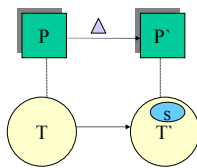


## Modularizing Test Cases: An Application to Regression Testing

Daghstul Seminar - Dec 2003  
Sebastian Elbaum



### Selection Techniques:



Most of the selection techniques are based on the information about the code of the program and the modified version.

Some however are based on the program specifications.

Following are some of the code based techniques, which are used for this study.



## Regression Testing

Regression testing is a necessary and expensive maintenance task performed on modified programs to ensure that the changes have not adversely effected the unchanged code of the program.

One strategy is to rerun the entire test suit on the changed program. This is a heavy resource and time consuming process.

*A solution to this is:*

**Regression test selection techniques:** selects a subset of test cases, thus reducing the time and resources required.



### Selection technique algorithms used for study

**Safe:** selects all the test cases that cover/execute the changed methods at least once.

**Minimization:** selects a minimum set of test cases that execute all the changed methods.

**Random25:** selects randomly 25% of the total test cases.

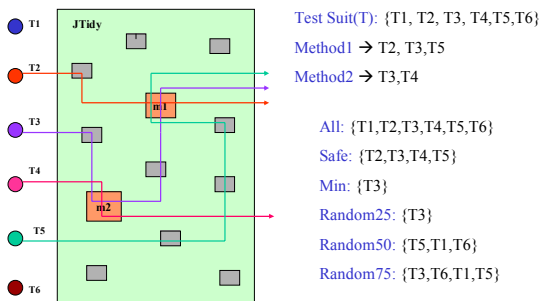
**Random50:** selects randomly 50% of the total test cases.

**Random75:** selects randomly 75% of the total test cases.

**All:** all the test cases.



### Example



## MRT

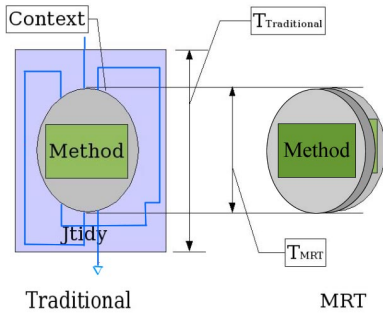
**Modular regression testing:** Instead of running a test case that covers a changed method, re-run only that part of the test case that executes the changed method.

In other words re-run the path of execution of the changed method/module.

➤ This is more likely to gain benefit over execution time.

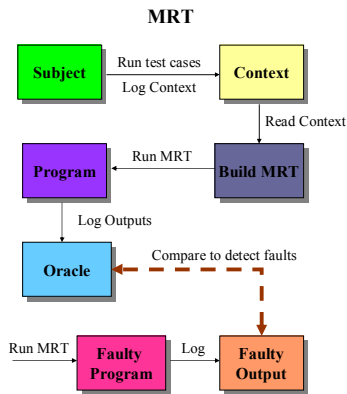
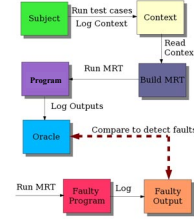


### Time taken to run a test case: comparison



### Steps in MRT implementation

1. Log context
2. Build MRT test case(s)
3. Log oracle
4. Log output on faulty program
5. Compute execution time of MRT test cases and number of faults detected.



### Context Logging

- In java, logging the objects can be done using the ObjectOutputStream class. For this the object's class should implement java.io.Serializable.
- Not all the classes in Jtidy implement Serializable. Instrumented all the classes to implement Serializable.
- Some of the class member objects in Jtidy are system class objects. Java system classes need to implement Serializable, so as to log the context.
- But, not all the system classes can be modified to implement Serializable. Like the java.io.FileInputStream



▪ For each such class in java.io which is used by Jtidy, a new class is created overriding the system class, which can implement Serializable.

**Example:** FileInputStream, this class is replaced with a MRTFileInputStream class which provides the same functionality, but rather than reading from a file, it reads from a buffer.

▪ Logged the context for the method under test, to prepare MRT.



