Chris Cargile - March 24, 2014 CSCI658 - Testing and Maintenance

HOMEWORK ASSIGNMENT: CHAPTER 16

Question: 16.3:
Motivate the need for competent programmer and the coupling effect hypotheses. Would mutation analysis still make sense if these hypotheses did not hold? Why?

Answer: Competent programmer and coupling effect hypotheses are needed regardless of how syntactically correct programs are in order to produce programs that are valid in terms of identifying the probabilities of potential problem sources, as suspected attributable to certain fault causes.

If history tells us a particular type of Mutation Operator(s) for a given language are likely to repeatedly produce programs with certain fault characteristics, then based on the assumption the programs are relatively the same but for textual differences, a competent programmer should be able to determine for which tests the originally modified (and faulty) program would have failed. Identification of the relevant test cases could allow propagating[1] relevant cases for testing further variants, of additional-varied complexity.

The coupling hypothesis pairs with the competent programmer hypothesis because although a "competent" programmer should be able to recognize idiosyncrasies in a piece of code which could cause bugs, minor syntactical differences can lead to vast logical differences. The coupling hypothesis may substantiate the competent programmer's initial assessment, where even if the programmer misunderstood the cause/effect of an idiosyncrasy, the relevant test cases identified may still catch a fault cause, later.

Mutation analysis would make sense if these hypothesis did not hold, together. It would seem that whether the fault was attributable to a natural cause (in the original program) or that a fault was introduced by a small variation of an original program, introduced accidentally, mutation analysis might still identify for which test cases the program would fail still (coupling effect).

Identifying from where bugs may propagate in further testing is a goal of mutation testing. I later discovered this from an ncsu.edu paper.

Question 16.4:
Generate some invalid, valid-but-nt-userful, useful, equivalent, and non-equivalent mutants for Fig16.1

Mutant-type	Operator	Line	Original/ Mutant	Description
Invalid:	uoi	29	Buf[pos++]=inChar; Buf[pos+++]=inChar;	The mutant is not valid b/c the syntax error would be caught by the c compiler
Valid-but-not-useful:	Uoi	29	Buf[pos++]=inChar; Buf[+pos++]=inChar;	A '+' by itself changes nothing. A '-' would
Useful:				

Equiv:	abs	29	Buf[pos++]=inChar;
			Buf[abs(pos++)]=inChar;
				There's no way to distinguish/kill the mutant

Non-equivalent:	uoi	29	Buf[pos++]=inChar;
			Buf[~pos++]=inChar;
				Taking the 1s-compliment is not equivalent