Third International Conference on Software Testing, Verification, and Validation Workshops

# A Manifesto for Higher Order Mutation Testing†

Mark Harman, Yue Jia and William B. Langdon
King's College London, CREST centre, Strand, London, WC2R 2LS, UK.

***Abstract*—We argue that higher order mutants are potentially better able to simulate real faults and to reveal insights into bugs than the restricted class of first order mutants. The Mutation Testing community has previously shied away from Higher Order Mutation Testing believing it to be too expensive and therefore impractical. However, this paper argues that Search Based Software Engineering can provide a solution to this apparent problem, citing results from recent work on search based optimization techniques for constructing higher order mutants. We also present a research agenda for the development of Higher Order Mutation Testing.**

## I. Introduction

Mutation Testing has proved to be a very effective way to test programs. It is capable of simulating the effect of other white box testing techniques, while providing improved fault detection. This has led to much interest in Mutation Testing. There is evidence to suggest that the approach is increasing in maturity and practical application [30].

Mutants can be classified into two types: First Order Mutants (FOMs) and Higher Order Mutants (HOMs). FOMs are generated by applying mutation operators only once. HOMs are generated by applying mutation operators more than once. Historically, Mutation Testing was always concerned with First Order Mutants (FOM Testing) [15], [22].

The view of Mutation Testing as a process of inserting a *single* fault into a program under test is very well entrenched in the literature. This view also pervades the collective subconsciousness of the research community. It is widely believed that Higher Order Mutants are far too numerous to be practical as a source of simulated faults. Furthermore, many might claim that the coupling effect means that Higher Order Mutants are most likely to be unimportant because they are all coupled to First Order Mutants?

Anyone who has applied traditional First Order Mutation Testing techniques will most likely have experienced two doubts about the validity of the processes being undertaken. First, they will have had the experience of applying mutation operators in the sure knowledge that only a very small proportion of the mutants produced are likely to simulate *real* faults. Secondly, the mutation tester will observe that even the most trivial, small and unimaginative test suite will kill a very large proportion of the First Order Mutants to which it is applied.

These observations do not detract from the results achieved for Mutation Testing. It is true that Mutation Testing can subsume other structural testing techniques and that results have shown that some mutants do, indeed, denote real faults. However, consider the set of all real faults in programs. The First Order Mutants we are so used to generating cannot capture many of these; real faults are simply too complex and subtle to be denoted by a single syntactic change. Furthermore, a lot of time is typically wasted considering mutants that are destined to be killed by the first test case they encounter. This would not be a problem were the whole process to be entirely automated, but it is not and cannot be, because of the problem of equivalent mutants.

We seek to challenge the long–held belief that Higher Order Mutants are 'too numerous and too coupled' to be worthy of consideration. The paper is written as a polemic to challenge the prevailing consensus that the First Order Restriction is acceptable and as a manifesto for Higher Order Mutation. We argue that the space of all mutants (first and higher order) is a search space, in which we should apply search based optimization techniques in order to find mutants that are fit for purpose. We will show that various characterizations of fitness for purpose can be defined, yielding a rich set of possible optimization approaches. We also argue that it is wrong to restrict this search merely to first order mutants, since higher order mutants may be able to capture faults which no first order mutant is capable of capturing.

We use a search based optimization approach to locate very fit mutants within the search space of all possible mutants. The use of Search Based Software Engineering (SBSE) techniques [3], [23], [26], [54] means that we can simultaneously deal with two important problems. By seeking only those mutants which are fit for purpose, we avoid the consideration of trivial, easily killed or unrealistic mutants. At the same time, by selecting only fit mutants we also cope with the large numbers of possible mutants.

Our approach aims to search for a small set of highly fit mutants within an enormous space, rather than to enumerate a complete (and larger) set within a restricted space. It is even possible that a well–guided search may be able to avoid many equivalent mutants, thereby additionally addressing a problem that has hampered mutation testing since its inception.

## II. The Higher Order Approach to Mutation Testing

Higher Order Mutation Testing is a generalization of traditional Mutation Testing in which mutants are permitted to have arbitrary order. More precisely, from a given set of First Order Mutants, $S$, HOM Testing seeks high quality mutants from the set of Higher Order Mutants constructed by inserting one *or more* of the mutants from $S$ into the program under test.

†This paper accompanies the Mutation 2010 keynote given by Mark Harman and based on the work of all three authors on Higher Order Mutation Testing.

978-0-7695-4050-4/10 $26.00 © 2010 Crown Copyright
DOI 10.1109/ICSTW.2010.13

In order to make the use of Higher Order Mutation Testing practicable, we advocate the use of search based optimization techniques. This provides a means of efficiently exploring the space of Higher Order Mutants, guided by a fitness function that seeks to capture mutant 'quality' (or fitness for purpose). The determination of what makes a mutant high quality depends on the application to which HOM testing is put. Naturally, there are a lot of possibilities, of which, only a few have so far been the subject of experimentation. We have considered two possibilities:

1) **single objective**: A HOM is considered to be of high quality if it is harder to kill than the First Order Mutants from which it is constructed [28], [31]. In this approach we explicitly seek subsuming mutants.
2) **multi objective**: Mutant $m_1$ is superior to mutant $m_2$ if $m_1$ is syntactically and semantically smaller than $m_2$ [36]. In this approach we seek lower or equally low order mutants that are harder to kill than those we have in our current set of mutants.

Existing results obtained by the authors from these two approaches are summarised in Sections IV and V.

### A. A Classification of Higher Order Mutants

HOMs can be classified in terms of the way that they are 'coupled' and 'subsuming', as shown in Figure 1. In Figure 1, the region area in the central Venn diagram represents the domain of all HOMs. The sub-diagrams surrounding the central region illustrate each category. For sake of simplicity of exposition these examples illustrate the second order mutant case; one that assumes that there are two FOMs $f_1$ and $f_2$, and $h$ denotes the HOM constructed from the FOMs $f_1$ and $f_2$. The two regions depicted by each sub-diagram represent the test sets containing all the test cases that kill FOMs $f_1$ and $f_2$. The shaded area represents the test set that contains all test cases that kill HOM $h$. The areas of the regions indicate the proportion of the domain of HOMs for each category.

Following the coupling effect hypothesis, if a test set that kills the FOMs also contains cases that kill the HOM, we shall say that the HOM is a 'coupled HOM', otherwise we shall say it is a 'de-coupled HOM'. Therefore, in Figure 1, the sub-diagram is a coupled HOM if it contains an area where the shaded region overlaps with the unshaded regions. For example the sub-diagrams (a), (b) and (f). Since the shaded region from sub-diagrams (c) and (d) do not overlap with the unshaded regions, (c) and (d) are de-coupled HOMs. Sub-diagram (e) is a special case of a de-coupled HOM, because there is no test case that can kill the HOM; there is no overlap, the HOM is an equivalent mutant.

