The Theory of Composite Faults

Rahul Gopinath*, Carlos Jensen†, Alex Groce‡
*†School of EECS, Oregon State University, ‡Northern Arizona University
*gopinatr@oregonstate.edu, *cjensen@oregonstate.edu, *agroce@gmail.com

Abstract*—Fault masking happens when the effect of one fault serves to mask that of another fault for particular test inputs. The *coupling effect* is relied upon by testing practitioners to ensure that fault masking is rare. It states that *complex faults are coupled to simple faults in such a way that a test data set that detects all simple faults in a program will detect a high percentage of the complex faults.

While this effect has been empirically evaluated, our theoretical understanding of the *coupling effect* is as yet incomplete. Wah proposed a theory of the *coupling effect* on finite bijective (or near bijective) functions with the same *domain* and *co-domain* and assuming a uniform distribution for candidate functions. This model, however, was criticized as being too simple to model real systems, as it did not account for differing *domain* and *co-domain* in real programs, or for the syntactic neighborhood.

We propose a new theory of fault coupling for general functions (with certain constraints). We show that there are two kinds of fault interactions, of which only the weak interaction can be modeled by the theory of the *coupling effect*. The strong interaction can produce faults that are semantically different from the original faults. These faults should hence be considered as independent atomic faults. Our analysis shows that the theory holds even when the effect of the syntactic neighborhood of the program is considered. We analyze numerous real-world programs with real faults to validate our hypothesis.

I. Introduction

Fault masking occurs when interactions between component faults in a complex fault result in expected (non-faulty) values being produced for particular test inputs. This can result in faults being missed by test cases, and undeserved overconfidence in the reliability of a software system.

The *coupling effect* [1] hypothesis concerns the semiotics[1] of fault masking. It asserts that *"complex faults are coupled to simple faults in such a way that a test data set that detects* ***all simple faults in a program*** *will detect a high percentage of the complex faults."* [2], [3], [4].

This is relied upon by software testers to assert that fault masking is indeed rare. However, our understanding of the *coupling effect* is woefully inadequate. We do not know when (and how often) fault coupling can happen, whether multiple faults will always result in fault coupling, or the effect of increase in number of faults on the number of faults masked. Further, the formal statement of the coupling effect itself is ambiguous and inadequate as it covers only the case where all simple faults are detected. Even worse, it has no unambiguous definition of what a simple (or atomic) fault is. We propose a stronger version of the *coupling effect* (called the *composite fault hypothesis* to avoid confusion):

Composite fault hypothesis: *Tests detecting a fault in isolation will (with high probability κ) continue to detect the fault even when it occurs in combination with other faults.*

We investigate our hypothesis theoretically and empirically. The terms used in this paper are given in Note 1.

A. Theory

Wah et al. [5], [6], [7] investigated the theory of the *coupling effect*, which assumes that any software is built by composition of q independent functions, with a few restrictions:

- Functions have the same *domain* and *range* (order n), and the functions are *bijective*. The non-*bijective* functions are modeled as *degenerate* functions.
- *Separability of faults*: A program with two faults can be split into two independent faulty programs [2].
- *Democratic assumption*: Any applicable function may be chosen as the faulty representation with equal probability.
- The number of functions considered, q, is much smaller than the size of the domain. That is, $q \ll n$. Wah suggests that as q nears n, the *coupling effect* weakens.

For q functions, the survival ratio of I and II order test sets are $\frac{1}{n}$ and $\frac{1}{n^2}$. Wah also makes an observation, used as a heuristic, that the survival ratio of a multi-fault alternate is $\frac{p+1}{n}$ if there are p fault free functions left over after the last faulty function. That is, there are $2^{p-1}-1$ multi fault alternates with last faulty function at p, and the expected number of survivors for q-function composition is:

$$\frac{1}{n^r}\sum_{p=1}^{q}(2^{p-1}-1)(q-p+1)^r$$

for test sets of order r. Wah's analysis lacks wider applicability due to these constraints. Real programs vary widely in their *domain* and *co-domain*. Second, the number of mathematical functions with same *domain* and *co-domain* is not identical to that of programs with same type. Third, the democratic assumption ignores the impact of syntactical neighborhood. That is, it is possible that a *quick sort* implementation can have a small bug, resulting in an incorrect sort. However, it is quite improbable that it is replaced by an algorithm for — say — *random shuffle*, which has the same *domain* and *co-domain* as that of a sorting function. While syntactical nearness does not completely capture semantic nearness, it is closer than assuming any function is a plausible fault for any other function. Next, the separability of complex faults, as we

[1] The relation between syntax and semantics of faults.

[2] Wah assumed this to be true for all general functions, but Section III shows that it is not.

(Semantic) Separability of faults: Two faults present in a function are said to be separable *if and only if* the smallest possible chunk containing both faults can be decomposed into two functions g and h such that each fault is isolated within a single function (providing g_a and h_b as faulty functions), the behavior of composition $h \circ g$ equals the behavior of the original chunk in terms of input and output , and composition $h_b \circ g_a$ equals the behavior of the chunk with both faults.
Simple fault (first order fault): A fault that cannot be *lexically* separated into other independent smaller faults.
Complex fault: (or *higher order* or *combined* fault) A fault that can be *lexically* separated into smaller independent faults.
Constituent fault: A fault that is *lexically* contained in another.
Atomic fault: A fault that cannot be *semantically* separated.
Composite fault: A fault that can be *semantically* separated.
Traditional coupling ratio (C): The ratio between the percentage of complex faults detected and the percentage of simple faults that were detected by a test suite.
Composite coupling ratio (κ): The ratio between the percentage of complex faults detected by the same set of test cases that detected the constituent simple faults, and the percentage of constituent simple faults detected.
Domain *of a function*: The set of all values a function can take as inputs (this is practically the input type of a function).
Co-Domain *of a function*: The set of all values that a function can produce when it is provided with a valid input from its *domain* (this is practically the output type of a function).
Range of a function: The set of all values in *co-domain* that directly maps to a value in the *domain*.
Syntactic neighborhood: The set of functions that can be reached from a given function by modifying its *syntactical representation in a given language* a given number of times.

Note 1. Terms used in this paper

show in Section III, is valid only in certain cases, and does not account for recursion and iteration. Finally, Wah's analysis suggested that the survival ratio of mutants is dependent on the *domain* of the function. We show that the survival ratio of a mutant is actually dependent on the *co-domain* of the function examined, but bounded by *domain*.

We propose a simpler theory of fault coupling that uses a similar model to Wah's, but with relaxed constraints, and incorporates differing *domain* and *co-domain*. We clarify the semantic separability of complex faults, and show how it affects the coupling effect. We also show that certain common classes of complex faults may not be semantically separable. This provides us with a definition of an *atomic fault*: a fault that cannot be semantically separated into simpler faults. This is important because two faults that may be lexically separate but inseparable can be expected to produce a different behavior than either fault considered independently. Further, we consider the impact of syntactic neighborhood. Using both case analysis and statistical argument, we show that our analysis remains valid even when the syntactic neighborhood is considered.

B. Empirical Validation

Lipton et al. [8], [1], and Offutt [2], [3], observed that the tests for first order mutants were sufficient to kill up to 99% of all 2^{nd} order mutants, and 99% of 3^{rd} order mutants sampled. Further research [9], [10], [11], [12], [13], [14] confirms that mutants are coupled to real faults.

