EECS 4313 Assignment 3
Data Flow Testing, Slice-Based Testing and Mutation Testing

Student Name — Student Number — EECS Account
Edward Vaisman — 212849857 — eddyv
Robin Bandzar — 212200531 — cse23028
Kirusanth Thiruchelvam — 212918298 — kirusant
Sadman Sakib Hasan — 212497509 — cse23152

April 5, 2018

Contents

1 BORG Calendar

1.1 Slice Testing

1.1.1 Chosen Method for Testing

- **Class**: *net.sf.borg.common.DateUtil.java*
- **Method**: *minuteString(int mins)*
- **Method Description**: This method generate a human reable string for a particular number of minutes. It returns the string in terms of hours or minutes or both hours and mintues.
  - **mins** - The first argument is of type integer.

Following is the code of the *minuteString* method:

```
100  public static String minuteString(int mins) {
101
102    int hours = mins / 60;
103    int minsPast = mins % 60;
104
105    String minutesString;
106    String hoursString;
107
108    if (hours > 1) {
109      hoursString = hours + " " +
           Resource.getResourceString("Hours");
110    } else if (hours > 0) {
111      hoursString = hours + " " + Resource.getResourceString("Hour");
112    } else {
113      hoursString = "";
114    }
115
116    if (minsPast > 1) {
117      minutesString = minsPast + " " +
           Resource.getResourceString("Minutes");
118    } else if (minsPast > 0) {
119      minutesString = minsPast + " " +
           Resource.getResourceString("Minute");
120    } else if (hours >= 1) {
121      minutesString = "";
```

```
122     } else {
123       minutesString = minsPast + " " +
              Resource.getResourceString("Minutes");
124     }
125
126     // space between hours and minutes
127     if (!hoursString.equals("") && !minutesString.equals(""))
128       minutesString = " " + minutesString;
129
130     return hoursString + minutesString;
131   }
```

1.1.2 backward Slicing

The backward slicing is one of the static program slicing. The backward slicing of a program can be defined in the form of S(v,n) which refers to statment fragments that contribute to the value of v at statment n. The method we choose for this slice-based testing is *minuterString* from DataUtil class of the Borg Calendar application. The method has two integer primitive types variable called *hours* and *minsPast*. The slices are created based on the references of the statments to these primitive variables.

S(hours, 109)

```
100   public static String minuteString(int mins) {
101
102     int hours = mins / 60;
103
104
105
106     String hoursString;
107
108     if (hours > 1) {
109       hoursString = hours + " " +
              Resource.getResourceString("Hours");
```

The slice can be tested by the following test case:

```
assertEquals("3 Hours",DateUtil.minuteString(180));
```

S(hours, 113)

```
100  public static String minuteString(int mins) {
101
102    int hours = mins / 60;
103
104
105
106    String hoursString;
107
108    if (hours > 1) {
109      hoursString = hours + " " +
           Resource.getResourceString("Hours");
110    } else if (hours > 0) {
111      hoursString = hours + " " + Resource.getResourceString("Hour");
```

This can be tested by the following test cases:

```
assertEquals("3 Hours",DateUtil.minuteString(180));
assertEquals("1 Hour",DateUtil.minuteString(60));
```

S(hours, 121)

```
100  public static String minuteString(int mins) {
101
102    int hours = mins / 60;
103    int minsPast = mins % 60;
104
105    String minutesString;
106    String hoursString;
107
108    if (hours > 1) {
109      hoursString = hours + " " +
           Resource.getResourceString("Hours");
110    } else if (hours > 0) {
111      hoursString = hours + " " + Resource.getResourceString("Hour");
112    } else {
113      hoursString = "";
114    }
115
116    if (minsPast > 1) {
117      minutesString = minsPast + " " +
```

```
        Resource.getResourceString("Minutes");
118   } else if (minsPast > 0) {
119     minutesString = minsPast + " " +
        Resource.getResourceString("Minute");
120   } else if (hours >= 1) {
121     minutesString = "";
```

This statment does not contain the referece to *hours* but it is affected by the elseif condtion on *hours* >= 1. Therefore, we have choose this statment for the slice. This contain minsPast variable staments since we need these varaible to reach the "elseif" condition on *hours*, so these statments are in the slice due to the nested-if condtion structural constraints. The slice can be tested by following test cases:

```
assertEquals("3 Hours",DateUtil.minuteString(180));
assertEquals("1 Hour",DateUtil.minuteString(60));
```

S(minsPast, 117)

```
100  public static String minuteString(int mins) {
101
102
103    int minsPast = mins % 60;
104
105    String minutesString;
106
107
108
109
110
111
112
113
114
115
116    if (minsPast > 1) {
117      minutesString = minsPast + " " +
           Resource.getResourceString("Minutes");
```

This can be tested by the following test cases:

```
assertEquals("50 Minutes",DateUtil.minuteString(50));
```

S(minsPast, 119)

```
100  public static String minuteString(int mins) {
101
102
103    int minsPast = mins % 60;
104
105    String minutesString;
106
107
108
109
110
111
112
113
114
115
116    if (minsPast > 1) {
117      minutesString = minsPast + " " +
           Resource.getResourceString("Minutes");
118    } else if (minsPast > 0) {
119      minutesString = minsPast + " " +
           Resource.getResourceString("Minute");
```

This can be tested by the following test cases:

```
assertEquals("50 Minutes",DateUtil.minuteString(50));
assertEquals("1 Minute",DateUtil.minuteString(1));
```

S(minsPast, 123)

```
100  public static String minuteString(int mins) {
101
102
103    int minsPast = mins % 60;
104
105    String minutesString;
106
107
108
109
```

```
110
111
112
113
114
115
116     if (minsPast > 1) {
117       minutesString = minsPast + " " +
              Resource.getResourceString("Minutes");
118     } else if (minsPast > 0) {
119       minutesString = minsPast + " " +
              Resource.getResourceString("Minute");
120     } else if (hours >= 1) {
121       minutesString = "";
122     } else {
123       minutesString = minsPast + " " +
              Resource.getResourceString("Minutes");
```

This can be tested by the following test cases:

```
assertEquals("50 Minutes",DateUtil.minuteString(50));
assertEquals("1 Minute",DateUtil.minuteString(1));
assertEquals("0 Minutes",DateUtil.minuteString(0));
```

This concludes all the backward slices related to the all the statments reference primitive types that contribute the statment n in the *minuteString* function.

- The data flow analysis you performed and the calculation of the coverage metrics. You must show which test cases are responsible for which dc-paths.
- A description of the test cases you added to improve coverage. If your coverage was already high, discuss how your testing was able to achieve this.
- The slices that you identified and the percentage of slices that your testing covers. You must show which test cases are responsible for which slices.
- A description of the test cases you added to improve slice coverage. If your coverage was already high, discuss how your testing was able to achieve this.
- Evaluate the effectiveness of your test cases using mutation testing. Discuss and address any issues if you have found in your written report.

- Attaching bug reports if bugs are discovered using your testing methods. You should use the same bug report format as in Assignment 1. Do not file these bug reports to the projects bug report system.
- An appendix with the specification of the methods you are testing

2 JPetStore

- The test scenarios that you have created;
- The request rates and the duration of the load tests;
- The analysis of your load tests and the description of any problems that you have found (if there are any).