Subsuming HOMs, by definition, are harder to kill than their constituent FOMs. Therefore, in Figure 1, the subsuming HOMs can be represented as those where the shaded area is smaller than the area of the union of the two unshaded regions, such as sub-diagrams (a), (b) and (c). By contrast, (d), (e) and (f) are non-subsuming. Furthermore, the subsuming HOMs can be classified into strongly subsuming HOMs and weakly subsuming HOMs. By definition, if a test case kills a strongly subsuming HOM, it guarantees that its constituent FOMs are killed as well. Therefore, if the shaded region lies only inside the intersection of the two unshaded regions, it is a strongly subsuming HOM, depicted in (a), otherwise, it is a weakly subsuming HOM, depicted in (b) and (c).

According to the combination of subsuming and de-coupled HOM types, the six possibilities we considered are: strongly subsuming and coupled (a), weakly subsuming and coupled (b), weakly subsuming and de-coupled (c), non-subsuming and de-coupled (d), non-subsuming, de-coupled which is equivalent (e), and non-subsuming and coupled (f) which is useless, as shown in Figure 1.

### B. Arguments in Favour of the Higher Order Mutation Testing Approach

This section summarizes the motivation for the introduction of the HOM Testing paradigm. It explains the concerns that have, hitherto, led researchers to avoid HOM Testing, why SBSE means that these concerns are no longer paramount and how HOM Testing may solve the three central problems that have bedeviled Mutation testing since its inception.

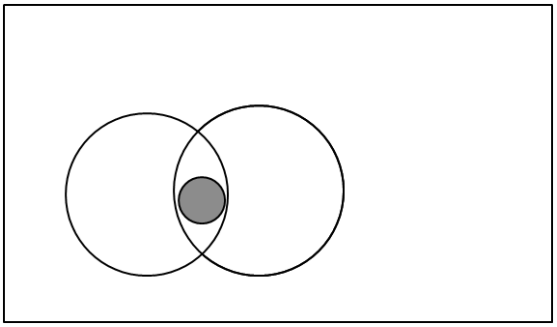
Traditional first order mutation testing has three significant problems:

1) **Cost**: The large number of mutants generated from even small programs leads to a consequently higher cost.
2) **Uncertainty**: The undecidable problem of determining whether a mutant is equivalent to the program from which it is generated leads to uncertainty; is the mutant merely as–yet–unkilled or is it simply unkillable?
3) **Realism**: The problem of whether mutants do or can simulate real faults; if we have killed all the killable mutants, have we also found a large proportion of any real faults present?

A slightly churlish interpretation of these three problems would be that Mutation Testing is expensive, theoretically impossible and has a questionable relationship to reality. It is a testament to the enduring appeal of Mutation Testing that it remains an active area of research and practice despite these significant challenges. Of course, much work on Mutation Testing over the past three decades has sought to address precisely these problems and this has tempered and ameliorated all three problems:

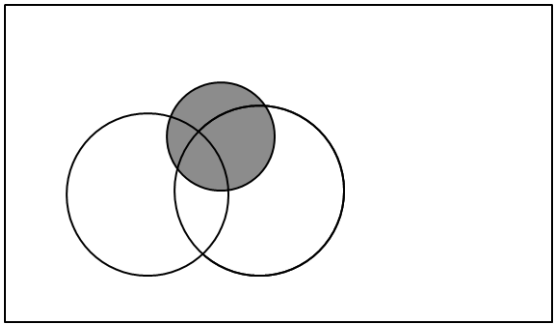
1) **Cost**: Work on Mutant Sampling and Selective Mutation has shown how the number of mutants can be reduced with only a small impact on test effectiveness [1], [9], [61], [39], [14], [32], [55], [38], [50], [32], [62], [48], [40], [7], [41], [42]
2) **Uncertainty**: Work on reducing the impact of equivalent mutants has reduced, though not eradicated, this problem [6], [27], [24], [45], [49], [21].
3) **Realism**: Empirical evidence has been provided that the faults denoted by mutants do, indeed, overlap with a class of real faults [16], [12], [5].

However these three issues remain at the heart of work on mutant testing. The Search Based approach to HOM Testing aims to address all three simultaneously. Before considering how it does this in a little more detail, let us briefly review the reasons why researchers have previously been reluctant



**a. Strongly Subsuming and Coupled**

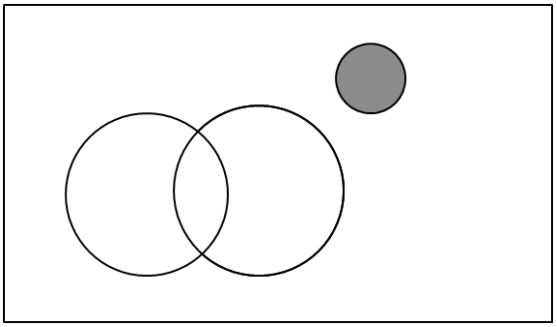
$T_h \subset \bigcap_i T_i \quad and \quad T_h \neq \emptyset$



**b. Weakly Subsuming and Coupled**

$|T_h| < |\bigcup_i T_i| \ , \quad T_h \neq \emptyset$

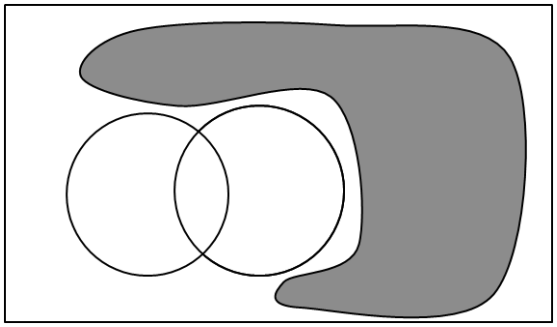
and $\quad T_h \cap \bigcup_i T_i \neq \emptyset$



**c. Weakly Subsuming and De-coupled**

$|T_h| < |\bigcup_i T_i| \ , \quad T_h \neq \emptyset$

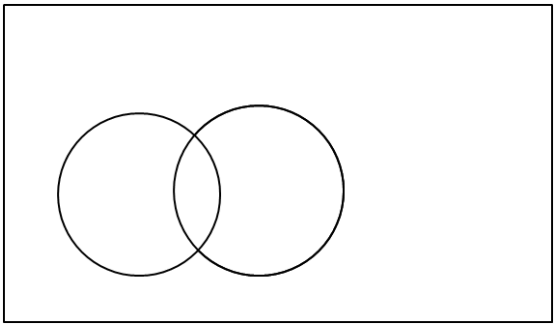
and $\quad T_h \cap \bigcup_i T_i = \emptyset$



