
University of Delaware
What affects readability?

```java
public Arguments(NativeCall activation) {
    this.activation = activation;
    Scriptable parent = activation.getParentScope();
    setParentScope(parent);
    setPrototype(ScriptableObject.getObjectPrototype(parent));
    args = activation.originalArgs;
    lengthObj = new Integer(args.length);
    NativeFunction f = activation.function;
    calleeObj = f; // comments if there is any
    int version = f.getLanguageVersion();
    if (version <= Context.VERSION_1_3 && version != Context.VERSION_DEFAULT)
    {
        callerObj = null;
    } else {
        callerObj = NOT_FOUND;
    }
}
```
Wouldn’t It Be Nice?

```
public Arguments(NativeCall activation) {
    this.activation = activation;
    Scriptable parent = activation.getParentScope();
    setParentScope(parent);
    setPrototype(ScriptableObject.getObjectPrototype(parent));
    args = activation.originalArgs;
    lengthObj = new Integer(args.length);
    NativeFunction f = activation.function;
    calleeObj = f;

    int version = f.getLanguageVersion();
    if (version <= Context.VERSION_1_3 && version != Context.VERSION_DEFAULT) {
        callerObj = null;
    } else {
        callerObj = NOT_FOUND;
    }
}
```
Vertical Spacing can Improve Readability

Developers help others by separating blocks with blank lines!
Observations about Vertical Spacing

• Used infrequently: In 16,236 methods with 16-40 lines of code, 35% methods contain only 0-1 blank line.

• Additionally, developers’ usage of vertical spacing is not always consistent.
Our Contributions

• Leverage both program structure and names to identify “meaningful blocks”: consecutive statements that logically implement a high level action

• Present a heuristic solution to insert blank lines automatically and implement a system - SEGMENT

• Evaluation: 88.7% of the time, SEGMENT-generated vertical spacing is as good as or better than original developers’ spacing
Automatic Method Segmentation Process

1. AST Body of Method M
2. Identify Meaningful Blocks
3. Determine Blank Line Insertion
4. Vertically-Segmented Method M

Automatic Blank Line Insertion
Phase I: Identify Meaningful Blocks

• Meaningful block – a sequence of consecutive statements that forms a high level action.
• Is it feasible to identify blocks automatically?
• Approach:
  – Observed a large number of methods
  – Analyzed developers’ usage of vertical spacing
  – Formed generalization
    → nearly every block falls into 3 types
Type 1: Data-Flow Chain Blocks (DFC)

DFC: A sequence of consecutive statements that are related through data flow.

```
Iterator[] classes = new Iterator[4];
classes[0] = sNode.eIterator("class");
classes[1] = sNode.eIterator("subclass");
classes[2] = sNode.eIterator("joined");
```

Four kinds:

- Initialize-Access (IA)

```
transla9onGroup.addChild(new Sphere());
writeNode(transla9onGroup);
```

- Define-Use (DU):

```
frame = movie.appendFrame();
sound = helper.getSoundDefini9on();
int frames = frame.startSound(sound, 30);
```

- Initialize-n-Use-n (InUn):

```
cfw.addLoadThis();
cfw.addALoad(CONTEXT_ARG);
cfw.addALoad(SCOPE_ARG);
```
Type 2: Extended SWIFT Blocks (E-SWIFT)

```
month = month % 12;
if (month < 0)
    month += 12;
```

```
Iterator it = creat_dirs.iterator();
while (it.hasNext()) {
    ....
}
```

**E-SWIFT** block = SWIFT initializations + SWIFT statement (with body)
SynS: A sequence of consecutive statements in which every statement belongs to the same syntactic category
Automatic Method Segmentation Process

1. Vertically-Segmented Method M
2. Automatic Blank Line Insertion
3. Identify Meaningful Blocks
4. Determine Blank Line Insertion
5. AST Body of Method M
Automatic Method Segmentation Process

Automatic Method Segmentation

AST Body of Method M

Identify Meaningful Blocks

- Handle Block Overlap
- Handle Large Block
- Handle Small Block

Vertically-Segmented Method M

Phase II
We believe in general relations between statements in a DFC block are stronger than in SynS.
Handling Different Kinds of Overlap

- In general, use the Order of Block Strength: 
  \[ DFC > E-SWIFT > SynS \]

- How to handle neighboring blocks of same type?

