

Modelling the effects of combining diverse software fault detection techniques

Bev Littlewood, Peter Popov, Lorenzo Strigini

Centre for Software Reliability, City University, London

Nick Shryane

Department of Psychology, City University, London

(To appear in IEEE Transactions on Software Engineering, Vol. 26, No. 12, 2000.

This version corresponds to Version 1.0, 28 May 1999, of the DISPO Technical Report of the same title, BL_DD_D03_v1_0)

Abstract

The software engineering literature contains many studies of the efficacy of fault finding techniques. Few of these, however, consider what happens when several different techniques are used together. We show that the effectiveness of such multi-technique approaches depends upon quite subtle interplay between their individual efficacies and *dependence* between them. The modelling tool we use to study this problem is closely related to earlier work on software *design diversity*. The earliest of these results showed that, under quite plausible assumptions, it would be unreasonable even to expect software versions that were developed 'truly independently' to fail independently of one another. The key idea here was a 'difficulty function' over the input space. Later work extended these ideas to introduce a notion of 'forced' diversity, in which it became possible to obtain system failure behaviour better even than could be expected if the versions failed independently. In this paper we show that many of these results for *design diversity* have counterparts in diverse *fault detection* in a single software version. We define measures of fault finding effectiveness, and of diversity, and show how these might be used to give guidance for the optimal application of different fault finding procedures to a particular program. We show that the effects upon reliability of repeated applications of a particular fault finding procedure are not statistically independent - in fact such an incorrect assumption of independence will always give results that are too optimistic. For *diverse* fault finding procedures, on the other hand, things are different: here it is possible for effectiveness to be even greater than it would be under an assumption of statistical independence. We show that diversity of fault finding procedures is, in a precisely defined way, 'a good thing', and should be applied as widely as possible. The new model and its results are illustrated using some data from an experimental investigation into diverse fault finding on a railway signalling application.

1 Introduction

Diversity is ubiquitous in human activity. In quite mundane contexts it is common to use diversity to improve confidence: for example, I might ask a colleague to check my arithmetic in a complex calculation. The informal idea is that the mistakes he might make will differ from those that I might make, and our arriving at the same answer suggests that neither of us has made a mistake, thus increasing my confidence that the answer is correct.

The key to the approach is, of course, the presence of intellectual differences in the two procedures. Note that this notion of diversity differs fundamentally from that of *redundancy* in which there is simply *replication* (e.g. of a component to increase hardware reliability) of exactly similar items: I would have less trust in my own exact replication of the calculation than in the different calculation of my colleague.

In software, *design diversity* has been proposed as a means of achieving higher reliability than could be achieved (for the same outlay of effort) from a single version. Such design diverse software architectures have seen fairly widespread industrial use [1]. In the early days, it seems that the motivation for the approach owed a great deal to the hardware redundancy metaphor, to the extent that independence of failures of the versions was seen as the goal (albeit recognised as difficult to achieve). Later, several experiments [2], [3] showed that such a goal was probably unrealistic: version failures tended to be highly dependent. Thus, whilst considerable improvement in reliability could be expected from a multi-version architecture compared with the reliabilities of the single versions, this nevertheless fell far short of what would have been achieved if these versions failed independently.

An insight into the nature of the dependence came from a probabilistic model developed by Eckhardt and Lee (EL) [4], and later generalised by Littlewood and Miller (LM) [5]. The basic idea here is that different inputs have different ‘difficulty’ - roughly, the difficulty (and thus proneness to failure) faced by the designer of the software in providing a correct way of processing them. As an example, consider an aircraft flight control system: we might think of the set of inputs corresponding to landing in turbulent wind-shear as ‘more difficult’ than those corresponding to straight and level flight in calm conditions. It is shown in EL that this variation of difficulty induces dependence upon the version failures, even though these might fail *conditionally* independently. The intuition here is simple. If there are two versions, *A* and *B*, and for a particular input we see that *A* has failed, we infer that the input was ‘probably difficult’, and *that B is therefore more likely to fail*. This is true even though, for every input, the versions fail (conditionally) independently - and it is this *conditional* independence that is the difficult goal to which the system designers aspire.¹

¹ In more precise mathematical terms, we say that *A* and *B* fail *conditionally* independently if, for every possible input, *x*,

$$P(A \text{ and } B \text{ both fail} | x) = P(A \text{ fails} | x) \cdot P(B \text{ fails} | x)$$

They fail *unconditionally* independently, on the other hand, if for a randomly chosen (i.e. unknown) input

$$P(A \text{ and } B \text{ both fail}) = P(A \text{ fails}) \cdot P(B \text{ fails}).$$

The point is that, in general, conditional independence does not imply unconditional independence.

The LM model generalises EL by introducing the possibility that the procedures used by the A and B teams may be different ('forced diversity'), and thus that what the A team finds difficult may be different from what the B team finds difficult. It introduces the possibility that diversity can be forced so that the versions will fail in *negatively correlated* ways - thus giving better system reliability even than would be obtained from independence. It has to be admitted, however, that such an outcome may be unlikely in practice.

A major conclusion to be drawn from these models is that there are traps for the unwary in simple reliability models of multi-version software. In particular, assumptions of conditional independence generally do not carry through to justify claims of unconditional independence.

Although the models are quite subtle, their intuitive underpinnings are quite simple. What is surprising is their very wide applicability. In this paper we shall show that similar results apply to diversity in the fault detection processes that are applied to a *single* software version. Once again, the benefits that we get from the application of diverse fault detection procedures may sometimes be less than we could expect under naïve assumptions of independence. However, the possibility of obtaining real benefits from forced diversity, and actually *predicting* the extent of these benefits, seems more plausible here than it does in the case of design diversity.

2 A model of diverse fault detection

Consider a single program that is going to be subjected to two different ('diverse') fault detection procedures, A and B . For simplicity we shall assume that the program is demand-based (e.g. a nuclear reactor protection system). As an example, one of the fault detection procedures might be testing, in which the program is subjected to a particular number of demands; another might be some form of static analysis, for which a certain amount of staff effort is allocated.

The practical intuition here is that each fault detection procedure varies in its efficacy from one fault to another, and that the different procedures may target *different* faults most effectively. That is, procedure A may be stronger on those faults for which B is weaker. This is a kind of forced diversity - applied to fault detection - similar to the forced diversity of design and development methods in the LM model. The difference here is that the diverse fault detection is being applied to a single program. We shall try to keep the notation here as close as possible to that of LM, so that readers familiar with this model can see the similarities.

We shall assume that there is at most one fault associated with each input. One way of thinking of a fault in a program is as the set of all inputs that change from being 'faulty' (i.e. cause a failure when executed) to 'non-faulty' (i.e. do not cause a failure when executed) when the program is changed 'in order to remove a fault' [6].

It is now possible to imagine all possible faults that *might* be in a program. We shall label these with the natural numbers: $\{i: i=1,2,3,\dots\}$. Clearly, some of these faults will be more likely to be present in a particular program than others. Let

$$p_i = P(\text{fault } i \text{ is present in a randomly selected program}) \quad (1)$$

This notion of randomly selected program is exactly the same as that used in EL and LM. There is a set of all programs that *could* be written, and the act of writing

a program is modelled as an act of selection, via some probability distribution over this set of all programs. Thus the single program that we are dealing with can be regarded as having been randomly selected from this set of possible programs.