**d. Non-Subsuming and De-coupled**

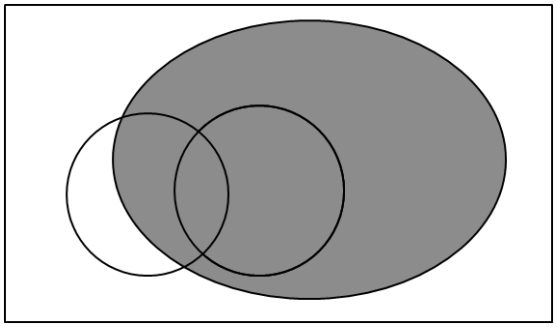
$|T_h| \geq |\bigcup_i T_i| \ , \quad T_h \neq \emptyset$

and $\quad T_h \cap \bigcup_i T_i = \emptyset$



**e. Non-Subsuming and De-coupled (Equivalent)**

$T_h = \emptyset$



**f. Non-Subsuming and Coupled (Useless)**

$|T_h| \geq |\bigcup_i T_i|$

and $\quad T_h \cap \bigcup_i T_i \neq \emptyset$

Fig. 1. HOMs Classification. These pseudo Venn diagrams depict the relationship between some of the interesting classes of HOMs and their first order constituents. In each of the six examples, the rectangle depicts all possible test inputs. The two circles within each rectangle depict the possible regions of test cases that can kill a FOM in the standard way for a Venn Diagram. The shaded region indicates the killing test sets for the HOMs. The size of the shaded region is intended to show the relative size of the HOM kill set compared to the FOM kill sets. Because it attempts to show set size, the diagram is a 'pseudo' Venn Diagram, rather than a true Venn Diagram. For ease of exposition, the diagrams illustrate only the second order case, whereas the definitions cover arbitrary order. HOMs of type (a), (b) and (c) are harder to kill than their constituent FOMs, thereby capturing potentially subtler faults. In particular, type (a) are both subtle and useful; they can replace their constituent FOMs because they are killed by a subset of the intersection of test cases that kill their constituents. For a HOM $h$, constructed from FOMs $f_1, ..., f_n$, the test set $T_h$ contains all the test cases that kill $h$, while the test sets $T_1, ..., T_n$ are the test sets that kill that kill $f_1, ..., f_n$ respectively.

to consider the higher order paradigm. These can be thought of as the two primary reasons for *not* considering the HOM Testing approach:

1) **Exponential Growth**: There are already a large number of first order mutants. This leads to the existing **cost** problem. There are exponentially more Higher Order Mutants, so moving to HOM Testing will surely exacerbate an already difficult problem.
2) **Coupling Effect**: The Coupling Effect Hypothesis [15], [43] suggests that it is unlikely that we shall find Higher Order Mutants that are not coupled to First Order Mutants. Therefore any increase in test effectiveness that accrues from HOM Testing will surely be slight.

These two concerns are understandable in the context of the historical development of Mutation Testing. However, fortunately, as has been demonstrated by our previous work [28], [31], [36], neither of them applies. That is:

1) **Exponential Growth**: using search based techniques allows us to search for a suitable subset of Higher Order Mutants. Rather than this leading to an increase in the number of mutants to be considered, the use of subsuming Higher Order Mutants actually means that we can find *smaller* sets of Higher Order Mutants with identical test effectiveness to larger sets restricted solely to First Order Mutants.
2) **Coupling Effect**: Results on the coupling effect [44] show only that there is a tendency for coupling between First Order and Higher Order Mutants. However, since there are exponentially many Higher Order Mutants, even if only a relatively small proportion are uncoupled this remains a large number of mutants. Furthermore, even coupled Higher Order Mutants can subsume their first order counterparts, thereby reducing test effort without reducing test effectiveness.

The HOM Testing paradigm also addresses, head on, the three challenges of **cost**, **uncertainty** and **realism** that have been present in Mutation Testing from its inception. Specifically:

1) **Cost**: Previous work [28], [31], [36] has shown that it is possible to find HOMs that subsume all of the First Order Mutants from which they are constructed. By focussing on the search for subsuming Higher Order Mutants, we can reduce the overall number of mutants required. For example, a strongly subsuming $5^{th}$ order mutant can replace all five of its first order constituents with no loss of test effectiveness. Therefore, far from increasing cost, HOM Testing is potentially an attractive 'do fewer' approach to reducing mutation testing cost.
2) **Uncertainty**: There is some evidence to suggest that Higher Order mutants may be less likely to be equivalent than First Order Mutants [44]. While this remains a topic for further investigation, it is tempting to speculate that Higher Order Mutants may ultimately provide a way to reduce the equivalent mutant problem. Furthermore, using a co-evolutionary approach, it has been argued [2] that Higher Order Mutants can be generated in a way that almost guarantees no equivalent mutants will be created.
3) **Realism**: There is empirical evidence to suggest that the majority of real faults are complex faults [53], [20]. That is, a number of distinct changes are required to fix them. If fixing a real fault requires several changes, then the injection of this same fault would require several changes to the fault–free version of the program. Therefore, simulating such a fault cannot be achieved by first order mutation. Such a complex fault could only be simulated by higher order mutation.

## III. Myths of Mutation Testing Revisited

The 'First Order Restriction' is deeply ingrained in the mutation testing community and is often expressed implicitly or explicitly in the literature. Jia and Harman [31] presented a list of 'seven Myths of Mutation Testing' in order to challenge the existing consensus. These seven myths are summarised here.

### Real Fault Representation Myth (RFR)

The RFR Myth states that FOMs denote faults that a typical programmer might make. This myth is an incorrect extrapolation from the Competent Programmer Hypothesis. It is true that real programmers are generally competent and that they write nearly correct programs. However, it is widely known that many upstream mistakes and misunderstandings in requirements and design can lead to very big errors [10], [52]. Notwithstanding these, there is empirical evidence that the majority of real faults are not denoted by first order mutants [53], [20]. In order to capture more complex faults some form of HOM Testing is thus unavoidable.

### Unscalability of Mutation Testing Myth (UMT)

The Unscalability of Mutation Testing Myth states that Mutation Testing cannot scale to larger programs because of the large number of mutants created. This paper argues that SBSE [23] can be used to search the limitless spaces of all mutants in order to find those that are fit for purpose. In this way the complexity of Mutation Testing (first order and higher order) can be tamed.

### All Mutants are Equal Myth (AME)

The AME Myth states that all mutants are created equal and effort must be put into trying to kill them all. The AME myth is more implicit in the literature than explicit; authors implicitly treat all mutants as equals though they seldom state this view explicitly. This paper argues that all mutants are definitely not equal and that a search based approach can be defined to provide the necessary discernment between them. Some are trivial to kill and denote only the faults that a competent programmer would most definitely *not* have made. That is, the competent programmer hypothesis is a single implication not a double implication: a competent programmer produces near correct programs, but this does not mean that all near correct programs are equally likely to emanate from a competent programmer.

**Global Mutant Operator Myth (GMO)**
The Global Mutant Operator Myth (GMO) assumes that the best way to create mutants is to define a set of global mutation operators *before* any programs under test have been encountered and then to apply this *same* set of global mutation operators to all programs. By contrast, this paper argues that mutants should be *tailored* to the program under test. This can be achieved using SBSE. The fitness function can be tailored to the program under test in many ways. For instance, the fitness function can use static analysis to guide the search away from likely equivalent mutants, while fault histories can be employed to help it guide the search towards likely realistic faults.