Offutt suggests [2], [3] that there are two distinct definitions of coupling involved. The *general coupling effect*: simple faults are coupled to more complex faults such that test data adequate for simple faults will be able to kill a majority of more complex faults. The *mutation coupling effect*: test data adequate for simple first order mutants will be able to detect a majority of more complex mutants. Previous research validates *mutation coupling effect* but not *general coupling effect*.

Our empirical analysis aims to accomplish the following: First, we empirically evaluate the composite coupling ratio κ for numerous real-world projects. This gives us confidence in the assumptions made in the theoretical analysis, and serves to validate the *composite fault hypothesis*. Second, we empirically evaluate the general coupling effect for faults, and compute the traditional coupling ratio C. Lastly, as the size of the faults increase, it is possible that strong interactions also increase, which can produce semantically different faults. Hence, it is important to empirically validate both composite coupling and the general coupling effect for syntactically large fault clusters.

What is the relation between the composite coupling ratio κ and the traditional coupling ratio C? We can regard the composite coupling ratio as a lower limit of the traditional coupling ratio. As we explain further, the general coupling ratio does not discount the effect of strong fault interactions, which can produce complex faults semantically independent from the constituent faults. Hence, C is not bounded by any number, and will often be larger than κ, with $\kappa < 1$.

Contributions:

- We propose the *composite fault hypothesis* that resolves vagueness and ambiguity in the formal statement of the *coupling effect* for non-adequate mutation scores.
- Our theoretical analysis results in the *composite fault hypothesis* for general functions. We find the composite coupling ratio to be $1 - \frac{1}{n}$, where n is the *co-domain*.
- We show that our analysis remains valid even when considering recursion and loops.
- Using 25 projects, we compute the composite coupling ratio κ to be greater than 0.99, with 95% confidence. This helps substantiate the impact of composite coupling.

Our full data set is available for replication[3].

II. Related Work

Fault masking in digital circuits was studied before it was studied in software. Dias [15] studies the problem of fault masking, and derives an algebraic expression that details the

[3] http://eecs.osuosl.org/rahul/icst2017/

number of faults to be considered for detection of all multiple faults. Morell [16] provided a theoretical treatment of fault based testing, and also [17] gave a formal treatment of the *coupling effect*. and shows impossibility of a general algorithm to identify fault coupling. Wah et al. [5], [6], [18], [7] using a simple model of finite functions (the *q-function* model, where q represents the number of functions thus composed) showed that the survival ratio of first and second order test sets are respectively $\frac{1}{n}$ and $\frac{1}{(n^2-n)}$ where n is the order of the *domain* [4]. A major finding of Wah is that *the coupling effect weakens as the system size (in terms of number of functions in an execution path) increases (i.e. q increases), and it becomes unreliable when the system size nears the domain of functions*. Another important finding was that *minimization of test sets has a detrimental effect*. That is, for n faults, one should use n test cases, with each test case able to detect $n-1$ faults (rather than a single fault) to ensure that the test suite minimizes the risk of missing higher order faults due to fault masking. Kapoor [19] proved the existence of the *coupling effect* on logical faults. Voas et al. [20] and later Woodward et al. [21] suggested that functions with a high *DRR* (*domain* to *range* ratio) tend to mask faults. Al-Khanjari et al. [22], found that in some programs there is a strong relationship between *DRD* (Dynamic Range to Domain) ratio and testability.

Androutsopoulos et al. [23] found that one in ten tests suffered from failed error propagation. Clark et al. [24] found that likelihood of collisions was strongly correlated with an information theoretic measure called *squeeziness*, related to the amount of information destroyed on function application.

Our research is an extension of the theoretical work of Wah [6] and Offutt [3], [2]. The major theoretical difference from Wah [6] is that, given a pair of faulty functions that compose, we try to find the probability that, for given test data, the second function masks the error produced by the first one. On the other hand, Wah [6] tries to show that the *coupling effect* exists considering the *entire* program composed of q functions, each having a single fault (given by q in the q-function model). Next, Wah [6] assumes semantic separability of all complex faults. However, as we show, there exist a class of complex faults that are not semantically separable. We make this restriction clear. Further, our analysis shows that the probability of coupling is related to the *co-domain*, not the *domain*, as Wah [6] suggests. In fact, Wah [6] considers only functions which have exactly same *domain* and *range*, and hence are more restricted than our analysis. Finally, we show that even if syntax is considered, our analysis remains valid.

While Offutt [2] evaluates the traditional coupling effect, and shows the empirical relation with respect to *all* simple faults and their combinations, we aim to demonstrate the *composite fault hypothesis* and evaluate the relation between any pair of faults, and the combined fault including both.

III. Theory of Fault Coupling

We start with a function compositional view of programs (similar to Wah [7]). While Wah considered composition of q functions, with as many as q faults, we consider only pairs of faulty functions, since any faulty program with a number of separable faults can be modeled as composition of two functions with (possibly complex) faults.

We have the following assumptions, and simplifications (also made by Wah [6]): our biggest simplification is modeling programs by mathematical functions. While, theoretically, there can be an infinite number of alternatives to any given program, practically, the *domain* and *co-domain* often determines the plausible syntactical alternatives. Next, we assume a finite *domain* and *co-domain*, and consider only total functions. We also assume that faulty versions have same *domain* and *co-domain* (that is, the same type) as that of the non-faulty version. Since any function can be regarded as a single parameter function by considering the input as composed of a tuple of all the original parameters, we restrict our analysis to single parameter functions. While Wah considers how a known number of test inputs (1, 2, 3, or more than 3), some of which can detect some of the component faulty functions, can together detect the composite faulty function, we consider the probability of any single test input that can detect a fault being masked by a new fault. This allow us to significantly simplify our analysis.

Note that the theory *does not* rely on the constituent faults being considered to be simple.

A major idea in our analysis is the semantic separability of faults. Two faults present in a function are said to be separable *if and only if* the smallest possible chunk containing both faults can be decomposed into two functions g and h such that each fault is isolated within a single function (providing g_a and h_b as faulty functions), the behavior of composition $h \circ g$ equals the behavior of the original chunk in terms of input and output , and composition $h_b \circ g_a$ equals the behavior of the function with both faults. A *chunk* here is any small section of the program that can be replaced by an independent function preserving the behavior.

That is, given a function:

```
def functionX(x, y, n)
  for i in (1..n):
      y = faultyA(x)            (1)
      if odd(i): x = faultyB(y)    (2)
      x += 1
```

The lines (1) and (2) together form a chunk. The interaction between the faults and their separability is discussed next.

A. Interaction Between Faults

There are two kinds of interaction between faults: *weak*, and *strong*. *Weak* interactions occur when faults can be semantically separated. That is, given two faults $\hat{a}$ and $\hat{b}$ in a function f, which can be split into $f_{ab} = h_b \circ g_a$, where g_a and h_b are faulty functions, the only interaction between $\hat{a}$ and $\hat{b}$ is because the fault $\hat{a}$ modifies the input of h (or h_b) from $g(i_0)$ to $g_a(i_0)$ (where i_0 is an input for f). That is, the interaction can be represented by a modified input value.

