Assignment 2 - Internal Softare Quality

Heiko Joshua Jungen
Software Engineering
Chalmers University of Technology
Sweden, Gothenburg
Email: jungen@student.chalmers.se

David Fogelberg
Software Engineering
Chalmers University of Technology
Sweden, Gothenburg
Email: fodavid@student.chalmers.se

Contents

I Introduction 1

II Perceived and measured complexity 1

II-A Ranking of perceived complexity 1

II-B Comparison to measured complexity . . 1

III Acknowledgement 1

IV Conclusion 1

Appendix 2

***Abstract*—Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.**

I. Introduction

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

II. Perceived and measured complexity

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

A. Ranking of perceived complexity

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

B. Comparison to measured complexity

III. Acknowledgement

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

IV. Conclusion

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Appendix

Listing 1: The listing shows the sourcecode of the draw-method. Perceived complexity is high, because the method relies on several other methods and global variables. Further, low cohesion and lack of comments additionaly increase the complexity level. The measured complexity for this method is 2.

```
public void draw(DrawHandler drawHandler, DrawingInfo drawingInfo) {
  double width = drawingInfo.getSymmetricWidth(getFirstLifeline(), getLastLifeline(),
       tick);
  double height = TextSplitter.getSplitStringHeight(textLines, width -
      ROUND_PART_WIDTH * 2, drawHandler) + VERTICAL_BORDER_PADDING * 2;
  double topY = drawingInfo.getVerticalStart(tick);
  topY += (drawingInfo.getTickHeight(tick) - height) / 2;
  double leftX = drawingInfo.getHDrawingInfo(getFirstLifeline()).
      getSymmetricHorizontalStart(tick);

  drawHandler.drawArc(leftX, topY, ROUND_PART_WIDTH * 2, height, 90, 180, true);
  width = width - ROUND_PART_WIDTH * 2;
  drawHandler.drawArc(leftX + width, topY, ROUND_PART_WIDTH * 2, height, 270, 180,
      true);
  drawHandler.drawLine(leftX + ROUND_PART_WIDTH, topY, leftX + width +
      ROUND_PART_WIDTH, topY);
  drawHandler.drawLine(leftX + ROUND_PART_WIDTH, topY + height, leftX + width +
      ROUND_PART_WIDTH, topY + height);
  TextSplitter.drawText(drawHandler, textLines, leftX + ROUND_PART_WIDTH, topY, width
      , height,
       AlignHorizontal.CENTER, AlignVertical.CENTER);

  for (Lifeline ll : coveredLifelines) {
    drawingInfo.getDrawingInfo(ll).addInterruptedArea(new Line1D(topY, topY + height)
        );
  }
}
```

Listing 2: checkKeyword

```
private boolean checkKeyword(String keyword) {
  String libName = null;
  if (keyword.contains(".")) {
    String[] split = keyword.split("\\.");
    if (split.length != 2) {
      return false;
    }
    libName = split[0];
    keyword = split[1];
  }

  if (libName != null && !checkLibraryName(libName)) {
    return false;
  }

  String REGEX_KEYWORD = "[\\w␣\\d␣\\s␣\\.␣ß␣äöü␣ÄÖÜ]{1,}";
  if (!Pattern.matches(REGEX_KEYWORD, keyword)) {
    return false;
  }
  return true;
}
```

Listing 3: This code listing contains the intersect-method. It is loosely coupled and the variable names are easy to understand. However, the perceived complexity is increased because cohesion is low, because the method performs two actions at the same time (calculating a minimum and maximum value). The measured complexity for this method is 11.

```java
/**
 * returns the intersection of both points [eg: (2,5) intersect (1,4) = (2,4)]
 * @param nanPriority if true then NaN has priority over other values, otherwise
    other values have priority
 */
public XValues intersect(XValues other, boolean nanPriority) {
  Double maxLeft = left;
  Double minRight = right;
  if (nanPriority) {
    if (other.left.equals(Double.NaN) || other.left > left) {
      maxLeft = other.left;
    }
    if (other.right.equals(Double.NaN) || other.right < right) {
      minRight = other.right;
    }
  }
  else {
    if (left.equals(Double.NaN) || other.left > left) {
      maxLeft = other.left;
    }
    if (right.equals(Double.NaN) || other.right < right) {
      minRight = other.right;
    }
  }
  return new XValues(maxLeft, minRight);
}
```

Listing 4: getParameters

```java
/**
 * Splits up comma-seperated parameters into single parameters.
 *
 * @param parameterLine
 * @return
 * @throws TestfileException
 */
private Object[] getParameters(String parameterLine) throws TestfileException {
  if (parameterLine.length() == 0) {
    return new Object[0];
  }
  String[] parStrArray = parameterLine.split(",");
  Object[] res = new Object[parStrArray.length];
  for (int i = 0; i < parStrArray.length; i++) {
    String strPar = parStrArray[i].toString().trim();

    Object value = null;
    try {
      value = interpretValue(strPar);
    } catch (KeywordException | AssertionError e) {
      throw TestfileExceptionHandler.InvalidParameter(strPar);
    }
    res[i] = value;
  }
  return res;
}
```

Listing 5: getAngle

```java
/**
 * Calculates and returns the angle of the line defined by the coordinates
 */
public static double getAngle(double x1, double y1, double x2, double y2) {
  double res;
  double x = x2 - x1;
  double y = y2 - y1;
  res = Math.atan(y / x);
  if (x >= 0.0 && y >= 0.0) {
    res += 0.0;
  }
  else if (x < 0.0 && y >= 0.0) {
    res += Math.PI;
  }
  else if (x < 0.0 && y < 0.0) {
    res += Math.PI;
  }
  else if (x >= 0.0 && y < 0.0) {
    res += 2.0 * Math.PI;
  }
  return res;
}
```