**Competent Programmer Hypothesis Myth (CPH)**
The Competent Programmer Hypothesis states that programmers are generally within a few *keystrokes* of being correct. This formulation of the Competent Programmer Hypothesis is mythical because of from the role played by the crucial word '*keystrokes*'. Faulty programs may be close in their *behaviour* to correct versions (if not their faults would already have been revealed).

However, this does not mean that they are within a few *keystrokes* of correctness. If a seldom used, but nonetheless important requirement is omitted, this may lead to no behavioural differences on typical executions and its omission may only manifest itself on very occasional executions. If a data structure envisaged in the design turns out to be impractical this may only be revealed by relatively infrequent extreme cases. In both situations, the behaviour of the faulty program and the correct version are similar. However, the syntactic changes required to implement the correction are far from a mere few keystrokes.

The well–documented observation that many faults arise from incorrect and misunderstood requirements [10], [52], [57] means that, at best, the CPH Myth can only be maintained if we narrowly constrain correctness to 'correctness with respect to stated requirements'. However, even ignoring requirements, design and other 'upstream' issues, there is empirical evidence [53], [20] sufficiently syntactically proximate to justify the First Order Restriction.

**The Syntactic Semantic Size Myth (SSS)**

The SSS is a direct consequence of the CPH myth. In this paper we take the view that competent programmer produces programs that are within a few *logical* steps of correctness, but that this does not necessarily mean that they are within a few *keystrokes*. There is a big difference between syntactic proximity and semantic proximity. We argue that this can be explored using a multi objective approach [36].

**Coupling Hypothesis Extension Myth (CHE)**

The CHE myth over extrapolates from the coupling effect hypothesis to exclude all Higher Order Mutants from consideration. There are several statements of the coupling effect in the literature. One widely cited statement is due to Offutt [44]:

> "Complex faults are coupled to simple ones in such a way that test data which find all simple ones will detect a high percentage of complex faults."

Observe that, in this formulation, Offutt does *not* claim that test data which finds all simple faults will find *all* complex faults. This paper argues that the coupling effect hypothesis is correct; only a small proportion of higher order mutants are not coupled to their first order constituents. However, there are exponentially many higher order mutants. A small proportion of an exponentially large space can, nevertheless, encompass a large number. Furthermore, even a coupled higher order mutant may be more valuable than its first order constituents when it subsumes its constituents. Therefore, the CHE myth over extends the coupling effect hypothesis to create an artificial restriction to first order mutants that wrongly excludes higher order mutants.

## IV. A Single Objective Approach to Finding Fit HOMs: Reducing Fragility

We summarize the results of our previous work on single objective search based optimization techniques for locating subsuming HOMs [28], [31], [29]. This previous work demonstrated that subsuming (and strongly subsuming) HOMs can be found, efficiently, in real programs (and even in the toy triangle program).

### A. Fitness as Fragility

In our experiments on single objective HOM Testing, fitness was defined to capture a HOM's reduced Fragility, as follows [31]: Let $T$ be a set of test cases, $\{M_1,...,M_n\}$ be a set of mutants, and the $kill(\{M_1, ..., M_n\})$ function returns the set of test cases which kill mutants $M_1, ..., M_n$. We shall define fragility for a set of mutants so that a single definition caters for individual mutants (which may be either first order *or* higher order), but also for sets of individual mutants. That is, the fragility of a mutant shall be defined as follows:

*Definition 1 (fragility):*

$$fragility(\{M_1, ..., M_n\}) = \frac{|\bigcup_{i=1}^{n} kill(M_i)|}{|T|}$$

The value of fragility lies between 0 and 1. When it equals 0 this means that there is no test case that can kill this mutant, which indicates that this mutant is potentially an equivalent mutant. As the value of fragility increases from 0 to 1, the mutant is assessed to be weaker, until the value equals 1, which means that the mutant is so weak that it can be killed by any of the test cases. In the following, we use $M_{1...n}$ to denote a HOM consisting of the FOMs $F_1$ to $F_n$. The fitness function for a HOM is defined as follows.

*Definition 2 (Fitness Function):*

$$fitness(M_{1...n}) = \frac{fragility(\{M_{1...n}\})}{fragility(\{F_1, ..., F_n\})}$$

That is, the fitness of a HOM is defined to be the ratio of the fragility of its HOM to the fragility of the constituent FOMs. When the fitness value reaches 0, the mutant is considered to be a potentially equivalent HOM, and so all such zero-valued HOMs were discarded.

### B. The MILU Tool

In order to conduct experiments with the Single Objective formulation we implemented a tool called MILU [29]. MILU supports both first order and higher order mutation testing for C. It uses the 77 C mutation operators of Agrawal et al. [4] as its set of first order mutants. The tool is publicly available. Further details can be found on the MILU website at

`www.dcs.kcl.ac.uk/pg/jiayue/milu/`

MILU provides a flexible scripting language that supports mutation operator customization. It also includes several engineering features that reduce the cost of mutation testing (both first order and higher order). The tool uses a novel 'test harness' technique to invoke mutants efficiently using shared libraries [29].

The name MILU (麋鹿) was chosen to capture our beliefs about HOM Testing. MiLu is the Chinese name of a special kind of deer that composed of four other animals. It has a horse's head, a deer's antlers, a donkey's body and a cow's hooves. In non-Chinese documents, it is sometimes referred to as Père David's Deer (Elaphurus Davidianus) [60]. MiLus are a kind of $4^{th}$ order mutant. The MiLu is a rare beast, but a valuable one, just like a HOM. Unfortunately the analogy stops there; MiLus are not hard to kill. In fact, the MiLu is an endangered species.

### C. Summary of Results Obtained

In our previous work [31] we report experiments using ten benchmark C programs from the Software-artifact Infrastructure Repository (SIR) [19]. These range from the well-known but trivial `Triangle` program to larger real world examples such as `Gzip` (the open source compression utility) and `Space` (the widely studied European Space Agency program). The sizes of the subject programs range from 50 to 6,000 lines of code. The SIR was used to provide test sets.

Experiments were performed with three algorithms: a Genetic Algorithm, a Hill Climbing Algorithm and a Greedy Algorithm, all of which out–performed a simple random search. The best performing algorithm was found to be the Genetic Algorithm which found more HOMs than any of the others. However, although the Genetic Algorithm found more of the subsuming HOMs, the Hill Climber and Greedy algorithms were also found to have their own peculiar advantages, indicating that all three algorithms may have a role to play in the search for valuable HOMs. The Hill Climber tended to find the fittest HOMs, while the Greedy Algorithm tended to find the highest order HOMs. The fittest HOMs may be interesting because they are the most stubborn. The highest order mutants may find application in attempts to reduce mutation effort because they subsume the largest number of first order mutants.

The fitness function sought only to find subsuming HOMs, it did not explicitly target Strongly Subsuming HOMs. Nonetheless, of all subsuming HOMs found in the experiments, approximately 15% were strongly subsuming. This finding indicates that strongly subsuming HOMs may not be too hard to find and that, therefore, there is reason to hope that HOMs can be used to replace whole sets of constituent FOMs without loss of test effectiveness.

