ELSEVIER

Information and Software Technology 39 (1996) 707–717

INFORMATION AND SOFTWARE TECHNOLOGY

Software diversity: practical statistics for its measurement and exploitation

D. Partridge[a,*], W. Krzanowski[b]

[a]*Department of Computer Science, University of Exeter, Prince of Wales Road, Old Library, Exeter EX4 4QE, U.K.*
[b]*Department of Mathematical Statistics and Operations Research, University of Exeter, Exeter EX4 4QE, U.K.*

Received 14 January 1997; revised 9 June 1997; accepted 17 June 1997

Abstract

The topic of this paper is the exploitation of diversity to enhance computer system reliability. It is well established that a diverse system composed of multiple alternative versions is more reliable than any single version alone, and this knowledge has occasionally been exploited in safety-critical applications. However, it is not clear what this property is, nor how the available diversity in a collection of versions is best exploited. We develop, define, illustrate and assess diversity measures, voting strategies for diversity exploitation, and interactions between the two. We take the view that a proper understanding of such issues is required if multiversion software engineering is to be elevated from the current ''try it and see'' procedure to a systematic technology. In addition, we introduce inductive programming techniques, particularly neural computing, as a cost-effective route to the practical use of multiversion systems outside the demanding requirements of safety-critical systems, i.e. in general software engineering. © 1997 Elsevier Science B.V.

Keywords: Diversity measures; Inductive programming; Multiversion systems; Neural computing; Software diversity; Voting strategies

1. Introduction

It is widely accepted that complex software will not be error-free, or if by chance it is error-free, we can never know that this is the case. This unsatisfactory situation is further aggravated by the ever-increasing demand for more reliability in computer-based systems. This demand has led to the emergence of extreme reliability requirements in some systems—ultra-high reliability for safety-critical systems. For example, aircraft guidance systems and nuclear reactor protection systems are two such safety-critical applications for which regulatory bodies may require reliabilities of 10^{-9} failures per hour and 10^{-5} failures on demand, respectively.

Given that error cannot be eliminated, the strategy for ultra-high reliability is to organize the software system (and its usage) in such a way that the inevitable errors do not cause system failure. One common approach to such a strategy of ''error distribution'' is to construct multiple implementations of the critical functions—a multiversion system. However, creating multiple versions is not enough. In order to reap benefit (in terms of enhanced reliability) the version set must be ''diverse''. Then a majority-vote of the versions, for example, may produce more reliable results than those from any single version in the set.

The necessary diversity can be in terms of either hardware or software. It is common practice in nuclear reactor protection systems, for example, to use different hardware technologies in the different versions (referred to as redundant channels). For software versions the diversity of process may be obtained by using very different target languages (e.g. Modula2 and Prolog [1], FORTRAN and assembly language [2]), or more fundamentally, by using deductive and inductive programming paradigms (e.g. conventional programming and neural computing). In summary, there is a wide variety of possibilities for constructing diverse sets of versions.

In this context, however, the term ''diversity'', although much used, is undefined and ill-understood. It is, for example, commonly equated with the property of lack of coincident failure, i.e. that different versions should fail on different inputs—we term this coincident-failure diversity. However, there is, for example, useful diversity (with respect to overall system reliability enhancement) that results from different versions failing differently on the same inputs—this we term distinct-failure diversity [3]. It is quite possible to have a 100% reliable system built from a low coincident-failure version set composed of individually unreliable versions, provided the distinct-failure diversity is high [4].

* Corresponding author. Tel.: 00 44 01392 264061; fax: 00 44 01392 264067; e-mail: derek@dcs.exeter.ac.uk

With certain technologies (both hardware and software), it is possible to have ''random scatter'' diversity which is exploited with an averaging decision strategy, rather than a voting mechanism [3]. Finally, it is possible to have ''expert'' diversity (each version, or subset of versions, is highly reliable on some subset of the input domain of the complete function); this type of diversity is exploited using a ''switching'' decision strategy—this mechanism selects the appropriate ''expert'' subset of the multiversions dependent upon some characteristic of each input to be computed [5].