Strong interactions happen when the interpretation of the second fault is affected by the first, and hence faults can't be semantically separated. For example, consider:

```
def swap(x,y): x,y=y,x
```

Say this was mutated into

```
def swap(x,y): x,y=x,y
```

Clearly, there were two independent lexical changes: $x \to y$ and $y \to x$. However, consider the disassembly:

```
>>> dis.dis(swap)
  1     0 LOAD_FAST        1 (y)
        3 LOAD_FAST        0 (x)
        6 ROT_TWO
        7 STORE_FAST       0 (x)
       10 STORE_FAST       1 (y)
       13 LOAD_CONST       0 (None)
       16 RETURN_VALUE
```

The changes in source resulted in intertwined bytecode changes, and hence cannot be separated. Since the faults cannot be separated, *strong* interactions produce faults with different characteristic from the component simple faults, and hence should be considered independent atomic faults[4]. Why should we consider the semantically inseparable faults as independent faults? An intuitive argument is to consider two functions that implement id (these are not strongly interacting). That is, given any value x, we have $g(x) = h(x) = x$. If two faults $\hat{a}$, and $\hat{b}$ occur as we suggest above in g and h, causing inputs i to g_a and inputs j to h_b to fail, then the faulty inputs for $h_b \circ g_a$ are bounded by $i \cup j$, where i represents inputs to f that result in faulty outputs due to faulty g and j, inputs to f resulting in faulty outputs due to faulty h.

What about fault masking? Any input i that failed for g_a could possibly result in an input value that would cause a failure for h_b. For any element outside of i, there is no possibility of two faults acting on it, and hence no possibility of fault masking. However, if the faults are not semantically separable, one cannot make these guarantees, as the faulty inputs may be larger than $i \cup j$ or even completely different. In the general case, when the interaction is weak, we expect the faulty output for up to $i \cup j$.

For formal proof, consider a function f that has *domain* x, represented as $h \circ g$ using two functions. Replacing g with g_a causes $i \in x$ inputs to result in faults. Similarly, replacing h with h_b causes $j \in x$ inputs to f to result in faults. Joining together to form f_{ab}, we know that any of $i \in x$ has a potential to produce a faulty output unless it was masked by h_b. Similarly, any of $j \in x$ also has the possibility of producing a faulty output. Now, consider any element k not in either i or j. It will not result in a faulty output while passing through g_a because it is not in i, further, the value $g_a(k) = g(k) = k_1$. We already know that k_1 would not result in a faulty output from h_b because $k \notin j$. Hence, any element $k \notin i \cup j$ will not be affected by faults $\hat{a}$ and $\hat{b}$.

We can make this assertion only because we can replace g and h separately. If $\hat{a}$ and $\hat{b}$ interacted strongly, any function could potentially replace f. Hence, any element in x may potentially result in a fault when f_{ab} is applied. Harman et al. [25] calls these de-coupled higher order mutants.

Depending on the language used, other features causing strong faulty interaction may exist.

B. Analysis

Consider a program f with two simple faults $\hat{a}$, and $\hat{b}$, which can be applied to f to produce two functions f_a and f_b containing one fault each, and f_{ab} containing both faults (Figure 2). Say such a program can be partitioned into two functions g and h ($f = h \circ g$) with restriction that $\hat{a}$ lies in g, producing alternative g_a, and $\hat{b}$ lies in h producing h_b, such that the new faulty version of f containing both is given by $f_{ab} = h_b \circ g_a$. We note that the particular kind of fault depends on the syntax and semantics of the programming language used, and there can be fault pairs that cannot be separated cleanly. As stated previously, we ignore these kinds of fault pairs as they are syntax dependent and strongly interacting. Hence, no general solution is possible for these faults.

Given that we can distinguish a fault in isolation using a given input, what is the probability that another fault would not result in the masking of that fault for the same input? That is, given a test input i_0 for f, able to distinguish (f, f_a), what is the probability that (f, f_{ab}) can be distinguished by the same input?

Since we know that f_a is distinguished from f, we know that $g_a(i_0) \neq g(i_0)$. Hence, the function h_b will have a different input than h. Thus, the question simplifies to: given an alternate input for function h (or anything that can be substituted in its place), what is the probability that a faulty h, with the new input $g_a(i_0)$ will result in same output as the old h, with the old input $g(i_0)$?

Let us assume for simplicity that functions g and h have fixed *domain* and a *co-domain* given by $g \in G : \mathbb{L} \to \mathbb{M}$ and $h \in H : \mathbb{M} \to \mathbb{N}$. That is, h belongs to a set of functions H, which has a *domain* M, and a *co-domain* N such that $m = |M|$ and $n = |N|$. Considering all possible functions in H, with the given *domain* and *co-domain*, there will be n^m unique functions in H (separated by at least one different $\{input, output\}$ pair).

The only constraint on h_b we have is that $h_b(g_a(i_0))$ should result in the same output as $h(g(i_0))$. We are looking for functions that can vary in every other $\{input, output\}$ pair except for the pair given by $\{g_a(i_0), h(g(i_0))\}$. There are n^{m-1} functions that can do that out of $|H| = n^m$ functions. That is, the composite coupling ratio is given by $\kappa = 1 - \frac{n^{m-1}}{n^m}$, which is simplified to $1 - \frac{1}{n}$ of the total number of eligible functions where m is the size of *domain*, and n is the size of *co-domain* of the function. That is, given any test input, the probability of the composite coupling effect where the fault in one constituent is not masked by the fault in another is $1 - \frac{1}{n}$, and $\frac{1}{n}$ tends to be very small when the *co-domain* of the function (n) is large.

A symmetric argument can be made when the function fixed is h, and g varies. There are m^l functions in G, of which m^{l-1} can be used as a replacement without affecting

[4] Wah [6] ignores strong interaction of faults.

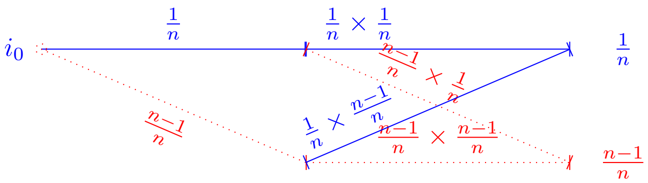
i_0
$\frac{1}{n}$
$\frac{1}{n} \times \frac{1}{n}$
$\frac{1}{n}$
$\frac{n-1}{n} \times \frac{1}{n}$
$\frac{n-1}{n}$
$\frac{1}{n} \times \frac{n-1}{n}$
$\frac{n-1}{n} \times \frac{n-1}{n}$
$\frac{n-1}{n}$

Fig. 1. Recursive interaction. The *blue solid* lines represent the masking values where the values are same as what would be expected before the fault was introduced, and the *red dotted* lines represent values that are different from the non-faulty version so that faults could be detected.

$\{input, output\}$, in which case, the probability of composite coupling effect is $1 - \frac{1}{m}$ where m is the *co-domain*[5].

C. Recursion and Iteration

Recursion and *iteration* can present challenges to our analysis. For example, consider:

```
while y > 0: y = h(g(y)
```

The two functions g and h are otherwise independent. However, the input of h influences g, and vice versa. Here, we do not know when the loop will end, and any faults will be detected. The faults may be detected after a larger or smaller number of iterations than the non faulty version. Hence, we consider the chances of propagation of the faulty value after each iteration. That is, if a faulty value is present after executing the function g_a once, what are the chances that it will be caught at the end of each iteration?