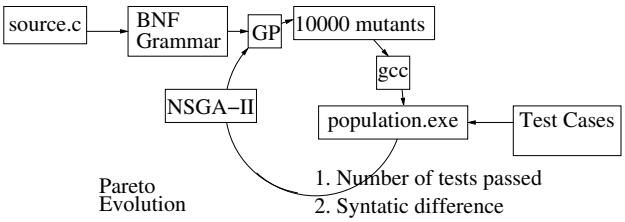



Fig. 2. High order Multi-objective mutation testing. The BNF grammar tells GP [51] where it can insert mutations into the original program `source.c`. Initially GP creates a population of random mutations, which are compiled and run against the test suite providing two objectives to NSGA-II [13] which selects the mutants to retain using a non-dominated Pareto approach [33, Sec. 3.9] and instructs the GP which mutants to recombine or further change.

## V. A Multi Objective Approach to Finding Fit HOMs: Balancing Syntactic and Semantic Differences

Figure 2 shows our recent work [36] where we use search based techniques to look for higher order mutants according to two simultaneous objectives: how easy are they to kill and how different are they from the code under test. Each C program is analysed and a context free grammar that describes it and its mutation sites is automatically generated. This is not the grammar of C but rather a very specific grammar for the current code. However, it describes precisely each first order mutation and how, by combining these, higher order mutants are created. A strongly typed genetic programming (GP) system uses the grammar [37] to generate high order mutants. The grammar is set up to ensure every mutant can be compiled. GP creates a population of mutants. The two dimensional fitness of each is evaluated by running them using the existing test suite. NSGA-II is a complete genetic algorithm [13]. We have stripped out the fitness evaluation and breeding steps and use the remainder to perform Pareto ranking multi-objective selection. That is, NSGA-II does not do fitness calculation or crossover, instead it simply selects members of the current population of mutants according to their multi-objective fitness and then instructs the GP to create the next population of mutants.

Since we are dealing with many mutants at once there are several economies of scale we can take advantage of. Instead of compiling each mutant individually, we can place them all in the same file and compile them all in one go. Also functions which are known not to have been mutated, need not be repeatedly compiled. By linking all the mutants and the non-mutated code into a single image, a whole generation of mutants can be tested in a single process. This allows many one-off costs associated with creating processes, including obtaining resources and opening files, to be shared by the whole population. This can be an appreciable saving. Depending upon the nature of the program and its test suite, it may be more efficient to run all the mutants on a given

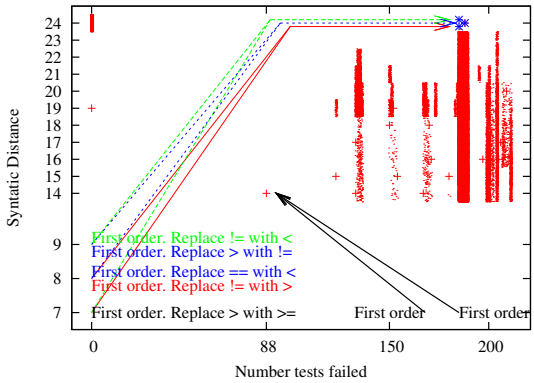



Fig. 3. Distribution of fitness of all $\frac{84\times83}{2}5^2$=87 150 well tested second order gzip comparison mutants. Unit noise used to spread dots. Clearly mutants have a range of behaviours (cf. AME, and SSS). (*) Three non equivalent mutants $2^{nd}$ order mutants each formed from two $1^{st}$ order mutants each of which individually passed all the tests. (Total of 5 first order, lower left.)

test before moving onto the next one. Having all the mutants together makes it easy to invert the normal order in which tests are run. (E.g. [34].) Other efficiencies and practical measures, e.g. ensuring mutants can always be run and always terminate in a reasonable time, and avoiding non-determinism, are listed in Table I.

Figure 3 gives two counter examples to CHE. The coloured arrows and * highlight three second order mutations found in gzip by search which are easy to kill but are each made from two first order mutants which make no difference to the test cases. (i.e. they cannot be killed.) For completeness Figure 3 shows all second order mutants of a specific class (i.e. interchanging comparison operators, cf. GMO, in code which is frequently tested). Of course one would not use them all. Instead we use Figure 3 to illustrate interesting examples. The second counter example to CHE in gzip, is that there is a second order mutant which is significantly harder to kill than either of its two components. (It is highlighted by the pair of solid black arrows in Figure 3.) It is killed by 88 tests, less than half the number that kill either of its component first order mutants.

The vertical bars in Figure 3 are not accidental but indicate structure. Analysis of high order mutants, even very high order, has revealed structure which in turn gave insight into the test suite.

## VI. Future Directions for Higher Order Mutation Testing

We set out some directions for the future development of work on HOM Testing.

### A. Automated Test Data Generation

Most work on Mutation Testing has been concerned with the problem of constructing mutants and controlling the problems of cost and uncertainty that derive from the large number of mutants and the undecidability of program equivalence. There has been some work on techniques for generating test data to kill mutants, thereby using Mutation Testing and a test data generation technique rather than merely as a test data assessment technique. Previous authors have used constraint satisfaction [17], [18], dynamic domain reduction [46], [47] and search based techniques [8], [56].

Voas [59] characterised the problem of killing a mutant in terms of the PIE framework. That is, the change in state caused by the mutant's execution needs to be propagated (the 'P' in PIE) to some point at which it has an observable effect (e.g. through output of an incorrect value). However, in order to have a chance to do this the data state needs to be altered (or infected: the 'I' in PIE) by the execution of the mutant. Of course, none of this can occur unless the mutant is actually executed by a test case (the 'E' in PIE).

In first order mutation testing, the attempt to execute a mutant is essentially a reformulation of the problem of reachability, so the 'E' in PIE can be addressed using techniques for branch coverage. Infection can be assessed at the point of mutation itself (and this would be sufficient for weak mutation). For strong mutation we additionally require propagation to some output. This is essentially a reformulation of path sensitivity, since all paths from the mutation point to all reaching uses in an output can be safely approximated using a static analysis, such as slicing [25], [58].

For higher order mutants, the problem is a little more complex. In order to kill an arbitrary HOM it may be sufficient to execute at least one of its constituent FOMs. However, for strongly subsuming HOMs, it is only possible to kill the HOM if all the first order mutants are visited by the execution of the killing test case. This makes the problem of killing a HOM harder. However, this is only to be expected. The whole idea of HOM Testing is to create mutants that are hard to kill and therefore to simulate faults that may have gone unnoticed up to that point. We are working on techniques to extend and develop search based and constraint solving to create techniques for killing higher order mutants.

### B. Exploration of Types of Higher Order Mutant

As shown in Figure 1 there are many different classes of higher order mutants. Each may be interesting in its own right. Search based techniques can be used to target a particular kind of HOM because the fitness function can be written specifically to reward the search when it locates mutants that more closely resemble the desired characteristics. It will be interesting to explore the spaces of different classes of HOMs using search based optimization to see whether they denote particular types of fault of behaviour. It will also be interesting to sample the space of HOMs to approximate the relative proportion of HOMs in each mutant class. This will allow us to answer research questions such as whether or not a particular class is uniformly distributed across all programs or whether certain types of programming style give rise to certain characteristics in the distribution of classes of HOMs.