Of particular interest is the *probability distribution* $\{p_i^*\}$ where

$$p_i^* = \frac{P_i}{\sum_i P_i} = P(\text{randomly selected fault is fault } i) \quad (2)$$

For any particular fault, i , we define $\theta_A(i)$ to be the probability that a (randomly chosen) application of the fault detection procedure A fails to find this fault. The key idea, as in EL, is that this function varies from one fault to another. For a particular fault it can be thought of as the ‘difficulty’ of finding that fault using procedure A , similar to the ‘difficulty’ of inputs in the EL and LM models.

If, as an example, we think of A as being the execution of n operational test cases - i.e. chosen at random using the operational profile over the input space - then clearly each such ‘test’ will either reveal the fault i or it will not. If we were to repeat this procedure many times, generating many sequences, each of n inputs, from the operational profile, $\theta_A(i)$ can be thought of as the proportion in which the procedure fails to detect the fault i .

The difficulty functions determine the efficacy of the fault detection procedures. For example, we can define a measure of A ’s fault-detection efficacy as the chance that an ‘average’ or ‘typical’ fault is found in a (randomly chosen) application of A . In order to keep our notation in step with that of the EL and LM models we shall generally express the results here in terms of *ineffectiveness* (cf. unreliability in the earlier models) of the fault detection procedure A :

$$\begin{aligned} &P(A \text{ fails to detect a randomly chosen fault}) \\ &= \sum_i p_i^* \cdot \theta_A(i) = E_{p^*}(\theta_A(i)) \end{aligned} \quad (3)$$

where the notation E_{p^*} indicates a mean obtained with respect to the probability distribution p^* . We have the following:

Result 1:

$$\begin{aligned} &E(\text{number of faults in program undetected after the application of } A) \\ &= E_{p^*}(\theta_A(i)) \cdot E(\text{number of faults in program before the application of } A) \end{aligned} \quad (4)$$

and

$$\begin{aligned} &E(\text{number of faults detected by application of } A) \\ &= (1 - E_{p^*}(\theta_A(i))) E(\text{number of faults present in program}) \end{aligned} \quad (5)$$

Proof:

$$\begin{aligned}
& E(\text{number of faults undetected in program after the application of } A) \\
&= \sum_i P(\text{fault } i \text{ present and is not detected by application of } A) \\
&= \sum_i p_i \cdot \theta_A(i) = \left(\sum_i p_i \right) \left(\sum_i p_i^* \cdot \theta_A(i) \right) \\
&= E_{p^*}(\theta_A(i)) \cdot E(\text{number of faults in program before the application of } A)
\end{aligned}$$

The second part of the result follows trivially.

QED

We shall define the *effectiveness* of A to be

$$1 - \textit{ineffectiveness} = \left(1 - E_{p^*}(\theta_A(i)) \right)$$

which is simply the probability that A successfully detects a randomly chosen fault.

For a different fault removal procedure, B , we could define similarly $\theta_B(i)$ and $E_{p^*}(\theta_B(i))$. If $E_{p^*}(\theta_A(i))$ were greater than $E_{p^*}(\theta_B(i))$ we would say that A is less effective at finding faults than B , in the sense that it would be expected to detect fewer faults.

We shall now show that, corresponding to the EL result in design diversity, we have here a lack of independence in the effects of successive applications of the same fault detection procedure, and that this implies a *law of diminishing returns*. Consider again, as an example, the situation where A is operational testing. An interesting question is how the effectiveness of this kind of testing changes as we apply more randomly chosen operational inputs. Consider the case where we carry out *two* such fault detection procedures, independently, A_1 and A_2 . Each comprises n independently randomly chosen inputs, and the two sequences of n inputs are independent of one another. In general in such a case of a double application of a procedure we have:

Result 2:

$$\begin{aligned}
& P(A_1 \text{ and } A_2 \text{ fail to detect a randomly chosen fault}) \\
&\geq P(A_1 \text{ fails to detect a randomly chosen fault}) \cdot \\
&P(A_2 \text{ fails to detect a randomly chosen fault})
\end{aligned} \tag{6}$$

or, equivalently,

$$\begin{aligned}
& P(A_2 \text{ fails to detect a randomly chosen fault} \mid A_1 \text{ failed}) \\
&\geq P(A_2 \text{ fails to detect a randomly chosen fault})
\end{aligned} \tag{7}$$

Proof:

Clearly, for each fault i

$$P(A_1 \text{ and } A_2 \text{ fail to detect fault } i) = \theta_A^2(i)$$

i.e. failures of the two applications of the fault detection procedure are conditionally *independent*, for every fault.

For a randomly selected fault, on the other hand

$$\begin{aligned} &P(A_1 \text{ and } A_2 \text{ fail to detect a randomly chosen fault}) \\ &= \sum_i p_i^* \cdot \theta_A^2(i) = E_{p^*}(\theta_A^2(i)) \geq [E_{p^*}(\theta_A(i))]^2 \end{aligned} \quad (8)$$

$$\text{since } E_{p^*}(\theta_A^2(i)) - [E_{p^*}(\theta_A(i))]^2 = \text{Var}(\theta_A(i)) \geq 0$$

QED

It is easy to show (*cf.* the EL and LM models) that there is equality in these expressions *if and only if* $\theta_A(i) = \theta$ identically for all faults. Clearly, it seems certain that no real fault finding procedures have the property of completely constant difficulty with respect to all faults.

Another way to think of these expressions is in terms of the number of faults you would expect to detect in a program by applying a particular fault finding procedure. Clearly, if you apply the procedure twice you would not expect to find twice as many faults as applying it just once - informally, there is a chance that some of the faults that could have been found in the second application would already have been found in the first application. This would be true even if there were complete independence in the two applications (i.e., in the terminology of the model, $\theta_A(i) = \theta$ identically for all faults). What the result above says is that things are even worse than this independence case when there is variation of difficulty: i.e. you would expect to find even fewer faults with two applications than you would if you could assume independence.

These results depend on the fact that, even though the two applications of the fault detection procedure are conditionally independent for all faults, they are not unconditionally independent. Informally, the failure of the first application of A to find the fault suggests that it is a ‘difficult’ fault for A , and thus that a second application of A will also be likely to fail to find it. If the first application of operational testing has revealed only a few faults, for example, we should tend to lose confidence in the likely effectiveness of a second application of the procedure.

This result corresponds to our intuition. Most people would not persevere with putting all their fault detection effort into a single procedure, such as, for example, inspection or operational testing. Rather they would tend to expect there to be a law of diminishing returns operating, whereby most of the faults that *can* easily be detected by a procedure, *are* eventually detected. At some point, therefore, it would seem sensible to cease one fault detection activity and switch to another, which it is hoped will target a different class of faults. We now examine this case of *diverse* fault removal: the reader familiar with the earlier models of design diversity will see that the following is similar to the LM model, as the former was similar to EL.