Let f denote the program segment composed of g and h. After the first iteration of f, we will have $\frac{1}{n}$ possibility for fault masking as we discussed before, and $\frac{n-1}{n}$ possibility for detectable faulty values. Now, consider the next iteration. In this case, of the original $\frac{1}{n}$ masked outputs, $\frac{1}{n}$ will again be masked, for a total of $\frac{1}{n^2}$, and the remaining $\frac{(n-1)}{n^2}$ will have a value that is faulty. Consider the original $\frac{n-1}{n}$ that had faulty values in the first iteration. Out of that, $\frac{1}{n}$ will be masked in the second iteration (i.e. $\frac{n-1}{n^2}$). Similarly, $\frac{(n-1)^2}{n^2}$ of the original faulty outputs will remain faulty. That is, after second iteration, we will have $\frac{1}{n^2} + \frac{n-1}{n^2} = \frac{1}{n}$ masked output values. Similarly, we will have $\frac{n-1}{n^2} + \frac{(n-1)^2}{n^2} = \frac{n-1}{n}$ possibility of faulty output values. That is, after each iteration, we will have $\frac{1}{n}$ possibility of fault masking (See Figure 1). Hence, composite fault hypothesis will hold even for recursion and iteration.

1) Premature loop exits: What if a fraction of inputs – say x – diverge so much (crashes or gets detected by asserts) that they never make it through all iterations? We can model this as the case where the remaining fraction ($y = 1 - x$) of inputs belong to a function with reduced *domain* and hence *co-domain*. This is more involved because functions with a smaller *co-domain* are more prone to fault masking. We need to show that the total fraction of masked values is lesser than the original $\frac{1}{n}$, or show the other side

$$x + \frac{ny - 1}{ny} \geq \frac{n-1}{n} \qquad (1)$$

We assume that $nx \geq 1$ (at least one input causes a crash) and $ny \geq 1$ (at least one input reaches the end – otherwise, there is no fault masking involved).

We simplify Equation 1 by first making the denominator the same (ny) and then simplifying, which results in the equation $nxy + ny - 1 \geq ny - y$. On expanding y to $1 - x$, and simplifying, we get $ny \geq 1$. Note that this was our original assumption. Hence, premature loop exits result in a stronger coupling between faults.

What happens if instead of a fixed fraction, we have say $r\%$ input values detected at the end of each iteration? Of course, any finite number of loops could be modeled as we did above. If instead, we rely on the crashes alone to distinguish faulty values, we are still in luck. Each iteration detects $r\%$ of the input values, and the remaining $q = 1 - r\%$ of the values restart the iteration. This results in $r + rq + rq^2 + \ldots rq^{n-1}$ values getting detected at the end of n^{th} iteration. This infinite sum converges to 1. That is, no faults will be masked.

2) Different execution paths: Another wrinkle is the pattern where iteration proceeds in different paths during different executions. For example:

```
for i in 1..10:
  if odd(i): x = g(y)
  else: y = h(x)
```

In programs such as this, one may unroll the loop, i.e.

```
for i in 1..10:2:
    x = g(y)
    y = h(x)
```

which can make it amenable to the above treatment. Recursion can be resolved similarly. We do not claim that this is exhaustive. There could exist other patterns of recursion or iteration that do not fit this template. However, most common patterns of recursion and iteration could be captured in this pattern.

Can we extend the bounds we found ($i \cup j$ for faulty outputs) to recursion? Unfortunately, it is possible for a faulty function to interact with its own output during recursion, and hence mask a failure. Hence, we can not bound the failure causing inputs in a doubly faulty function that incorporates recursion.

D. Accounting for Multiple Faults

What happens when there are multiple faults? Say, we have a system modeled by $p \circ q \circ r \circ s \circ t \circ u$, where any of the functions may be faulty or not faulty, for example $p_a \circ q \circ r_b \circ s_c \circ t_d \circ u$. We can not directly apply the technique in recursion because there are non-faulty functions interspersed. The thing to remember here is that a non faulty function immediately adjacent to a faulty function can together be considered a faulty function. Hence, the above reduces to $(p_a \circ q) \circ r_b \circ s_c \circ (t_d \circ u)$, or equivalently $pq_a \circ r_b \circ s_c \circ tu_d$. This is now amenable

[5] We note that the logic of probability is very similar to Wah [6], and this is the same value derived by Wah for single test input, where n is the *domain* of the function as Wah does not consider functions that have a different *domain* and *co-domain*.

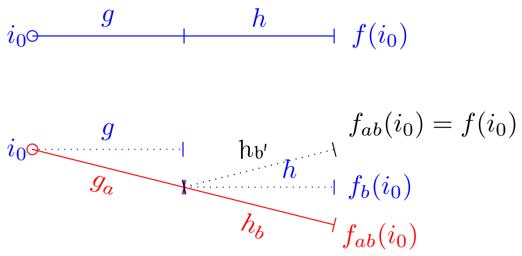

Fig. 2. Fault interaction ($g_a(i_0)$ is masked by $h_{b'}$)

to the treatment in Figure 1 because each function now can produce $\frac{1}{n}$ non-faulty and $\frac{n-1}{n}$ faulty outputs. An additional complication is that a general expression is not possible unless we simplify further, and assumes that *domain* and *co-domain* of all functions are same. With this simplification, even when we consider a number of faulty functions, the mean ratio of fault masking remains the same at $\frac{1}{n}$. Indeed, this is one of the significant differences from Wah. Wah does not attempt to collapse the non-faulty functions to their neighbours. Why do we do this? Because we know that each faulty function on its own was detected by the test suite. That is, we know that $p \circ q_a \circ r \circ s \circ t \circ u$ would have been detected. Hence, we can certainly consider $pq_a \circ r \circ s \circ t \circ u$ as the set of functions where the function pq_a is the function with an atomic fault.

E. Dynamically Checked Languages

In the case of *dynamically checked* or *unityped* languages, every single function has the same type (*domain*, *co-domain*), and alternatives are large (but finite), because one may not identify a faulty input type until execution. Hence, we can expect large composite coupling ratio.

F. Impact of Syntax

In order to model composite coupling, we assumed that all faults are equally probable, which is often not the case, with faults that are closer syntactically being more probable than faults which are not in the syntactic neighborhood of correctness. In fact, we have some reasonable estimate of the distribution of size of faults that programmers make [26].

Implementation of functions as code need not necessarily follow the same distribution as that of their mathematical counterparts. For example, for mathematical functions, there exist only 4 functions that map from a boolean to a boolean. However, there can be an infinite number of program implementations of that function. The way it can be made tractable is again to consider the human element. The *competent programmer hypothesis* suggests that faulty programs are close (syntactically) to the correct versions. So one need only consider a limited number of alternatives (the number of which is a function of the size of the correct version, if one assumes that each token may be legally replaced by another).

As soon as we speak about syntactic neighborhood, the syntax of a language can have a large influence on which faults can be considered to be in a neighborhood. However, we note that most languages seem to follow a similar distribution of faults with a size below 10 tokens for 90% of faults [26].