There are many more potentially valuable and interesting types of HOM that those depicted in Figure 1. We are currently developing a theory of Higher Order Mutation Testing which will classify and explain each of the possible types of HOM in terms of how it interacts with its constituent FOMS. The theory

TABLE I
PRACTICAL TECHNIQUES TO MAKE MUTATION TESTING SCALE. CF. UMT. FROM [35].

| Problems | Solutions |
|---|---|
| Automatic comparison of the output of SIR programs | Replace `printf` results and error messages by status codes (`int`) |
| Array indexing errors | Automatic array index checking before all array accesses |
| Run out of memory or (corrupt the heap) | Allocate heap memory large enough for all of the test cases |
| Read or write illegal memory | Automatic pointer checking before it is used. |
| Non-terminated loops | Loop counter technique to kill mutants |
| Harmful system calls and IO operations | Record original program's use of system calls and IO by instrumenting the code. Intercept and check system & IO during mutation testing |
| Heavy disk usage | Combine tests into a single file. A potential alternative might be to use RAM disk |
| Non-deterministic mutants | Force the initialisation of all variables |

will allow us to explore the relationship between HOMs, the test cases that kill them and the sets of FOMs from which they are constructed. We believe that this theory will shed more light on the value and applicability of Higher Order Mutation Testing.

### C. Co-evolution of Mutants and Test Cases

Co-evolution is an approach to evolutionary optimization in which two or more candidate populations evolve together, with the fitness of one being determined by the other [11]. In this way the two populations simultaneously evolve. This can be a cooperative process, simulating symbiotic behaviour in natural evolution, or it can be competitive, simulating the familiar predator/prey model of co-evolutionary adaption and advancement.

For Mutation Testing, it has been argued [2] that the predator/prey model of competitive co-evolution can be used to develop sets of hard–to–kill higher order mutants and, simultaneously, sets of very good quality test cases that are adapted to reveal subtle and hard–to–detect faults. In this approach, the two populations are the population of candidate HOMs and the population of candidate test cases. The fitness for the HOMs is measured in terms of their ability to evade the test cases (how many test cases fail to kill them). The fitness of the test cases is measured in terms of their ability to kill the mutants.

We can also give a low fitness to mutants that evade all test cases. These may be equivalent. Of course, they may also merely be stubborn and our test cases are insufficiently good to reveal them. Such stubborn (nearly equivalent) mutants are precisely the kind we seek to find. However, evolution is a mercifully robust process. The genes of such stubborn mutants will be scattered through the population. If mutants which initially appeared to be equivalent are, in fact, merely stubborn, then it is likely that they will be re-discovered at a later stages of the evolution, because they remain distributed through the gene pool. As ever, this means that maintenance of population diversity will be important for this form of co-evolution to succeed.

The argument for mutation testing, developed over the thirty years' of its history can seem circular: the mutants are good if they avoid being killed by the test cases, but how do we know that the test cases are any good; they are good if they kill the mutants. The Co-Evolutionary approach turns this uncomfortable circularity from a problem into an advantage. That is, the apparently circular nature of mutation testing makes it an ideal candidate for a co-evolutionary approach. The aim is to make this a virtuous circle of co-evolutionary improvement.

### D. Incorporation of Fault Models into HOM Testing

For systems developed over a longer period of time, there is often fault data available. For systems developed in a certain domain or by a certain team, there may also be fault information available about the domain or team. In such situations, we are effectively constructing a fault model. Rather than simply constructing all possible faults, we can focus on the faults characterised by the fault model. Furthermore, using HOM Testing, we can seek combinations of these faults that may have gone undetected because they partly mask each other. By definition a subsuming HOM is one in which the first order constituent mutants partly mask one another so that the HOM so-constructed is harder to kill than its constituent FOMs.

The search based approach is very well adapted to the presence of a fault model. It can be used to search for faults that are not only exemplars of the fault model, but also HOMs which denote subtle combinations of known likely faults. The search based approach can also be used to seek out near neighbours of known faults, using the fault model as a guide. In this way the search based approach can relax constraints so that the fault model is not used 'literally'. Rather, it is treated as a guide to the *kind* of faults that may occur.

### E. Cost Benefit Analysis of Selective HOM Testing

A strongly subsuming HOM subsumes the FOMs from which it is constructed and so any test case that kills the HOM is guaranteed to kill all of the FOMs. This means that it would be pointless to retain any of the FOMs in the set of mutants; we need only retain the single HOM. We can reduce the number of mutants by focusing on strongly subsuming higher order mutants. In this way, from a staring set of first order mutants, we can seek a reduced set of mutants that has the same power simply by allowing higher order mutants.

The reduced number of mutants may help to reduce the number of test cases required to kill the mutants. This suggests that there may be a rich space of cost–benefit trade offs that can be captured by exploring the space of higher order mutants.

Search based optimization techniques are well adapted to cost benefit analysis. All that is required is to capture both the cost and benefit as a fitness function in order to quantify costs and benefits. The search process can then seek to balance the costs and benefits in a multi objective search.

## VII. Conclusion

Higher order mutation testing (HOM Testing) can be practical when implemented as a search process that seeks fit mutants (both first and higher order) from the space of all possible mutants. The fitness function can be tailored to the program under test and the specific goals of testing, thereby reducing the number of mutants required (compared to the traditional enumerative approach) and simultaneously increasing the quality and fitness for purpose of the mutants selected by the search.

The fitness function can take account of fault histories, known problems and likely pitfalls and can thereby specifically seek to simulate relevant potential faults that may have gone unnoticed in preceding testing efforts. In this way the search based approach can generate not only smaller sets of fitter mutants, it can also seek to target more realistic sets of mutants. It may even prove possible to use appropriately defined fitness functions to guide the search away from likely equivalent mutants, thereby reducing the impact of the equivalent mutant problem.

For all these reasons, we argue that the Mutation Testing community should de-couple itself from the belief that first order mutants are the only set of worthwhile mutants. This paper has presented a polemic against this 'First Order Restriction' to which the community has been wedded for too long. The First Order Restriction is holding back the development of the field of Mutation Testing. This paper is an unapologetic manifesto for Higher Order Mutation Testing (HOM Testing). Armed with search based optimization, and bold enough to explore the space of higher order mutants, the possibilities for the development of Mutation Testing are, indeed, limitless.

## VIII. Acknowledgements

Many other authors from the Mutation Testing community have helped to form the ideas expressed in this paper. Specifically, we would like to thank Giulio Antoniol, Benoit Baudry, Len Bottaci, Lionel Briand, John Clark, Jim Cordy, Rich DeMillo, Robert Hierons, Dick Lipton, Yves Le Traon, Phil McMinn, Jeff Offutt, Paolo Tonella, Martin Woodward (sadly missed), Eric Wong, Tao Xie and Andreas Zeller for many valuable discussions on Mutation Testing. We would also like to thank Dan Hoffman and William Bader for discussions on Genetic Programming for Mutation testing.