Consider now, therefore, two different fault detection procedures A and B :

Result 3:

$$\begin{aligned} &P(A \text{ and } B \text{ fail to detect randomly selected fault}) \\ &> P(A \text{ fails})P(B \text{ fails}) \end{aligned} \quad (9)$$

and

$$\begin{aligned} &P(B \text{ fails to detect a randomly chosen fault} \mid A \text{ failed}) \\ &> P(B \text{ fails to detect a randomly chosen fault}) \end{aligned} \quad (10)$$

if and only if

$$Cov_{p^*}(\theta_A(i), \theta_B(i)) > 0 \quad (11)$$

Proof:

We assume conditional independence in the following obvious way:

$$P(A \text{ and } B \text{ fail to detect fault } i \mid i \text{ present}) = \theta_A(i) \cdot \theta_B(i) \quad (12)$$

for every fault i .

For a randomly selected fault, on the other hand:

$$\begin{aligned} &P(A \text{ and } B \text{ fail to detect randomly selected fault}) \\ &= \sum_i P_i^* \cdot \theta_A(i) \cdot \theta_B(i) = E_{p^*}(\theta_A(i) \cdot \theta_B(i)) \end{aligned} \quad (13)$$

which, in general

$$> E_{p^*}(\theta_A(i)) \cdot E_{p^*}(\theta_B(i)) = P(A \text{ fails to detect randomly selected fault}).$$

$$P(B \text{ fails to detect randomly selected fault})$$

$$\text{if and only if } Cov_{p^*}(\theta_A(i), \theta_B(i)) = E_{p^*}(\theta_A(i) \cdot \theta_B(i)) - E_{p^*}(\theta_A(i)) \cdot E_{p^*}(\theta_B(i)) > 0$$

QED

In other words, you would expect to find fewer faults by applying A and B when $Cov_{p^*}(\theta_A(i), \theta_B(i)) > 0$ than you would if you could assume independence (and, of course, this latter expected number will be smaller² than the sum of the numbers you would expect from a single application of A and a single application of B).

There is an intriguing possibility of *better than independent* behaviour of the two procedures if the covariance is negative. Such negative covariance would be possible whenever the different fault detection procedures targeted different types of

² Strictly 'less than or equal to'. There can be equality here, apart from the case of constant difficulty for either A or B , if there is a kind of 'complete disjointness' in the difficulty functions of the two procedures, i.e. if for every fault i one of the difficulty functions takes the value 1 - for every fault one of the procedures is completely ineffective. Such a case does not seem to have any practical relevance.

faults, i.e. whenever the ‘difficult’ faults for one procedure are the ‘easy’ ones for the other, and vice versa.

An expectation of something like negative covariance does seem to lie behind the intuition of those people involved in software verification and validation, for whom an eclectic mix of different fault finding procedures is preferred to the extensive application of a single one. What is novel in the work reported here is that we introduce the possibility of *measures* that characterise this desirable diversity between different procedures. Interestingly, the measures seem more amenable to statistical estimation than the corresponding ones in design diversity, as we shall see in a preliminary example in Section 6.

3 Effects on reliability

The results of the previous section concern the efficacy of different procedures at finding faults. Of course, knowing that a procedure is good at finding faults is not the same as knowing that it is good at improving *reliability* (when the faults that are detected are removed). It is well-known that in real programs different faults can have very different impacts on a program’s unreliability [7]: a fault-finding procedure that was very efficient at finding ‘small’ faults could be very inefficient at finding ‘large’ ones, and thus at improving reliability. A ‘good’ fault finding (and removal) procedure is one that tends to improve reliability most efficiently [6]. We now show that similar results to those above also apply to the efficacy of fault finding procedures in improving reliability. Throughout this section we shall assume that when a fault is detected it is removed with certainty.

Let

$$\pi_i = P(\text{fault } i \text{ is activated by randomly selected input and is not detected and fixed}) \quad (14)$$

The unreliability of the program is then

$$P(\text{program fails on randomly selected input}) = \sum_i \pi_i \quad (15)$$

and

$$\begin{aligned} \pi_i^* &= \frac{\pi_i}{\sum_i \pi_i} \\ &= P(\text{randomly selected failure caused by fault } i) \end{aligned} \quad (16)$$

Consider now the effect of applying the fault removal procedure A on the program unreliability, i.e. on the probability that it fails on a randomly selected input.

Result 4:

$$\begin{aligned} &P(\text{program fails on randomly selected input following application of } A) \\ &= E_{\pi^*}(\theta_A(i)) P(\text{program fails on randomly selected input before applying } A) \end{aligned} \quad (17)$$

Proof:

$$\begin{aligned}
 & P(\text{program fails on randomly selected input following application of } A) \\
 &= \sum_i P(\text{fault } i \text{ activated following application of } A) \\
 &= \sum_i P(A \text{ has not removed fault } i \text{ and it is activated}) \\
 &= \sum_i \pi_i \cdot \theta_A(i) = \left(\sum_i \pi_i \right) \left(\sum_i \pi_i^* \cdot \theta_A(i) \right) \\
 &= E_{\pi^*}(\theta_A(i)) \cdot P(\text{program fails on randomly selected input before applying } A)
 \end{aligned}$$

QED

Thus the factor $E_{\pi^*}(\theta_A(i))$ can be seen as a measure of the *ineffectiveness* of procedure A in improving reliability: the smaller this is, the better. Similarly, its complement can be thought of as the *effectiveness*.

Consider now the application of fault removal procedure A twice, A_1, A_2 . We have a result similar to Result 2:

Result 5:

The ineffectiveness of A_1 and A_2 together is greater than or equal to the product of the ineffectiveness of A_1 and the ineffectiveness of A_2 .

Proof:

$$\begin{aligned}
 & P(\text{program fails on randomly selected input following application of } A_1 \text{ and } A_2) \\
 &= \sum_i P(\text{fault } i \text{ activated following application of } A_1 \text{ and } A_2) \\
 &= \sum_i P(A_1 \text{ and } A_2 \text{ have not removed fault } i \text{ and it is activated}) \\
 &= \sum_i \pi_i \cdot \theta_{A_1}^2(i) = \sum_i \pi_i \sum_i \pi_i^* \cdot \theta_{A_1}^2(i) \\
 &= E_{\pi^*}(\theta_{A_1}^2(i)) \cdot P(\text{program fails on randomly selected input before applying } A_1 \text{ and } A_2) \\
 &\geq \left(E_{\pi^*}(\theta_{A_1}^2(i)) \right)^2 \cdot P(\text{program fails on randomly selected input before applying } A_1 \text{ and } A_2)
 \end{aligned}$$

QED

As before, there will be equality here if and only if $\theta_A(i) \equiv \theta$ for some constant θ , identically for all faults.

The effectiveness of applying A_1 and A_2 together in reducing the probability of failure of the program on a randomly selected input is always less than the result we

would expect under independence, since $E_{\pi^*}(\theta_A^2(i))$ is always greater than $(E_{\pi^*}(\theta_A(i)))^2$. This is, again, the analogue of the EL result.

If, on the other hand, we apply *diverse* fault removal procedures, A and B , we find, as before, that we can do better than this:

Result 6:

The ineffectiveness of applying A and B together is greater than the product of the ineffectiveness of A and the ineffectiveness of B if and only if $Cov_{\pi^*}(\theta_A(i), \theta_B(i)) > 0$.