*By definition, (SynS, SynS), (E-SWIFT, E-SWIFT) do not occur.
Let S be the overlap statement between B1 and B2

If S is **init**

   Insert blank line above S

Else

   Form a bigger block // var use in S stronger with B1 than var def with B2

```java
Tree t = new Tree();
t.getNodeTable().addColumn(LABEL_SCHEMA);

Node r = t.addRoot();
r.setString(LABEL, "0,0");
```

```
Tree t = new Tree();
t.getNodeTable().addColumn(LABEL_SCHEMA);

Node r = t.addRoot();
r.setString(LABEL, "0,0");
```
Automatic Method Segment Process

Phase III

1. AST Body of Method M
2. Identify Meaningful Blocks
3. Handle Block Overlap
4. Handle Large Block
5. Handle Small Block
6. Vertically-Segmented Method M

Automatic Blank Line Insertion
Phase III: Handle Large Blocks

Observed some blocks are over 20 lines
Almost all are SynS blocks

1 in.start();
2 out.start();
3 error.start();
4 out.join();
5.error.join();
Partition Large Blocks by Levels of Similarity

Group consecutive statements of SynS block by **refined similarity measure**.

1 in.start();
2 out.start();
3 error.start();
4 out.join();
5.error.join();

1 in.start();
2 out.start();
3 error.start();
4 out.join();
5.error.join();
Automatic Method Segment Process

1. AST Body of Method M
2. Identify Meaningful Blocks
3. Handle Block Overlap
4. Handle Large Block
5. Handle Small Block
6. Vertically-Segmented Method M

Phase IV
Phase IV: Handle Short Blocks

- Because we examine statement pairs individually, sometimes we create single-statement blocks.
- Heuristic: merge single-statement blocks based on similarity with neighboring statements.

* Other merging heuristics details are in the paper.
Automatic Method Segment Process

- AST Body of Method M
- Identify Meaningful Blocks
  - Handle Block Overlap
  - Handle Large Block
  - Handle Small Block
- Vertically-Segmented Method M
Two Evaluation Studies

1. SEGMENT-generated vs. original developers’

2. SEGMENT-generated vs. Gold Standard
   (inserted by people reading the method code)
Study One: Developer-written vs. SEGMENT-inserted

• Two copies of 50 randomly chosen methods
  – SEGMENT-inserted blank lines
  – Original developer-written blank lines

• Manually categorized differences
  Type 1: SEGMENT inserts a blank line; developer does not.
  Type 2: Developer inserts a blank line; SEGMENT does not.
  Type 3: Developer and SEGMENT insert a blank line at different locations.
  Type 4: Developer and SEGMENT insert a blank line at the same location.

• 3 Java programmers judged SEGMENT vs. developer
### Results

<table>
<thead>
<tr>
<th>Type</th>
<th>Count</th>
<th>Agree/Preferred</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Different</td>
<td>1, 2, 3</td>
<td>119</td>
<td>91 (preferred)</td>
</tr>
<tr>
<td>Same</td>
<td>4</td>
<td>128</td>
<td>127 (agreed)</td>
</tr>
</tbody>
</table>

In 25 of the cases where SEGMENT differed from developers, SEGMENT insertions were judged to be better than developers.

Overall, + =219 cases out of 247(i.e. 88.7%) are as good as or better than original developers’!

Humans judged insertion as a whole for entire methods:
- 33 of 50 instances where SEGMENT was preferred
- 10 of 50 instances where the developer’s was preferred
- 4 of 50 equally well

→ (33+4)/50 = 74% as good as or better than developers’!
Study Two:
Gold Standard vs. SEGMENT-inserted

- 50 Java methods without any blank lines
- Human annotators manually inserted blank lines
- 3 separate judgments for each method
- 150 independent method annotations totally
### Results

<table>
<thead>
<tr>
<th></th>
<th>Human Majority</th>
<th>SEGMENT-inserted</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>3/3 Agree</td>
<td>93</td>
<td>92</td>
<td>98.2%</td>
</tr>
<tr>
<td>2/3 Agree</td>
<td>86</td>
<td>69</td>
<td>80.2%</td>
</tr>
<tr>
<td>Total</td>
<td>179</td>
<td>161</td>
<td>89.9%</td>
</tr>
</tbody>
</table>

Among majority agreed places, SEGMENT inserted in **89.9%** of those locations.

Note: 29 places where SEGMENT inserted a blank line but none of the human judges did
Related Work

• Blank lines are important
  – Readability metrics [Buse et.al. 08]
  – Coding standards [SUN 99][Humphrey 96]

• Identify groups of statements that implement high level actions [Sridhara et.al. 11]

• “Beacons” [Brooks et.al. 83][Crosby et.al. 02]

• Other automated tools for comprehension: [Relf 05]
  [Sridhara et.al. 10][Higo et.al. 08]
Summary

• Contributions:
  – First automatic system to insert blank lines into source code to improve code readability
  – Evaluation shows automatically generated blank lines can be as good as or better than developers and that people reading code expect

• Future:
  – Augment system by examining potential code patterns to identify further blocks
  – Suggest points for internal documentation