Bill Langdon is supported by EPSRC grant EP/D050863 (SEBASE). Yue Jia is also supported by SEBASE and additionally, by the ORSAS award scheme. Mark Harman is partly supported by SEBASE and also by EPSRC grants EP/G060525 (CREST Platform grant) and EP/F059442 (SLIM: SLIcing state based Models) and by donations from Sogeti and Motorola Inc. Lorna Anderson kindly assisted with proof reading.

## References

[1] A. T. Acree. *On Mutation*. Phd thesis, Georgia Institute of Technology, Atlanta, Georgia, 1980.

[2] K. Adamopoulos, M. Harman, and R. M. Hierons. How to Overcome the Equivalent Mutant Problem and Achieve Tailored Selective Mutation Using Co-evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'04)*, volume 3103 of *LNCS*, pages 1338–1349, Seattle, Washington, USA, 26th-30th, June 2004. Springer.

[3] W. Afzal, R. Torkar, and R. Feldt. A systematic review of search-based testing for non-functional system properties. *Information and Software Technology*, 51(6):957–976, 2009.

[4] H. Agrawal, R. A. DeMillo, B. Hathaway, W. Hsu, W. Hsu, E. W. Krauser, R. J. Martin, A. P. Mathur, and E. Spafford. Design of Mutant Operators for the C Programming Language. Technique Report SERC-TR-41-P, Purdue University, West Lafayette, Indiana, March 1989.

[5] J. H. Andrews, L. C. Briand, and Y. Labiche. Is Mutation an Appropriate Tool for Testing Experiments? In *Proceedings of the 27th International Conference on Software Engineering (ICSE'05)*, pages 402 – 411, St Louis, Missouri, 15-21 May 2005.

[6] D. Baldwin and F. G. Sayward. Heuristics for Determining Equivalence of Program Mutations. Research Report 276, Yale University, New Haven, Connecticut, 1979.

[7] E. F. Barbosa, J. C. Maldonado, and A. M. R. Vincenzi. Toward the determination of sufficient mutant operators for C. *Software Testing, Verification and Reliability*, 11(2):113–136, May 2001.

[8] B. Baudry, F. Fleurey, J.-M. Jezequel, and Y. Le Traon. Genes and Bacteria for Automatic Test Cases Optimization in the .NET Environment. In *Proceedings of the 13th International Symposium on Software Reliability Engineering (ISSRE'02)*, pages 195–206, Annapolis, Maryland, 12-15 November 2002.

[9] T. A. Budd. *Mutation Analysis of Program Test Data*. Phd thesis, Yale University, New Haven, Connecticut, 1980.

[10] B. Cheng and J. Atlee. From state of the art to the future of requirements engineering. In L. Briand and A. Wolf, editors, *Future of Software Engineering 2007*, Los Alamitos, California, USA, 2007. IEEE Computer Society Press. This volume.

[11] S. Y. Chong, P. Tino, and X. Yao. Measuring generalization performance in co-evolutionary learning. *IEEE Transactions on Evolutionary Computation*, 12(4):479–505, August 2008.

[12] M. Daran and P. Thévenod-Fosse. Software Error Analysis: A Real Case Study Involving Real Faults and Mutations. *ACM SIGSOFT Software Engineering Notes*, 21(3):158–177, May 1996.

[13] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6:182–197, Apr. 2002.

[14] R. A. DeMillo, D. S. Guindi, K. N. King, W. M. McCracken, and A. J. Offutt. An Extended Overview of the Mothra Software Testing Environment. In *Proceedings of the 2nd Workshop on Software Testing, Verification, and Analysis (TVA'88)*, pages 142–151, Banff Alberta,Canada, July 1988. IEEE Computer society.

[15] R. A. DeMillo, R. J. Lipton, and F. G. Sayward. Hints on Test Data Selection: Help for the Practicing Programmer. *Computer*, 11(4):34–41, April 1978.

[16] R. A. DeMillo and A. P. Mathur. On the Use of Software Artifacts to Evaluate the Effectiveness of Mutation Analysis in Detecting Errors in Production Software. Technique Report SERC-TR-92-P, Purdue University, West Lafayette, Indiana, 1992.

[17] R. A. DeMillo and A. J. Offutt. Constraint-Based Automatic Test Data Generation. *IEEE Transactions on Software Engineering*, 17(9):900–910, September 1991.

[18] R. A. DeMillo and A. J. Offutt. Experimental Results From an Automatic Test Case Generator. *ACM Transactions on Software Engineering and Methodology*, 2(2):109–127, April 1993.

[19] H. Do and G. Rothermel. A Controlled Experiment Assessing Test Case Prioritization Techniques via Mutation Faults. In *Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM'05)*, pages 411–420, Budapest, Hungary, 25-30 September 2005.

[20] S. Eldh, S. Punnekkat, H. Hansson, and P. Jönsson. Component Testing Is Not Enough - A Study of Software Faults in Telecom Middleware. In *Proceedings of the 19th IFIP International Conference on Testing of Communicating Systems and 7th International Workshop on Formal Approaches to Testing of Software (TestCom'07) and the 7th International Workshop (FATES'07)*, Tallinn, Estonia, 26-29 June 2007.

[21] B. J. M. Grün, D. Schuler, and A. Zeller. The Impact of Equivalent Mutants. In *Proceedings of the 4th International Workshop on Mutation Analysis (MUTATION'09)*, pages 192–199, Denver, Colorado, 1-4 April

2009. IEEE Computer Society. published with *Proceedings of the 2nd International Conference on Software Testing, Verification, and Validation Workshops*.

[22] R. G. Hamlet. Testing Programs with the Aid of a Compiler. *IEEE Transactions on Software Engineering*, 3(4):279–290, July 1977.

[23] M. Harman. The current state and future of search based software engineering. In L. Briand and A. Wolf, editors, *Future of Software Engineering 2007*, pages 342–357, Los Alamitos, California, USA, 2007. IEEE Computer Society Press.

[24] M. Harman, R. Hierons, and S. Danicic. The Relationship Between Program Dependence and Mutation Analysis. In *Proceedings of the 1st Workshop on Mutation Analysis (MUTATION'00)*, pages 5–13, San Jose, California, 6-7 October 2001. published in book form, as *Mutation Testing for the New Century*.

[25] M. Harman and R. M. Hierons. An overview of program slicing. *Software Focus*, 2(3):85–92, 2001.

[26] M. Harman, A. Mansouri, and Y. Zhang. Search based software engineering: A comprehensive analysis and review of trends techniques and applications. Technical Report TR-09-03, Department of Computer Science, King's College London, April 2009.

[27] R. M. Hierons, M. Harman, and S. Danicic. Using Program Slicing to Assist in the Detection of Equivalent Mutants. *Software Testing, Verification and Reliability*, 9(4):233–262, December 1999.

[28] Y. Jia and M. Harman. Constructing Subtle Faults Using Higher Order Mutation Testing. In *Proceedings of the 8th International Working Conference on Source Code Analysis and Manipulation (SCAM'08)*, pages 249–258, Beijing, China, 28-29 September 2008.