Proof:

$$\begin{aligned}
& P(\text{program fails on randomly selected input following application of } A \text{ and } B) \\
&= \sum_i P(\text{fault } i \text{ activated following application of } A \text{ and } B) \\
&= \sum_i P(A \text{ and } B \text{ have not removed fault } i \text{ and it is activated}) \\
&= \sum_i \pi_i \cdot \theta_A(i) \cdot \theta_B(i) = \left(\sum_i \pi_i \right) \left(\sum_i \pi_i^* \cdot \theta_A(i) \cdot \theta_B(i) \right) \\
&= E_{\pi^*}(\theta_A(i) \cdot \theta_B(i)) \cdot P(\text{program fails on randomly selected input before applying } A \text{ and } B) \\
&> E_{\pi^*}(\theta_A(i)) \cdot E_{\pi^*}(\theta_B(i)) \cdot P(\text{program fails on randomly selected input before applying } A \text{ and } B) \\
&\text{if and only if } Cov_{\pi^*}(\theta_A(i), \theta_B(i)) = E_{\pi^*}(\theta_A(i) \cdot \theta_B(i)) - E_{\pi^*}(\theta_A(i)) \cdot E_{\pi^*}(\theta_B(i)) > 0
\end{aligned}$$

QED

Thus the factor $E_{\pi^*}(\theta_A(i) \cdot \theta_B(i))$, which determines the reduction in unreliability (improvement in reliability) coming from the successive applications of A and B , can be either smaller or larger than the equivalent expression that assumes independence, $E_{\pi^*}(\theta_A(i)) \cdot E_{\pi^*}(\theta_B(i))$, according to whether the covariance $Cov_{\pi^*}(\theta_A(i), \theta_B(i))$ is positive or negative. We would like it to be smaller, i.e. the covariance to be negative.

4 Optimal allocation of fault detection procedures

The results of the previous two sections show that, in a limited sense, negative correlation between the difficulty functions of two fault finding procedures is ‘a good thing’. In practice, of course, this does not tell us how to deploy A and B when we have a fixed amount of effort available. There will be a trade-off between the efficacies of the individual fault finding procedures and their dependence, and it may be that it will be most effective to use only A (or only B). In this section we provide some tentative advice for optimal allocation of procedures, by showing that, *when we believe that there is nothing to choose between the different procedures in terms of their individual efficacies*, their most diverse application is always best.

Consider the case where we have two fault finding procedures, A and B . For example, let A , as before, be operational testing; let B be a form of static analysis. Assume that we are *indifferent* between two randomly chosen applications of A (i.e. two randomly chosen sets of operational inputs), and two randomly chosen applications of B (i.e. two particular applications of the static analysis procedure). That is, if we were to apply only one of these procedures to our program we would have no preference between them. Then we can show that it is always better to apply A once and B once rather than either of them twice; more precisely:

Result 7:

If $E_{\pi^*}(\theta_A^2(i)) = E_{\pi^*}(\theta_B^2(i))$ then

$P(\text{program fails on randomly selected input following application of } A \text{ and } B)$
 $< P(\text{program fails on randomly selected input following application of } A_1 \text{ and } A_2)$

and

$P(\text{program fails on randomly selected input following application of } A \text{ and } B)$
 $< P(\text{program fails on randomly selected input following application of } B_1 \text{ and } B_2)$

Proof:

By the Cauchy-Schwarz inequality

$$E_{\pi^*}(\theta_A(i) \cdot \theta_B(i)) < \left(E_{\pi^*}(\theta_A^2(i))\right)^{1/2} \left(E_{\pi^*}(\theta_B^2(i))\right)^{1/2} = E_{\pi^*}(\theta_A^2(i)) = E_{\pi^*}(\theta_B^2(i))$$

so

$P(\text{program fails on randomly selected input following application of } A \text{ and } B)$
 $= E_{\pi^*}(\theta_A(i) \cdot \theta_B(i)) P(\text{program fails on randomly selected input before fault finding})$
 $< E_{\pi^*}(\theta_A^2(i)) P(\text{program fails on randomly selected input before fault finding})$
 $= P(\text{program fails on randomly selected input following application of } A_1 \text{ and } A_2)$

and similarly for B_1 and B_2 .

QED

A similar result holds for efficacy defined in terms of the expected numbers of faults removed: we would expect more faults to be removed by an application of A and B than by two applications of A or two of B (subject to a similar indifference assumption³). The similarity of this result to that just proved is just a generalisation of the parallelism that we have seen between the results involving efficacy of fault removal (results 1-3) and those involving improvement in reliability (results 4-6). The

³ Note, however, that indifference with respect to fault finding ability does not generally imply indifference with respect to reliability improvement, nor *vice versa*.

proof follows simply and will not be given here: similar comments apply to all the results of this section.

This result can be generalised to more than two fault finding procedures. Thus for three procedures A , B and C , we have a choice between the following applications (in an obvious notation): AAA , BBB , CCC , AAB , AAC , ABB , . . . , BCC , ABC . If we are indifferent between all the single fault finding procedures

$$E_{\pi^*}(\theta_A^3(i)) = E_{\pi^*}(\theta_B^3(i)) = E_{\pi^*}(\theta_C^3(i)) \quad (18)$$

and between all those using two procedures

$$E_{\pi^*}(\theta_A^2(i) \cdot \theta_B(i)) = E_{\pi^*}(\theta_A^2(i) \cdot \theta_C(i)) = E_{\pi^*}(\theta_A(i) \cdot \theta_B^2(i)) \dots \quad (19)$$

it can be shown that $E_{\pi^*}(\theta_A(i) \cdot \theta_B(i) \cdot \theta_C(i))$ is smaller than all these other inefficiency factors, and so the greatest improvement in reliability will come by applying ABC .

More generally, when k different fault finding procedures are available, subject to certain indifference assumptions between them, it is best to use all of them and to spread their use as evenly as possible. Thus, for example, if we are prepared to apply 5 randomly chosen fault finding procedures, but only have 3 different types, then $AABCC$ is better than $AAABC$, etc. This can be expressed in general as:

Result 8:

Let (n_1, n_2, \dots, n_k) represent the allocation of n_i randomly chosen applications of fault finding procedure i ($i=1, 2, \dots, k$),

$$\sum_{i=1}^k n_i = n.$$

Assuming indifference between allocations that merely involve permutations of procedures, the best allocations (there will generally be more than one) are those for which

$$\lfloor n/k \rfloor \leq n_i \leq \lfloor n/k \rfloor + 1, \quad 1 \leq i \leq k \quad (20)$$

where $\lfloor x \rfloor$ denotes the greatest integer not greater than x .

Proof:

See [5].

The importance of this result lies in its potential for optimally allocating a given amount of effort among several fault finding procedures. The key to its practical application lies in the notion of ‘indifference’: we need to define a unit of fault finding effort which is meaningful for all the different types and which is such that there is indifference in the terms required by the result. The simplest way that this could be done would be for indifference between a single application of A and B to imply indifference between double applications, between triple applications, etc. In such a case, defining the initial ‘unit’ of each kind of fault finding effort is sufficient to

ensure that the conditions of the result apply, and the best allocation(s) are those that ‘spread effort as equally as possible’ among different procedures.