Let us call the original input to h, $g(i_0) = j_0$, and the changed value $g_a(i_0) = j_a$. Similarly, let $f(i_0) = k_0$, $f_a(i_0) = k_a$, $f_b(i_0) = k_b$, and $f_{ab}(i_0) = k_{ab}$. Given two inputs i_0, and i_1 for a function f, we call i_0, and i_1 semantically close if their execution paths in f follow equivalent profiles, e.g taking the same branches and conditionals. We call i_0 and i_1 semantically far in terms of f if their execution profiles are different.

Consider the possibility of masking the output of g_a by h_b ($h_{b'}$ in Figure 2)). We already know that $h(j_a) = k_a$ was detected. That is, we know that j_a was sufficiently different from j_0, that it propagated through h to be caught by a test case. Say j_a was semantically far from j_0, and the difference (i.e the skipped part) contained the fault $\hat{b}$. In that case, the fault $\hat{b}$ would not have been executed, and since $k_{ab} = k_a$, it will always be detected.

On the other hand, say j_a was semantically close to j_0 in terms of g and the fault $\hat{b}$ was executed. There are again three possibilities. The first is that $\hat{b}$ had no impact, in which case the analysis is the same as before. The second is that $\hat{b}$ caused a change in the output. It is possible that the execution of $\hat{b}$ could be problematic enough to always cause an error, in which case we have $k_{ab} = k_b$ (error), and detection. Thus masking requires k_{ab} to be equal to k_0.

Even if we assume that the function h_b is close syntactically to h, and that this implies semantic closeness of functions h and h_b, we expect the value k_{ab} to be near k_a, and not k_0. This suggests that masking, even when considered in the light of syntactical neighborhood, is still unlikely, but this belief requires empirical verification since we are unable to assign probabilities to the cases above. Our empirical data (provided in the next section of this paper) should shed light on the actual incidence of masking when syntactic/semantic neighborhoods are taken into account, since real faults are likely in the syntactic and semantic neighborhood of the correct code.

A statistical observation can further buttress our argument. We know that if all functions were equally probable, fault masking has low probability. Now, consider the functions that are syntactically close to a given function. For most input values, we can assume that the syntactically close functions will have same output as that of the given function, more so than functions that are far away lexically. If h did not mask a value originally, (which we know since we were able to detect fault $h(g_a(i_0))$), then the syntactically close functions to h will with a higher probability than a uniform sample, produce the same value as $h(g_a(i_0))$, which will be detected as faulty.

G. Can Strong Interaction be Avoided?

The *coupling effect* argues that if a test suite can find all atomic faults, then by composite fault hypothesis, a large percentage (κ) of complex faults will also be found. However, when can one assert that all atomic faults have been found? Any strong fault interaction has the potential to produce an atomic fault.

Given that the strong interaction is dependent on the execution, can runtime environment or compiler order computation so that strong interaction is no longer present?

Consider the function *swap (a,b) = (b,a)* that we examined earlier. We see how one may mistakenly use *id (a,b) = (a,b)* instead, and cause a strong interaction. Now, the question is, does there exist a way to split the two functions, so that the condition of separability can be satisfied? Given that there are only four possible functions that can operate on a tuple, (*swap (a,b) = (b,a)*, *id (a,b) = (a,b)*, *dupleft (a,b) = (a,a)*, *dupright (a,b) = (b,b)*) we could check it exhaustively. The condition is that the functions representing single faults should individually cause a detectable deviation on their own, and on composition, result in same behavior as *id*. Now, it can be seen that, neither of the single fault functions can behave like $swap$ since that represents *no* fault, so they can not behave like id, since that suggests that the other faulty function behaves like $swap$. Hence, no compiler or runtime environment can remove the strong interaction in *swap*.

Where can we expect strong interaction to appear? While we can not provide an exhaustive overview of possible language features, we can demonstrate that even very simple languages such as the λ-calculus are vulnerable. Consider the λ-calculus expression $\lambda x\, y\, .y\, x$, and its faulty version $\lambda x\, y\, .x\, y$. There are two lexical points where the faults have been injected $\{x \rightarrow y, y \rightarrow x\}$. However, they cannot be separated out. That is, even such simple features can cause strong interaction.

IV. Methodology for Assessment

Our methodology was guided by two principles [27]: We sought to minimize the number of variables, and tried to be as general as possible. Hence, we selected Apache commons for analysis.

For our set of projects, we iterated through their commit logs, and generated reverse patches for each commit. For each patch thus created, we applied the patch on the latest repository, and removed any changes to the test directory, thus ensuring that the test suite we tested with was always the latest. Any patch that resulted in a compilation error was removed. This resulted in a set of patches for each project that could be independently applied. The complete test suite for the project was executed on each of the patches left, and any patch that did not result in a test failure was removed. The failed test cases that corresponded to each patch were thus collected. At this point, we had a set of patches that introduce specific test case failures. The set of Apache projects, along with the set of reverse patches thus found, are given in Table I.

TABLE I
Apache Commons Libraries

	Projects	SLOC	TLOC	CPatches	Fails
1	commons-bcel	30,175	3,155	148	6
2	commons-beanutils	11,640	21,665	63	5
3	commons-cli	2,665	3,768	71	5
4	commons-codec	6,599	11,026	179	4
5	commons-collections	27,820	32,913	333	16
6	commons-compress	18,746	13,496	430	65
7	commons-configuration	26,793	37,806	322	78
8	commons-csv	1,421	3,168	150	8
9	commons-dbcp	11,259	8,487	98	18
10	commons-dbutils	3,064	3,699	43	1
11	commons-discovery	2,320	268	171	1
12	commons-exec	1,757	1,601	90	5
13	commons-fileupload	2,389	1,946	129	8
14	commons-imaging	31,152	6,525	174	4
15	commons-io	9,813	17,968	177	18
16	commons-jexl	10,921	9,509	54	10
17	commons-jxpath	18,773	6,137	10	2
18	commons-lang	25,468	43,981	571	49
19	commons-mail	2,720	3,869	48	5
20	commons-math	84,809	89,336	954	142
21	commons-net	19,749	7,465	454	21
22	commons-ognl	13,139	6,873	190	3
23	commons-pool	5,242	8,042	149	12
24	commons-scxml	9,524	5,119	74	7
25	commons-validator	6,681	7,926	126	17

SLOC is the program size in LOC, TLOC is the test suite size in LOC, CPatches is the number of compiled patches, and Fails is the number of test failures.

We conducted our remaining analysis in two parts. For the first part, we generated patch pairs by joining together two random patches for any given project. For the projects where the total number of unique pairs was larger than 100, we randomly sampled 100 of the pairs produced. After removing patch combinations that resulted in compilation errors, we had 1,126 patch combinations. We evaluated the test suite of each project against the pair-patches thus generated, and collected the test cases which failed against these. Adopting the terminology of Jia et al. [28], out of 1,126, we had 1,126 coupled higher order mutants, and 56 subsuming mutants.[6] Out of these, there were only 2 strongly subsuming mutants.

We tried to reduce the number of external variables further for the second part, and chose a single large project — *Apache commons-math*. We generated a set of combined patches by joining 2, 4, 8, 16, 32, and 64 patches at random, and evaluated the test suite for commons-math against each of these k^{th} order patches. We removed all patches that resulted in any compilation errors, producing 342 patch combinations.