[29] Y. Jia and M. Harman. MILU: A Customizable, Runtime-Optimized Higher Order Mutation Testing Tool for the Full C Language. In *Proceedings of the 3rd Testing: Academic and Industrial Conference Practice and Research Techniques (TAIC PART'08)*, pages 94–98, Windsor, UK, 29-31 August 2008. IEEE Computer Society.

[30] Y. Jia and M. Harman. An Analysis and Survey of the Development of Mutation Testing. *IEEE Transactions of Software Engineering*, To appear, 2009.

[31] Y. Jia and M. Harman. Higher Order Mutation Testing. *Journal of Information and Software Technology*, 51(10):1379–1393, October 2009.

[32] K. N. King and A. J. Offutt. A Fortran Language System for Mutation-Based Software Testing. *Software:Practice and Experience*, 21(7):685–718, October 1991.

[33] W. B. Langdon. *Genetic Programming and Data Structures: Genetic Programming + Data Structures = Automatic Programming!*, volume 1 of *Genetic Programming*. Kluwer, Boston, 24 Apr. 1998.

[34] W. B. Langdon and M. Harman. Evolving gzip matches kernel from an nvidia CUDA template. Technical Report TR-10-02, Department of Computer Science, King's College London, London, WC2R 2LS, UK, 5 Feb. 2010.

[35] W. B. Langdon, M. Harman, and Y. Jia. Efficient multi objective MC, GA, GP search and higher order mutation testing. Submitted. Extended version of TAICPART conference paper.

[36] W. B. Langdon, M. Harman, and Y. Jia. Multi objective mutation testing with genetic programming. In $4^{th}$ *Testing Academia and Industry Conference — Practice And Research Techniques (TAIC PART'09)*, pages 21–29, Windsor, UK, 4th–6th September 2009.

[37] W. B. Langdon and A. P. Harrison. Evolving DNA motifs to predict GeneChip probe performance. *Algorithms in Molecular Biology*, 4(6), 19 Mar. 2009.

[38] A. P. Mathur. Performance, Effectiveness, and Reliability Issues in Software Testing. In *Proceedings of the 5th International Computer Software and Applications Conference (COMPSAC'79)*, pages 604–605, Tokyo, Japan, 11-13 September 1991.

[39] A. P. Mathur and W. E. Wong. An Empirical Comparison of Mutation and Data Flow Based Test Adequacy Criteria. Technique report, Purdue University, West Lafayette, Indiana, 1993.

[40] E. S. Mresa and L. Bottaci. Efficiency of Mutation Operators and Selective Mutation Strategies: An Empirical Study. *Software Testing, Verification and Reliability*, 9(4):205–232, December 1999.

[41] A. S. Namin and J. H. Andrews. Finding Sufficient Mutation Operators via Variable Reduction. In *Proceedings of the 2nd Workshop on Mutation Analysis (MUTATION'06)*, page 5, Raleigh, North Carolina, November 2006. IEEE Computer Society.

[42] A. S. Namin and J. H. Andrews. On Sufficiency of Mutants. In *Proceedings of the 29th International Conference on Software Engineering (ICSE COMPANION'07)*, pages 73–74, Minneapolis, Minnesota, 20-26 May 2007.

[43] A. J. Offutt. The Coupling Effect: Fact or Fiction. *ACM SIGSOFT Software Engineering Notes*, 14(8):131–140, December 1989.

[44] A. J. Offutt. Investigations of the Software Testing Coupling Effect. *ACM Transactions on Software Engineering and Methodology*, 1(1):5–20, January 1992.

[45] A. J. Offutt and W. M. Craft. Using Compiler Optimization Techniques to Detect Equivalent Mutants. *Software Testing, Verification and Reliability*, 4(3):131–154, September 1994.

[46] A. J. Offutt, Z. Jin, and J. Pan. The Dynamic Domain Reduction Approach for Test Data Generation: Design and Algorithms. Technical Report ISSE-TR-94-110, George Mason University, Fairfax, Virginia, 1994.

[47] A. J. Offutt, Z. Jin, and J. Pan. The Dynamic Domain Reduction Procedure for Test Data Generation. *Software:Practice and Experience*, 29(2):167–193, February 1999.

[48] A. J. Offutt, A. Lee, G. Rothermel, R. H. Untch, and C. Zapf. An Experimental Determination of Sufficient Mutant Operators. *ACM Transactions on Software Engineering and Methodology*, 5(2):99–118, April 1996.

[49] A. J. Offutt and J. Pan. Detecting Equivalent Mutants and the Feasible Path Problem. In *Proceedings of the 1996 Annual Conference on Computer Assurance*, pages 224–236, Gaithersburg, Maryland, June 1996. IEEE Computer Society Press.

[50] A. J. Offutt, G. Rothermel, and C. Zapf. An Experimental Evaluation of Selective Mutation. In *Proceedings of the 15th International Conference on Software Engineering (ICSE'93)*, pages 100–107, Baltimore, Maryland, May 1993. IEEE Computer Society Press.

[51] R. Poli, W. B. Langdon, and N. F. McPhee. *A field guide to genetic programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008. (With contributions by J. R. Koza).

[52] R. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill Book Company Europe, Maidenhead, Berkshire, England, UK., 3rd edition, 1992. European adaptation (1994). Adapted by Darrel Ince. ISBN 0-07-707936-1.

[53] R. Purushothaman and D. E. Perry. Toward Understanding the Rhetoric of Small Source Code Changes. *IEEE Transactions on Software Engineering*, 31(6):511–526, 2005.

[54] O. Räihä. A survey on search based software design. Technical Report Technical Report D-2009-1, Department of Computer Sciences, University of Tampere, 2009.

[55] M. Sahinoglu and E. H. Spafford. A Bayes Sequential Statistical Procedure for Approving Software Products. In *Proceedings of the IFIP Conference on Approving Software Products (ASP'90)*, pages 43–56, Garmisch Partenkirchen, Germany, September 1990. Elsevier Science.

[56] B. H. Smith and L. Williams. On Guiding the Augmentation of an Automated Test Suite via Mutation Analysis. *Empirical Software Engineering*, 14(3):341–369, 2009.

[57] I. Sommerville. *Software Engineering*. Addison-Wesley, 6th edition, 2001.

[58] F. Tip. A survey of program slicing techniques. *Journal of Programming Languages*, 3(3):121–189, Sept. 1995.

[59] J. M. Voas. PIE: A Dynamic Failure-Based Technique. *IEEE Transactions of Software Engineering*, 18(8):717–726, August 1992.

[60] Wikipedia. Père David's Deer-Wikipedia. http://en.wikipedia.org, 2008.

[61] W. E. Wong. *On Mutation and Data Flow*. Phd thesis, Purdue University, West Lafayette, Indiana, 1993.

[62] W. E. Wong and A. P. Mathur. Reducing the Cost of Mutation Testing: An Empirical Study. *Journal of Systems and Software*, 31(3):185–196, December 1995.