If A were, as before, operational testing, and B were a type of static analysis, we would need to select a number of test cases for the ‘unit’ of operational testing, and an ‘amount’ of static analysis (e.g. the time for an analyst of a particular competence) so that we would be indifferent between (randomly chosen) single applications of each of the two⁴. It seems likely that in practice there is usually some appropriate ‘cost’ variable that will allow us to define units of different fault finding procedures that we think are equivalent in their likely impact upon reliability. This result can be seen as a formalisation of the kind of informal allocation between different procedures that is carried out on a day-to-day basis during real system verification and validation.

5 Degrees of diversity

It seems intuitively plausible that, if diversity is ‘a good thing’, then *more* diversity is better than *less*. In this section we present a measure of diversity between pairs of fault finding procedures that gives meaning to the notion of ‘more diverse’.

Consider the two procedures A and B . Since a procedure is completely characterised by its difficulty function over faults, θ , we can define a distance between the two procedures in this function space:

$$\|\theta_A - \theta_B\|^2 = E_{\pi^*}((\theta_A(i) - \theta_B(i))^2) \quad (21)$$

Clearly, $\|\theta_A - \theta_B\|^2 > 0$ unless $\theta_A(i) \equiv \theta_B(i)$. It follows trivially, if we assume indifference between the procedures when applied singly, that

$$E_{\pi^*}(\theta_A(i) \cdot \theta_B(i)) < E_{\pi^*}(\theta_A^2(i)) = E_{\pi^*}(\theta_B^2(i))$$

This is precisely Result 7, which says that the improvement in reliability by applying A and B is better than that obtained by applying either twice.

Consider now the case where three procedures are available, A , B and C , but it is only feasible (e.g. on cost grounds) to apply two of them. Suppose further that we believe that A and B are more diverse, in the sense of (21), than are A and C , or B and C . This is shown schematically in Figure 1:

⁴ Notice that indifference does not mean that we believe the effects will be the same. On the contrary, we know that the effects of the testing and analysis will be different in their fault finding - they will target different types of faults differently. The point is that we do not prefer one to another in their ability to *improve reliability*.

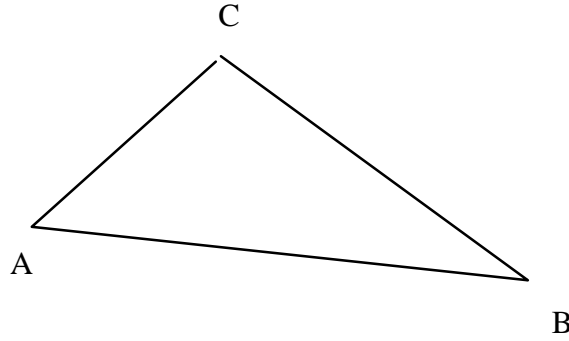


Figure 1

Result 9:

If A and B are more diverse than B and C , i.e.

$$\|\theta_A - \theta_B\| > \|\theta_B - \theta_C\|,$$

and we are indifferent between two-fold applications of each of the three procedures, i.e.

$$E_{\pi^*}(\theta_A^2(i)) = E_{\pi^*}(\theta_B^2(i)) = E_{\pi^*}(\theta_C^2(i)),$$

then

$$E_{\pi^*}(\theta_A(i), \theta_B(i)) < E_{\pi^*}(\theta_B(i), \theta_C(i))$$

[The proof is trivial]

In other words, if we have a choice of two procedures that are more diverse than another pair, we should prefer the former, all things being equal.

6 Example

Data to illustrate parts of the new model were obtained from a study into the verification of safety-critical computer programs used in railway signalling systems: this experiment pre-dated the development of the model described here.

Observational fieldwork conducted on-site at a number of railway signalling contractors had revealed qualitative differences between faults typically found by diverse verification methods, in this case static code checking and functional testing. However, variability in the complexity of work and personnel expertise found at different contractors' sites, and most importantly the lack of independence between the fault-finding methods as practised *in situ*, could have been confounding this result.

Computer-aided simulations of the checking and testing tasks were therefore developed to allow controlled laboratory experiments to be conducted. Eighty-eight

university computer science students were recruited for these experiments. After a period of ‘core’ instruction in railway signalling principles common to all participants, half were then given specific training for the code checking task and half were trained for the testing task.

Participants were then required to verify two simulated railway signalling programs, either by code checking or functional testing as appropriate. Each program was seeded with eight faults (taken from those found during the fieldwork), each of the faults being potentially detectable with either verification method. The dependent variable was thus the proportion of the total faults detected by each participant, the independent variable being whether the participant was a checker or a tester. For reasons that are beyond the scope of the present paper, some participants were excluded from the experiment, leaving a total of twenty-seven checkers and thirty-six testers (Further information on this issue, and on the experimental task in general, can be found in [8], [9]; for more information on the railway signalling task on which it is based, see [10]).

The raw data are shown in Table 1. Here the entries in the table are, for each fault, the proportion of the individuals in that part of the experiment who were unsuccessful in finding the fault: for fault i and procedure A we take this to be an estimate of $\theta_A(i)$ in the notation of the model. If we treat each program separately, we have $p_i^*=1/8$ for each of the eight faults in each program. If we let A represent checking and B represent testing, it is then easy to show that for Program 1 (estimates of) the relevant parameters of the model are:

Program 1			Program 2		
Fault id	Proportion in checking	Proportion in testing	Fault id	Proportion in checking	Proportion in testing
F11	.7778	.4168	F21	.2222	.0833
F12	.0000	.1389	F22	.8148	.2778
F13	.2593	.5556	F23	.5926	.5000
F14	.4815	.4722	F24	.1481	.9444
F15	.7778	.1944	F25	.4444	.2778
F16	.3704	.7222	F26	.2963	.7778
F17	.1852	.3611	F27	.2222	.8056
F18	.4444	.9167	F28	.7778	.2778

Table 1 Proportions of testers and checkers unsuccessful in finding each of eight seeded faults in two programs.

$$\begin{aligned}
E_{p^*}(\theta_A(i)) &= 0.412, & E_{p^*}(\theta_B(i)) &= 0.472 \\
E_{p^*}(\theta_A^2(i)) &= 0.235, & E_{p^*}(\theta_B^2(i)) &= 0.282 \\
E_{p^*}(\theta_A(i) \cdot \theta_B(i)) &= 0.199, & E_{p^*}(\theta_A(i))E_{p^*}(\theta_B(i)) &= 0.194 \\
Cov_{p^*}(\theta_A(i), \theta_B(i)) &= 0.0040
\end{aligned} \tag{22}$$

and for Program 2:

$$\begin{aligned}
E_{p^*}(\theta_A(i)) &= 0.440, & E_{p^*}(\theta_B(i)) &= 0.493 \\
E_{p^*}(\theta_A^2(i)) &= 0.253, & E_{p^*}(\theta_B^2(i)) &= 0.329 \\
E_{p^*}(\theta_A(i) \cdot \theta_B(i)) &= 0.179, & E_{p^*}(\theta_A(i))E_{p^*}(\theta_B(i)) &= 0.217 \\
Cov_{p^*}(\theta_A(i), \theta_B(i)) &= -0.0381
\end{aligned} \tag{23}$$

Notice that the efficacies of A and B are roughly similar between the two programs, but that B seems to be consistently worse than A . Also, there seems to be ‘more diversity’ between A and B in their application to Program 2 than to Program 1: there is negative correlation between the difficulty functions in the case of Program 2, but small positive correlation for Program 1.