For both parts of our analysis, we generated two sets. The first set containing the unique failures from the constituent faults in isolation, and the second containing the combined patches.

V. Analysis

There are two questions that we tackle here. The first investigates the fraction of test cases that detect any of the constituent mutants that also detect the combined mutant. That is, evaluates the following prediction from the model: "Given two faults, and the test cases killing each, (assuming a sufficiently large *domain* and *co-domain*, and ignoring the effects of strong interaction), there is a high probability for the same test cases to kill the combined fault."

The second investigates the general coupling effect. Since the general coupling ratio does not distinguish between strong and weak interaction, this also serves as an evaluation of the

[6] Of course, our patches are derived from actual faulty code, not mutants in the traditional sense of generated modification.

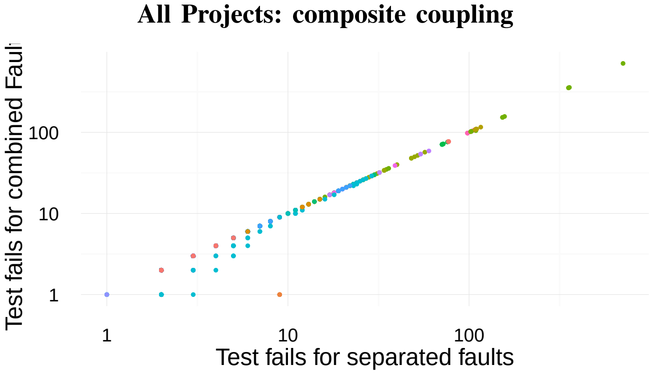

Fig. 3. The size of set of the test cases able to detect the faults when they were separate is in the *x-axis*, and the *subset of the same test cases* able to detect the combined fault is in the *y-axis*. Colors correspond to projects.

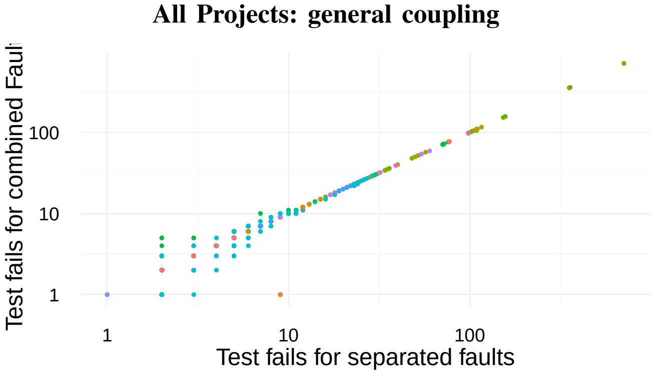

Fig. 4. The size of the set of test cases able to detect the faults when they were separate is the *x-axis*, and the *set of all test cases* able to detect the combined fault is in the *y-axis*. Colors correspond to projects.

strong interaction between faults where inputs other than the original i and j – that is, outside $i \cup j$ – becomes faulty (where i represents faulty inputs to f due to faults in h, and j represents faulty inputs to f due to faults in g).

Indeed, we believe that strong interaction between different faults is rarer than weak interaction. While there is no easy way to verify it, one may look at the newer faults (new test failures) that are introduced by a combination of patches when compared to the original patches as instances of strong fault interaction, which may be considered a reasonable proxy. Our empirical evaluation does not require individual patches to be simple faults. Our theory suggests that irrespective of whether the faults are complex or not, we can expect the same fault masking probability.

A. All Projects

This section investigates fault pairs from all projects.

1) The Composite Fault Model: Here, we try to answer the question: *what percentage of test cases detecting constituent faults can detect the complex faults?*

Figure 3 plots the set of test cases able to detect the faults when they were separate with the set of test cases able to detect the combined fault. To analyze the fraction of test cases expected to detect the combined mutant, we evaluate the regression model given by:

$$\mu\{AfterT|BeforeT\} = \beta_0 + \beta_1 \times BeforeT \quad (2)$$

where $BeforeT$ is the size of the test suite that includes all test cases that can detect both faults separately, and $AfterT$ is the size of the test suite which is a subset of $BeforeT$ that can detect the fault pair when combined. We force β_0 to zero to account for the fact that if no test cases detected the original mutant, then the question of their fraction does not arise. This linear regression model lets us predict the number of test fails for combined faults from the test fails for separated faults.

We note that we are interested in β_1 for another purpose. β_1 is also the composite coupling ratio κ. Thus this regression provides us with a model for prediction, its goodness of fit (R^2), and also the composite coupling ratio.

2) The General Coupling Model: Figure 4 plots the general coupling of faults. We evaluate the following regression model.

$$\mu\{NewT|BeforeT\} = \beta_0 + \beta_1 \times BeforeT \quad (3)$$

where $BeforeT$ is the size of the test suite that includes all test cases that can detect both faults separately, and $NewT$ is the size of the test suite that can detect the fault pair when combined. Note that we do not set $\beta_0 = 0$ here as the combined fault pair may be detected by a new test case even if its constituents were not detected. In fact, β_0 represents the complex faults that became detectable due to interaction even though the constituent faults are not detectable.

However, if one wishes to investigate the general coupling ratio, we have to investigate a simpler regression model, because the general coupling ratio does not permit an intercept.

$$\mu\{NewT|BeforeT\} = \beta_1 \times BeforeT \quad (4)$$

Here, similar to the previous section, β_1 corresponds to the general coupling ratio C.

3) Strong fault interaction: The incidence of strong fault interaction may be ascertained by the average number of new test cases that failed for the combined patch. Note that this number is not exhaustive, as some of the original test cases may fail for new faulty behavior too, even if the behavior is not same as that of the component faults.

B. Apache Commons-math

1) The Composite Fault Model: We try to answer the question *what percentage of test cases detecting constituent faults can detect the complex faults?* for Commons-math. We rely on the regression given by Equation 2. Figure 5 plots test cases able to detect the faults when they were separate with the test cases able to detect the combined fault.

2) The General Coupling Model: We rely on the regressions given by Equation 3 and Equation 4. Figure 6 plots the general coupling of faults for *Apache commons math*.

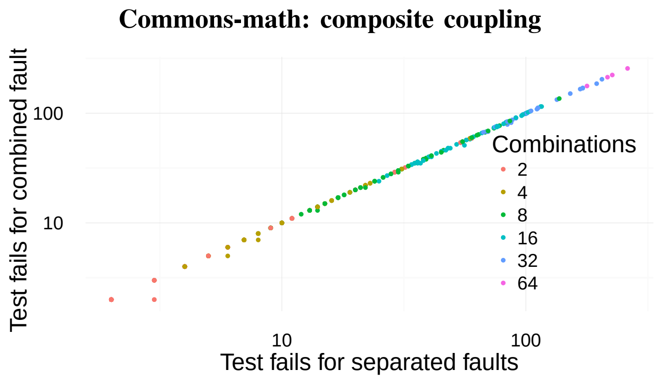

Fig. 5. The set of test cases able to detect the faults when they were separate is in the *x-axis*, and the *subset of the same test cases* able to detect the combined fault is in the *y-axis*. Colors correspond to number of patch combinations.