### Example1: Context logging for method

```
public static void trimInitialSpace(Lexer lexer, Node element, Node text)
{
    /* Log context */

    try {
        1
        MRTObjectLogger.writeObject("context", "trimInitialSpace.cxx", lexer);
        MRTObjectLogger.writeObject("context", "trimInitialSpace.cxx", element);
        MRTObjectLogger.writeObject("context", "trimInitialSpace.cxx", text);
    } catch (Exception contextE) {}
    System.out.println("Error while logging context ");
    contextE.printStackTrace();
}

/* End - log context */
}
```



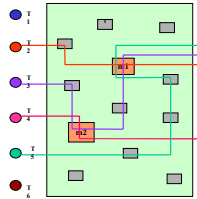
### Example2: Context logging for method

```
public String parseAttribute(MutableBoolean isEmpty, MutableObject asp, MutableObject php)
{
    /* Log context */
    try {
        MRTObject.logger.writeObject("context", "parseAttribute.cxx",this);
        MRTObject.logger.writeObject("context", "parseAttribute.cxx",isEmpty);
        MRTObject.logger.writeObject("context", "parseAttribute.cxx",asp);
        MRTObject.logger.writeObject("context", "parseAttribute.cxx",php);
    } catch (Exception contextE) {
        System.out.println("Error while logging context ");
        contextE.printStackTrace();
    }

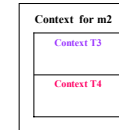
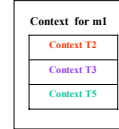
    /* End - log context */
    :
    :
}
```



### Running the test cases on JTidy



### Context logging in MRT by running the test cases

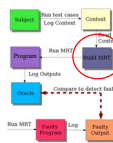


### Building MRT Test case

MRT test case for the method consists the following:

1. Set the context from the log, before the method call.
2. Call the method by passing the logged parameters.

In java, reading the logged objects can be achieved by using the ObjectOutputStream.



### Comparison Metrics

**Time of execution:** The time taken to run the MRT per method under test is added to get the time for the changed program.

**Faults detected:** A fault is detected if the output from seeded program differs from oracle for a test case

Both these values are compared against the values obtained from traditional techniques.



### Experiment

Total number of test cases for JTidy: 98

Version	Technique	No of Test cases
V1	Safe	80
	Min	1
	MRT	861
V2	Safe	48
	Min	2
	MRT	892
V3	Safe	98
	Min	2
	MRT	276
All versions	Random25	24
	Random50	49
	Random75	73
Average no of test cases	Safe	75
	Min	3
	MRT	676



### MRT: size of context

Version	Method	Size of context in KB
V1	trimInitialNamespace	5920.35
V2	fixHTMLNamespace	873.88
	insertedToken	207.11
	parseAttribute	326.72
V3	XMLPreserveWhiteSpace	10.39
	TagTable	369.76
	Main	0
	parseDocType	118.00



## Results

Version	Tot. No of Faults	No of Faults detected by techniques						
		All	Safe	Min	Random			MRT
					25%	50%	75%	
V1	12	8	8	1	3	5	5	8
V2	15	11	11	5	6	9	10	11
V3	9	5	5	3	3	4	5	5



Time including JVM initialization

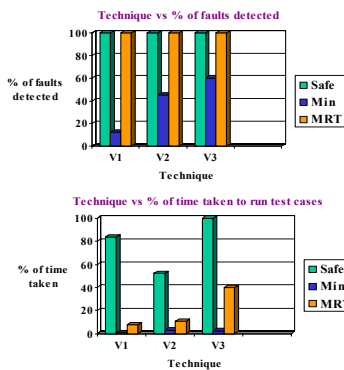
Version	Time taken to run test cases by each technique(in sec)						
	All	Safe	Min	Random			MRT
				25%	50%	75%	
V1	44.06	36.76	0.46	10.91	22.10	32.87	2.09
V2	44.08	22.21	1.10	11.01	22.24	33.02	3.57
V3	44.50	44.53	0.98	10.98	22.30	32.97	17.47

Time excluding JVM initialization

Version	Time taken to run test cases by each technique(in sec)						
	All	Safe	Min	Random			MRT
				25%	50%	75%	
V1	22.90	19.16	0.23	5.20	11.42	16.75	1.81
V2	23.01	11.98	0.75	5.75	11.47	16.90	2.52
V3	22.99	23.00	0.58	5.68	11.58	17.01	9.16



Graphs, taking values for all technique as 100%



## Summary

**Traditional selection techniques:** selects a subset of the test suit.

•**Safe** technique detects maximum faults, but is burdened by the time taken to run the test cases.

•**Minimization** technique selects only a very small set of test cases, but the rate of faults detected by it is low.

•**Random** % techniques vary from moderate to high in both detecting the faults and time taken, as the % of test cases selected increases.

**MRT:** this technique tests a changed module(method). Instead of running an entire set of test cases that cover the changed function, it is enough to run a part of the test case that covers the changed method.



## Conclusion

•Comparing MRT with traditional techniques, one can observe that it detects all the faults that are detected by safe technique and at a very less time.

•MRT is observed to take more time than minimization technique, but there is a tradeoff with the number of faults detected.

•This technique is safer, in that it detects maximum faults at optimal time compared to other techniques.

•To log the context for a method and reuse it, the subject classes and all the system classes need to be serialized.

Thank you !