After two applications of A to Program 2, we would expect 25.3%⁵ of faults to be undetected; after two applications of B , 32.9% of faults to be undetected; but after an application of each of A and B , we would expect only 17.9% of faults to be undetected. Thus, in this case we would expect the application of A and B to be more effective than a double application of A or of B . This is so even though we are not indifferent between A and B here: in fact A seems significantly better than B (and AA than BB). Here the naïve ‘independence’ argument would seriously underestimate the efficacy of AB , suggesting that it would leave 21.7% of faults undetected, rather than the correct figure of 17.9%.

For Program 1, the corresponding figures are: after two applications of A , we expect 23.5% of the faults to be undetected; after two applications of B , 28.2%; and after one application of A and one of B , 19.9%. Once again, the diverse application of AB is better than either AA or BB , even though the difficulty functions show slight positive correlation - but this superiority is less dramatic than in Program 2, where the procedures have negatively correlated difficulty functions. In this case the naïve ‘independence’ assumption would slightly *overestimate* the efficacy of AB , suggesting it would leave 19.4% of faults undetected rather than the correct figure of 19.9%.

The differences in the examples above, (22) and (23), suggest how we might use this model in practice. If we believe that the effectiveness of successive applications of particular diverse fault finding procedures will vary from one class of programs to another (programs developed by different processes, and/or for different application problems, e.g. real-time versus non-real-time), but are constant within

⁵ A word of warning about the number of significant figures here and in later expressions. Essentially we are computing finite-sample estimates of parameters, which can be regarded as exact, and then truncating the decimal expressions for convenience. If we regard these as estimates of infinite populations, of course, the number of significant figures here may be too optimistic. This observation does not affect the general reasoning of the paper.

each class, then we should strive to estimate the parameters of the model for each class of programs. Thus if we had two sets of data, like those for Programs 1 and 2, corresponding to two different types of program, we could estimate the parameters relating to their different behaviours.

If, on the other hand, we have no reason to believe that there are such differences, we should regard data from all programs as being evidence to be used for obtaining a single set of parameter estimates for the model. If we aggregate the above data in this way, treating each program as having been sampled with constant probability from a population of programs, we find that the model parameters are estimated as follows:

$$\begin{aligned}
E_{p^*}(\theta_A(i)) &= 0.426, & E_{p^*}(\theta_B(i)) &= 0.483 \\
E_{p^*}(\theta_A^2(i)) &= 0.244, & E_{p^*}(\theta_B^2(i)) &= 0.306 \\
E_{p^*}(\theta_A(i)\theta_B(i)) &= 0.189, & E_{p^*}(\theta_A(i))E_{p^*}(\theta_B(i)) &= 0.206 \\
Cov_{p^*}(\theta_A(i), \theta_B(i)) &= -0.0168
\end{aligned} \tag{24}$$

Now, after a double application of A we would expect 24.4% of faults to be undetected; after a double application of B , 30.6%; and after application of a single A and a single B , 18.9%. Once again, rather informally, the strong diversity of the procedures, indicated by their negative correlation, means that an application of each of A and B is best, even though A is clearly superior to B (and AA to BB). Once again, because of the negative correlation of the difficulty functions, the naïve independence estimate underestimates the efficacy of AB : it suggests that this will leave behind 20.6% of faults undetected whereas the true figure is 18.9%.

These results illustrate the quite subtle interplay that there can be between the efficacies of the individual procedures and their ‘degree of diverseness’: in general we need to know all first and second moments of the (random variable) difficulty functions before we can determine the most effective mix of procedures. We have been able to show earlier that by imposing certain indifference constraints it is possible to know that AB will be better than AA or BB whenever there is negative correlation. When we cannot make such assumptions, however, mere diverseness, as represented by negative correlation between the difficulty functions, is not sufficient to ensure that it will be best to apply both procedures. It is easy to show that when there is negative correlation AB will always be better than the *worst* of AA and BB , but it could be either better or worse than the *better* of these. Equally, negative correlation is not *necessary* for AB to be superior to AA and BB : there can be benefit in diversity even when the procedures are positively correlated, as in the case of Program 1.

These comments notwithstanding, the results of Section 3 (and similarly those of Section 4) do seem like a promising way of giving designers advice about the best fault finding approach when, as is sometimes the case, we are in a position to balance the efficacies of different procedures by applying one of them ‘more’ or ‘less’ extensively. Thus, in the example here we might require the testers to spend more time - examining more test cases - so as to make $E_{p^*}(\theta_B(i)) = E_{p^*}(\theta_A(i))$ in (24). Whenever we can do this, we shall have a *guarantee* that AB will be superior to AA or BB , even when we do not have the detailed extra information about the parameters of the model that we have from this controlled experiment.

So far, the model described here has been predictive - e.g. predicting the best fault finding combination of procedures for a novel program - only if we can assume that the program about which we wish to make a prediction is ‘similar’ to the ones used for the estimation of the model parameters. Clearly it would be desirable to lift this restriction, and in some cases this may be possible. The idea here is to deal with *classes* of faults, rather than with the faults themselves, and it can be illustrated again by the railway data example.

The psychologists who devised this experiment were interested in investigating differences in diverse fault finding between different types of fault; details can be found in the papers cited earlier, but the following is a brief account. As part of the signalling system design task, engineers must translate information contained in track layout plans, regarding permitted directions of movement of trains over track sections, into a machine-readable label. Configurations of track sections are variable, and so a rule is used to specify the mapping between the label and the route of the train. Engineers made mistakes when generating the labels, and more importantly when performing an independent check of others’ work, a vital part of the rigorous verification and validation process to which safety critical systems must be subjected. Faults in Programs 1 and 2 varied as to whether the code involved the use of such train subroutes (S) or not (N). There were thus two classes of fault, S and N: in Table 1, faults F11, F15, F22, F25 were of type S and the remaining 12 faults were of type N.

We can now easily compute a table similar to Table 1, but involving *classes of faults* rather than the individual faults themselves:

Fault class id	Probability in checking (A)	Probability in testing (B)
S	.704	.292
N	.333	.546

Table 2 Estimates of the probabilities that a randomly chosen fault of type S (N) will fail to be detected by an application of procedure A (B), obtained from the data of Table 1 aggregated over the two programs.