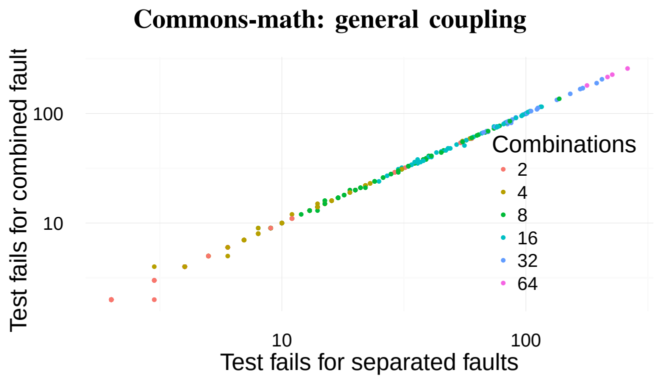

Fig. 6. The set of test cases able to detect the faults when they were separate is in the *x-axis*, and the *set of all test cases* able to detect the combined fault is in the *y-axis*. Colors correspond to number of patch combinations.

3) Strong fault interaction: The incidence of strong fault interaction may be ascertained by the average number of new test cases that failed for the combined patch. The difference of note here is that the number of patches are larger, and hence the chances of strong interaction are correspondingly larger.

VI. Results

A. All Projects

The results for regression for Equation 2 for all projects is given in Table II. The correlation between the dependent and independent variable is 0.99975. The composite coupling ratio was found to be 0.99916. The results for regression for Equation 3 for all projects is given in Table III. The correlation between the dependent and independent variable is 0.99967. The results for regression for Equation 4 for all projects is given in Table IV. The general coupling ratio was found to be 0.99931. Further, the mean number of faulty test cases that were not present in the component faults were found to be 0.0417. See Table V for the summary.

TABLE II
All projects for composite coupling ratio. R^2 =0.99975

	Estimate	Std. Error	t value	Pr(>|t|)
SeparateFaults	0.9992	0.0005	2,116.13	0.0000

TABLE III
All projects R^2 =0.99967

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	-0.0399	0.0189	-2.12	0.0343
SeparateFaults	0.9997	0.0005	1,847.83	0.0000

TABLE IV
All projects for general coupling ratio. R^2 =0.9997

	Estimate	Std. Error	t value	Pr(>|t|)
SeparateFaults	0.9993	0.0005	1,939.82	0.0000

B. Apache commons-math

The results for regression for Equation 2 for all projects is given in Table VI. The correlation between the dependent and independent variable is 0.99983. The composite coupling ratio was found to be 0.98956. The results for regression for Equation 3 for commons-math is given in Table VII. The correlation between the dependent and independent variable is 0.99971. The results for regression for Equation 4 for commons-math is given in Table VIII. The general coupling ratio was found to be 0.9944. Further, the mean number of faulty test cases that were not present in the component faults were found to be 0.137. See Table IX for the summary.

VII. Discussion

Fault masking is one of the key concerns in software testing. The *coupling effect* hypothesis asserts that fault masking is rare. Unfortunately, little is known about the theory behind fault coupling. We study the *coupling effect* and fault masking using theoretical and empirical methods.

TABLE V
Summary for all projects.

	SeparateFaults	JoinedFaults	RemovedFaults	AddedFaults
bcel	27.73	27.73	0.00	0.00
beanutils	4.80	1.60	3.20	0.00
cli	7.60	7.60	0.00	0.00
codec	2.50	2.50	0.00	0.00
collections	16.49	16.49	0.00	0.00
compress	11.60	11.60	0.00	0.00
configuration	37.19	37.16	0.04	0.02
csv	2.00	2.00	0.00	0.00
dbcp	10.60	10.91	0.01	0.32
exec	16.50	16.50	0.00	0.00
fileupload	4.64	4.64	0.00	0.00
imaging	6.50	6.50	0.00	0.00
io	7.93	7.55	0.54	0.17
jexl	3.58	3.56	0.02	0.00
jxpath	3.00	3.00	0.00	0.00
lang	4.46	4.46	0.00	0.00
mail	2.30	2.30	0.00	0.00
math	7.67	7.64	0.03	0.00
net	4.13	4.13	0.00	0.00
ognl	27.00	27.00	0.00	0.00
pool	4.48	4.45	0.05	0.02
scxml	36.62	36.62	0.00	0.00
validator	3.07	3.06	0.01	0.00

TABLE VI
C-MATH FOR COMPOSITE COUPLING RATIO. R^2 =0.99983

	Estimate	Std. Error	t value	Pr(>|t|)
SeparateFaults	0.9896	0.0007	1,418.94	0.0000

TABLE VII
C-MATH R^2 =0.99971

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	0.0924	0.0482	1.92	0.0563
SeparateFaults	0.9933	0.0009	1,090.92	0.0000

Our theoretical evaluation of the *composite fault hypothesis*. shows that for any pair of separable faults, composite coupling effect exists. We find that composite coupling ratio $\kappa = 1 - \frac{1}{n}$, where n is the *co-domain* of the function being considered, and that syntactical neighborhood does not have an adverse impact on our result. Further, while Wah suggests that, as system size increases the *coupling effect* weakens exponentially, our results suggest that the mean coupling ratio remains the same at $\frac{1}{n}$.

Why is our prediction on fault masking so important? Basic testing relies on fault masking. Say you are unit testing a function with multiple faults, and some of the faults are left undetected due to fault masking. Wah's analysis suggests that when we integrate these units into a larger system, the faults in the larger system have a much higher (indeed exponential) tendency to self correct, and avoid failure due to masking. Our analysis suggests that even on larger systems composed of smaller systems, the rate of fault masking remains the same.

We proposed the existence of strongly interacting faults, which cannot be accounted for within the formal coupling theory. Our empirical analysis (see Table V and Table IX) indicates that strong interaction is possibly rare, occurring at a similar frequency as fault masking. Figure 3 suggests that while there is some reduction in the combined faults for the faults with smaller semantic footprint (as given by the number of test cases that failed for that fault) with respect to constituent faults, the difference vanishes when the size of the fault increases. This same effect is also seen in Figure 5.

The results for regression (Equation 2) also suggest a similar observation — that test cases that are able to detect a fault in isolation will with very high probability detect the same fault in combination with other faults.

TABLE VIII
C-MATH FOR GENERAL COUPLING RATIO. R^2 =0.99983

	Estimate	Std. Error	t value	Pr(>|t|)
SeparateFaults	0.9944	0.0007	1,401.55	0.0000

TABLE IX
SUMMARY FOR ALL COMMONS-MATH.

	SeparateFaults	JoinedFaults	RemovedFaults	AddedFaults
2	7.67	7.64	0.03	0.00
4	14.67	14.70	0.05	0.07
8	30.53	30.42	0.17	0.06
16	59.25	59.09	0.42	0.25
32	109.85	108.96	1.37	0.48
64	220.25	219.50	3.00	2.25

Overall, our statistical analysis suggests that there is a very high probability (between $\{0.998$ & $1.000\}$ for all projects, and $\{0.988$ & $0.991\}$ for commons-math — 95% confidence interval with statistical significance $p < 0.0001$) that when two faults are paired to produce a combined fault, any test cases that detected either of the faults continue to detect the combined fault.

