

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

March 31, 2018

Contents

1	BORG Calendar	3
1.1	Slice Testing	3
1.1.1	Chosen Method for Testing	3
1.1.2	Forward Slicing	4
1.1.3	Backward Slicing	4
1.1.4	Dynamic Slicing	4
2	JPetStore	5

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 + " " +
110             Resource.getResourceString("Hours");
111     } else if (hours > 0) {
112         hoursString = hours + " " + Resource.getResourceString("Hour");
113     } else {
114         hoursString = "";
115     }
116
117     if (minsPast > 1) {
118         minutesString = minsPast + " " +
119             Resource.getResourceString("Minutes");
120     } else if (minsPast > 0) {
121         minutesString = minsPast + " " +
122             Resource.getResourceString("Minute");
123     } else if (hours >= 1) {
124         minutesString = "";
125     }
```

```

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 Forward Slicing

1.1.3 Backward Slicing

Backward slicing is in the form of $S(v,n)$ where the slices are code fragments that contribute to variable v and statement n .

$S(\text{hours}, 102)$

```

100     public static String minuteString(int mins) {
101
102         int hours = mins / 60;

```

$S(\text{minsPast}, 103)$

```

100     public static String minuteString(int mins) {
101
102
103         int minsPast = mins % 60;

```

The following test case covers the two slices listed above.

<pre>assertEquals("1 Hour", DateUtil.minuteString(60));</pre>

1.1.4 Dynamic Slicing

- 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).