In an obvious notation, the entries of Table 2 can be thought of as estimates of $\theta_A(s)$, etc. We can now perform calculations similar to those carried out earlier, using $p_S^* = 0.25$, $p_N^* = 0.75$, to obtain:

$$\begin{aligned}
E_{p^*_{classes}}(\theta_A(i)) &= 0.426, & E_{p^*_{classes}}(\theta_B(i)) &= 0.483 \\
E_{p^*_{classes}}(\theta_A^2(i)) &= 0.207, & E_{p^*_{classes}}(\theta_B^2(i)) &= 0.245 \\
E_{p^*_{classes}}(\theta_A(i) \cdot \theta_B(i)) &= 0.188, & E_{p^*_{classes}}(\theta_A(i))E_{p^*_{classes}}(\theta_B(i)) &= 0.206 \\
Cov_{p^*_{classes}}(\theta_A(i), \theta_B(i)) &= -0.0178
\end{aligned} \tag{25}$$

Clearly, the estimates of the efficacy of multiple procedures obtained from this class-based treatment will generally be in error compared with the ‘correct’ results

obtained in (24) from the full fault-based data. It is therefore interesting to compare the results of (24) and (25). The first moments, representing the single application efficacies, are completely identical, as is to be expected. The ineffectiveness estimates for AA and BB , on the other hand, are significantly underestimated in (25) in comparison with (24). The reason, of course, is that values in (25) ignore the variation in the difficulty functions of A and B *within* each class of faults (see the appropriate entries in Table 1): essentially they assume that all faults within a class have the same value of the difficulty function for a particular fault finding procedure (in Table 2 such a value has been computed as the average, over the different faults in the class, of the *actual* difficulty functions from Table 1). Thus, for example (in an obvious notation), the ineffectiveness of AA based on the full fault data can be expressed as:

$$\begin{aligned}
E_{p^*}(\theta_A^2(i)) &= p_S^* \cdot E_{i \in S}(\theta_A^2(i)) + p_N^* \cdot E_{i \in N}(\theta_A^2(i)) \\
&= p_S^* \cdot \left(\left(E_{i \in S}(\theta_A(i)) \right)^2 + \text{Var}_{i \in S}(\theta_A(i)) \right) + p_N^* \cdot \left(\left(E_{i \in N}(\theta_A(i)) \right)^2 + \text{Var}_{i \in N}(\theta_A(i)) \right) \\
&= E_{\text{between classes}} \left(E_{\text{within class}}(\theta_A(i)) \right)^2 + E_{\text{between classes}} \left(\text{Var}_{\text{within class}}(\theta_A(i)) \right) \quad (26)
\end{aligned}$$

and similarly for B .

That is, the first term - which we have called $E_{p^* \text{ classes}}(\theta_A^2(i))$ in the condensed notation of (25), and which takes the value 0.207 - underestimates the true value by an amount $E_{\text{between classes}}(\text{Var}_{\text{within class}}(\theta_A(i))) = 0.037$. This agrees with (24), since $0.244 = 0.207 + 0.037$.

What is interesting and surprising, though, is the closeness between the ineffectiveness estimates of AB in (24) and (25). We would expect these to differ since the expression in (24) can be expressed:

$$\begin{aligned}
E_{p^*}(\theta_A(i) \cdot \theta_B(i)) &= p_S^* \cdot E_{i \in S}(\theta_A(i) \cdot \theta_B(i)) + p_N^* \cdot E_{i \in N}(\theta_A(i) \cdot \theta_B(i)) \\
&= p_S^* \cdot \left(E_{i \in S}(\theta_A(i)) E_{i \in S}(\theta_B(i)) + \text{Cov}_{i \in S}(\theta_A(i), \theta_B(i)) \right) \\
&\quad + p_N^* \cdot \left(E_{i \in N}(\theta_A(i)) E_{i \in N}(\theta_B(i)) + \text{Cov}_{i \in N}(\theta_A(i), \theta_B(i)) \right) \\
&= E_{\text{between classes}} \left(E_{\text{within class}}(\theta_A(i)) \cdot E_{\text{within class}}(\theta_B(i)) \right) \\
&\quad + E_{\text{between classes}} \left(\text{Cov}_{\text{within class}}(\theta_A(i), \theta_B(i)) \right) \quad (27)
\end{aligned}$$

The first term here is the approximation using aggregated fault class data from (25), which in the condensed notation used there is $E_{p^* \text{ classes}}(\theta_A(i) \cdot \theta_B(i)) = 0.188$. The second term is the weighted average, using $p_S^* = 0.25$ and $p_N^* = 0.75$ respectively, of the covariances of the A and B difficulty functions for the different fault classes, which take the values 0.000 and 0.001 respectively (to 3 decimal places). The smallness of these is the reason for the closeness of the estimate of AB effectiveness based on the aggregated fault class data to the correct figure based on the full fault data.

The surprise here is that the devisors of the experiment have managed to arrive at fault classes, S and N , for each of which the difficulty functions of A and B are

almost independent. It would be worth investigating whether this can be done in other circumstances: if it can, we shall be able to obtain accurate estimates of AB ineffectiveness even from aggregated fault class data.

The practical advantage of the treatment via classes of faults, rather than the individual faults themselves, is as follows. Individual faults are likely to be ‘sparse’. If we collect our data from only a few real programs, we are likely to see each fault in no more than one program, in contrast to the artificially seeded programs which are used repeatedly in the experiment. There will thus be no opportunity to estimate the probabilities p_i^* . If, on the other hand, we can identify meaningful and interesting fault classes the associated probabilities may be estimable. Most importantly, we would be able to predict the efficacy of diverse fault finding, as in (25), for *an entirely novel program* if we can estimate the $p^*_{classes}$ values for this new program.

An especially desirable situation would be achieved if the fault classes identified had the special property that was fortuitously obtained in this experiment, i.e., that the difficulty functions for different fault detection methods have zero covariance over the different fault classes. Then, estimating the *mean* of the θ functions for each {fault class, fault detection method} combination - which is much simpler than estimating complete distributions - would be sufficient to allow useful predictions in many cases. As can be seen from (26), the predictions would be exact for combinations of diverse methods, and optimistic for repeated applications of one method: when the methods are ‘diverse’ enough, these estimates would provide sufficient conditions for a project manager to decide in favour of combining two diverse methods.

Essentially this approach reuses the hard-won data such as that of Table 2, gained from past experience of other projects or from experiments, by applying it to prediction in a new context. Whilst this does require that we know, or are prepared to estimate, the new $p^*_{classes}$ values, this is a feasible task in contrast to attempting to do so for the set of all faults. As an example, if we believe that for the new program $p_S^* = 0.50$, $p_N^* = 0.50$, we obtain:

$$\begin{aligned}
E_{p^*_{classes}}(\theta_A(i)) &= 0.519, & E_{p^*_{classes}}(\theta_B(i)) &= 0.419 \\
E_{p^*_{classes}}(\theta_A^2(i)) &= 0.303, & E_{p^*_{classes}}(\theta_B^2(i)) &= 0.192 \\
E_{p^*_{classes}}(\theta_A(i) \cdot \theta_B(i)) &= 0.194, & E_{p^*_{classes}}(\theta_A(i))E_{p^*_{classes}}(\theta_B(i)) &= 0.217 \\
Cov_{p^*_{classes}}(\theta_A(i), \theta_B(i)) &= -0.0235
\end{aligned} \tag{28}$$

For a new program with these $p^*_{classes}$, B is superior to A , and BB to AA . More interestingly, BB is now slightly better than AB , even though there is negative correlation between the difficulty functions.