Our results for Table IV suggests that between $\{0.998$ & $1.000\}$ of complex faults are caught (95% confidence interval, $p < 0.0001$). This is again confirmed by the deeper analysis of *Apache commons-math*, using larger size faults in Table VIII which suggests that between $\{0.993$ & $0.996\}$ fraction of complex faults are caught (95% confidence interval, $p < 0.0001$). We note that this is the first confirmation of the *general coupling effect* (unlike the *mutation coupling effect* which has been validated multiple times). Why is validating the general coupling effect important? We already know that faults emulated by traditional mutants are only a subset of the possible kinds of faults (Just et al. [14] found that up to 27% of faults were inadequately represented by mutants). Hence, it is important to verify the *general coupling effect* using real faults so that our results are applicable for faults in general, and especially for possible future mutation operators. Indeed, the *mutation coupling effect* has been validated multiple times, and we do not attempt it again here.

VIII. CONCLUSION

The *coupling effect* hypothesis is a general theory of fault interaction, and is used to quantify *fault masking*. It also finds use in mutation analysis. While there is compelling empirical evidence for the *coupling effect*, our theoretical understanding is lacking. The extant theory by Wah is too restrictive to be useful for real world systems. We address this limitation, and provide a stronger, modified version of the theory called the *composite fault hypothesis*.

Our theoretical analysis suggests that the composite fault hypothesis has a high probability of occurring ($1 - \frac{1}{n}$, where n is the *co-domain* of the function under consideration) under the assumptions of total functions, finite *domain*, and separability of faults, *irrespective of the size of the system*.

Our empirical study provides validation, and an empirical approximation of the composite coupling ratio κ (0.99), with 99% of the test cases that detected a fault in isolation continuing to detect it when it is combined with other faults.

REFERENCES

[1] R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "Hints on test data selection: Help for the practicing programmer," *Computer*, vol. 11, no. 4, pp. 34–41, 1978.

[2] A. J. Offutt, "Investigations of the software testing coupling effect," *ACM Transactions on Software Engineering and Methodology*, vol. 1, no. 1, pp. 5–20, 1992.

[3] ——, "The Coupling Effect : Fact or Fiction?" *ACM SIGSOFT Software Engineering Notes*, vol. 14, no. 8, pp. 131–140, Nov. 1989.

[4] Y. Jia and M. Harman, “An analysis and survey of the development of mutation testing,” *IEEE Transactions on Software Engineering*, vol. 37, no. 5, pp. 649–678, 2011.

[5] K. S. H. T. Wah, “Fault coupling in finite bijective functions,” *Software Testing, Verification and Reliability*, vol. 5, no. 1, pp. 3–47, 1995.

[6] ——, “A theoretical study of fault coupling,” *Software Testing, Verification and Reliability*, vol. 10, no. 1, pp. 3–45, 2000.

[7] ——, “An analysis of the coupling effect I: single test data,” *Science of Computer Programming*, vol. 48, no. 2, pp. 119–161, 2003.

[8] R. J. Lipton and F. G. Sayward, “The status of research on program mutation,” in *Digest for the Workshop on Software Testing and Test Documentation*, December 1978, pp. 355–373.

[9] R. A. DeMillo and A. P. Mathur, “On the use of software artifacts to evaluate the effectiveness of mutation analysis for detecting errors in production software,” Software Engineering Research Center, Purdue University, West Lafayette, IN,, Tech. Rep. SERC-TR92-P, 1991.

[10] J. H. Andrews, L. C. Briand, and Y. Labiche, “Is mutation an appropriate tool for testing experiments?” in *International Conference on Software Engineering*. IEEE, 2005, pp. 402–411.

[11] J. H. Andrews, L. C. Briand, Y. Labiche, and A. S. Namin, “Using mutation analysis for assessing and comparing testing coverage criteria,” *IEEE Transactions on Software Engineering*, vol. 32, no. 8, pp. 608–624, 2006.

[12] H. Do and G. Rothermel, “On the use of mutation faults in empirical assessments of test case prioritization techniques,” *IEEE Transactions on Software Engineering*, vol. 32, no. 9, pp. 733–752, 2006.

[13] N. Li, U. Praphamontripong, and J. Offutt, “An experimental comparison of four unit test criteria: Mutation, edge-pair, all-uses and prime path coverage,” in *International Conference on Software Testing, Verification and Validation Workshops*. IEEE, 2009, pp. 220–229.

[14] R. Just, D. Jalali, L. Inozemtseva, M. D. Ernst, R. Holmes, and G. Fraser, “Are mutants a valid substitute for real faults in software testing?” in *ACM SIGSOFT Symposium on The Foundations of Software Engineering*. Hong Kong, China: ACM, 2014, pp. 654–665.

[15] F. J. O. Dias, “Fault masking in combinational logic circuits,” *IEEE Trans. Comput.*, vol. 24, no. 5, pp. 476–482, May 1975.

[16] L. J. Morell, “A theory of fault-based testing,” *IEEE Transactions on Software Engineering*, vol. 16, no. 8, pp. 844–857, 1990.

[17] ——, “A model for code-based testing schemes,” in *Fifth Annual Pacific Northwest Software Quality Conf*, 1987, p. 309.

[18] K. S. H. T. Wah, “Theoretical insights into the coupling effect,” in *Mutation testing for the new century*, W. E. Wong, Ed. Norwell, MA, USA: Kluwer Academic Publishers, 2001, pp. 62–70.

[19] K. Kapoor, “Formal analysis of coupling hypothesis for logical faults,” *Innovations in Systems and Software Engineering*, vol. 2, no. 2, pp. 80–87, 2006.

[20] J. M. Voas and K. W. Miller, “Semantic metrics for software testability,” *The Journal of Systems and Software*, vol. 20, no. 3, pp. 207 – 216, 1993.

[21] M. R. Woodward and Z. A. Al-Khanjari, “Testability, fault size and the domain-to-range ratio: An eternal triangle,” *ACM SIGSOFT Software Engineering Notes*, vol. 25, no. 5, pp. 168–172, 2000.

[22] Z. A. Al-Khanjari and M. R. Woodward, “Investigating the partial relationships between testability and the dynamic range-to-domain ratio,” *Australasian Journal of Information Systems*, vol. 11, 2003.

[23] K. Androutsopoulos, D. Clark, H. Dan, R. M. Hierons, and M. Harman, “An analysis of the relationship between conditional entropy and failed error propagation in software testing,” in *International Conference on Software Engineering*. New York, NY, USA: ACM, 2014, pp. 573–583.

[24] D. Clark and R. M. Hierons, “Squeeziness: An information theoretic measure for avoiding fault masking,” *Information Processing Letters*, vol. 112, no. 8, pp. 335–340, 2012.

[25] M. Harman, Y. Jia, and W. B. Langdon, “A manifesto for higher order mutation testing,” in *International Conference on Software Testing, Verification and Validation Workshops*. IEEE, 2010, pp. 80–89.

[26] R. Gopinath, C. Jensen, and A. Groce, “Mutations: How close are they to real faults?” in *International Symposium on Software Reliability Engineering*, Nov 2014, pp. 189–200.

[27] J. Siegmund, N. Siegmund, and S. Apel, “Views on internal and external validity in empirical software engineering,” in *International Conference on Software Engineering*, 2015.

[28] Y. Jia and M. Harman, “Higher order mutation testing,” *Information and Software Technology*, vol. 51, no. 10, pp. 1379–1393, Oct. 2009.