7 Discussion and conclusions

Most work in the software engineering literature on the efficacy of fault finding procedures has concentrated upon assessing and comparing their individual efficacies. Whilst this is important, the reality is that in practice several of these techniques will be employed together. There are some well-known intuitions about how such combinations of procedures will work: we all know, for example, that it is

best to use fault finding procedures that are effective in some general way; but we equally know that any single such procedure may miss a whole class of faults even when applied most extensively. Many experimental comparisons for alternative fault-finding methods have been reported, e.g. [11], [12], [13], [14], [15]. Those experimenters that looked at the effectiveness of the methods on individual faults (e.g. [15]) corroborated the intuitive belief that different methods were most effective at discovering different faults. Thus, even when we know that procedure *A* is better at fault finding than *B*, we would be wary of using *only A* because it may have little chance of finding certain faults that *B* may find quite easily. The work described here is an attempt to formalise intuitions of this kind. The ultimate aim is to provide advice to practitioners on the best way of combining different approaches to fault finding on a single program.

An example of these arguments about fault finding efficacy has long raged in the testing community between advocates of operational testing, and those of other testing practices [6]. It is known that operational testing has the useful property that its chances of finding faults (for a given outlay of effort) are in direct proportion to the impact of these faults on the unreliability of the software - the greater the occurrence rate of a fault, the greater its chance of being discovered in an operational test. However, this is not the same as saying that operational testing is more efficient at finding faults than other testing procedures, and advocates of these alternatives often claim that it is very inefficient. Essentially they argue that it takes no account of any knowledge that the software designer may have about the possible location of faults. With such knowledge it may be possible to find certain faults more effectively by actively seeking them, than by allowing them to occur purely randomly. In our terminology, the two testing procedures have different ‘difficulty’ functions over the set of faults: the optimal allocation of testing to the two types would depend upon the diversity between them as well as upon their individual efficacies.

The model presented here shows that the key to understanding how best to apply different fault finding procedures lies in understanding the interplay between, on the one hand, the efficacies of the individual procedures (in single and multiple applications), and on the other the dependence between their ‘difficulty functions’. Probably the most important results are those of Sections 3 and 4. There we show that the effects upon reliability of repeated applications of a particular procedure are not statistically independent - in fact such an incorrect assumption of independence will always give results that are too optimistic. When we have *diverse* fault finding procedures, however, things are different: here it is possible for effectiveness to be even greater than it would be under an assumption of statistical independence. We show in Result 8 that diversity of fault finding procedures is ‘a good thing’, and should be applied as widely as possible. However, this result relies upon assumptions of indifference between the different procedures. As we have seen in the example of Section 6, it is likely in practice that such an assumption of indifference would be unreasonable unless it had been deliberately contrived. We believe that for some procedures it may be possible to contrive it: in our own example it may be possible to increase the *amount* of testing to make the checking and testing equally effective (in both single and double applications).

When we compare this model with its mathematically similar equivalent in software design diversity, it is striking how much easier it is to obtain estimates of the key model parameters here. Thus in the example of Section 6, although the experiment pre-dates the model, we were able to find estimates of the parameters

representing procedure effectiveness and diversity. When we look at *classes* of faults, the estimation problem becomes a tractable one even for real life applications where the experimental replications of our experiment are infeasible. What is needed now is a systematic investigation of these parameters for different fault-finding procedures - and different classes of software application domains - in industrially realistic situations.

Intuitive notions of diversity in fault finding have been around for a long time, and are used informally quite extensively, but they have lacked a rigorous formal basis. In particular, it has not been clear what were the important factors to *measure*. The work reported here is the start of such a formal measurement-based understanding. We hope that it will lead to a theory of fault detection - and removal - that allocates different fault finding procedures optimally to each problem, taking account of the likely distribution of fault types for that problem.

Finally, we are grateful to a reviewer for pointing out that these results might have application to hardware as well as software. Whilst we have not had the opportunity to analyse any hardware data, we agree that the models are formulated in a very general way and could find use outside software engineering.

Acknowledgements

This work was supported partially by Scottish Nuclear under the DISPO project, by EPSRC under the DISCS project and by the ESPRIT Long Term Research Project 20072, 'Design for Validation' (DeVa).

References

- [1] J. C. Rouquet and P. J. Traverse, "Safe and reliable computing on board the Airbus and ATR aircraft", in Proc. Safecom: 5th IFAC Workshop on Safety of Computer Control Systems, 1986, pp. 93-97.
- [2] D. E. Eckhardt, A. K. Caglayan, J. C. Knight, L. D. Lee, D. F. McAllister, M. A. Vouk and J. P. J. Kelly, "An experimental evaluation of software redundancy as a strategy for improving reliability", IEEE Trans Software Eng, 17, pp. 692-702, 1991.
- [3] J. C. Knight and N. G. Leveson, "Experimental evaluation of the assumption of independence in multiversion software", IEEE Trans Software Engineering, 12, pp. 96-109, 1986.
- [4] D. E. Eckhardt and L. D. Lee, "A Theoretical Basis of Multiversion Software Subject to Coincident Errors", IEEE Trans. on Software Engineering, 11, pp. 1511-1517, 1985.
- [5] B. Littlewood and D. R. Miller, "Conceptual Modelling of Coincident Failures in Multi-Version Software", IEEE Trans on Software Engineering, 15, pp. 1596-1614, 1989.
- [6] P. Frankl, R. Hamlet, B. Littlewood and L. Strigini, "Evaluating testing methods by delivered reliability", IEEE Trans Software Engineering, 24, pp. 586-601, 1998.
- [7] E. N. Adams, "Optimizing preventive maintenance of software products", IBM J. of Research and Development, 28, pp. 2-14, 1984.

- [8] S. J. Westerman, N. M. Shryane, C. M. Crawshaw and G. R. J. Hockey, "Engineering Cognitive Diversity", in Proc. Fifth Safety Critical Systems Symposium, Brighton, 1997, pp.
- [9] S. J. Westerman, N. M. Shryane, C. M. Crawshaw, G. R. J. Hockey and C. W. Wyatt-Millington, "Cognitive diversity: a structured approach to trapping human error", in Proc. 14th International Conference on Computer Safety, Reliability and Security (Safecomp95), Belgirate, Italy, 1995, pp. 142-155.
- [10] N. M. Shryane, S. J. Westerman, C. M. Crawshaw, G. R. J. Hockey and J. Sauer, "Task Analysis for the investigation of human error in safety-critical software design: a convergent methods approach", *Ergonomics*, 41, pp. 1719-1736, 1998.
- [11] V. Basili and S. Green, "Software process evolution at the SEL", *IEEE Software*, 11, pp. 58-66, 1994.
- [12] V. R. Basili and R. Selby, "Comparing the Effectiveness of Software Testing Strategies", *IEEE Transactions on Software Engineering*, 13, pp. 1278-1296, 1987.
- [13] P. Frankl and S. N. Weiss, "An experimental comparison of the effectiveness of branch testing and data flow testing", *IEEE Transaction on Software Engineering*, 19, pp. 774-787, 1993.
- [14] R. B. Grady, "Practical Software Metrics for Project Management and Process Improvement", Prentice-Hall, 1992.
- [15] T. J. Shimeall and N. G. Leveson, "An empirical comparison of software fault tolerance and fault elimination", *IEEE Transactions on Software Engineering*, 17, pp. 173-182, 1991.