The reliability of a multiversion system is a function of the decision strategy employed (which is itself constrained by the overall system architecture and the individual version technologies), the diversity characteristics of the available versions (both within and between the various subsets of versions), and some ''average'' level of version (or version subset) performance. Given a variety of versions, derived in accordance with a number of different methodologies, the choices for system design are many, e.g. one homogeneous set and majority voting, or a diverse system of subsets of versions and a majority in agreement of individual subset performances (which might each be the outcome of different decision strategies).

Clearly, the choices are many, but a relatively small number of diversity characteristics are, we suggest, the most important determinants in the one–many relation from diversity measures to system reliabilities. The aim of this paper is thus threefold:

1. To show that useful diversity is not limited to coincident-failure diversity.
2. To define and explore a few potential diversity measures as well as multiversion software decision strategies.
3. To examine and thus improve understanding of the role of diversity in multiversion system reliability. This final aim can be split into two major strands:

   (a) In the context of optimal multiversion system design and construction: given N versions, variously derived from different methodologies (A, B,...), there are many potential multiversion systems that could be constructed. Can we use diversity measures as an efficient guide to optimal system design, i.e. to select and arrange the N versions in conjunction with the most appropriate decision strategies?

   (b) In the context of a given multiversion system, can we systematically improve system reliability through measurement and understanding of its diversity makeup?

In a more general sense, this paper aims to improve understanding of the purely ''scientific'' issues of ''what exactly is diversity?'', and ''what role(s) does it play in multiversion system reliability?'' In terms of the purely pragmatic, this paper aims to explore the scope for efficient practical guidelines for systematic multiversion system construction and improvement, i.e. to begin to develop an engineering discipline of multiversion software construction to replace the current ''try and test reliability'' approach. For, although system reliability is considered to be the prime goal, a proper understanding of component diversities is thought to be a route to effective and efficient engineering of optimal systems.

It is immaterial whether the individual versions are hardware, software, conventional programs or neural networks. In general, we simply assume the existence of a set of classifiers.

2. Practical multiversion systems

The basic statistical framework is derived from the work of Littlewood and Miller [6] who presented a conceptual model of coincident failures in multiversion software. Their theory was developed in the context of infinite populations of versions and inputs (i.e. the populations of all versions that might ever be developed, and of all inputs that might ever be supplied), and they derived a number of useful statistics in terms of a few basic and simple probabilities.

In practical situations, data will only be available on finite sets of program versions and inputs, and, moreover, the former set is likely to be relatively small. We would often like to use Littlewood and Miller's statistics to describe such observed data, but substitution of the obvious estimates of the basic probabilities into their formulae may not give very accurate results. The basic problem is that these formulae implicitly rest on the idea of sampling with replacement for perfect accuracy, whereas all sampling in practice is without replacement. Of course, the larger the population the less difference is there between the two types of sampling, and in the limit (for infinite populations) there is no difference at all. However, if we have just a small number of inputs and versions, and we wish to derive accurate statistics *for these inputs and versions*, then we need to allow for sampling without replacement in the derived formulae. This is the main statistical focus of the present paper, but we also provide formulae for diversity estimation and examples of the utility of such estimates.

We first present our formal framework, which includes definitions of coincident-failure diversity (both within and between sets of versions). As illustrative examples, several decision strategies based on probability estimates for single-set systems (e.g. the probability that three randomly chosen versions are correct), and multi-set systems (e.g. the probability that three versions, each randomly chosen from a different set, are correct) are given. We then give some results of empirical studies that illustrate the various uses of diversity estimates in multiversion software engineering.

3. Single-set probabilities

We assume that the same M inputs are supplied to each of N versions, each version either being correct or failing on each input, and that the raw data consist of the frequencies m_i, the numbers of inputs on which i versions fail (for $i = 0, 1,\ldots, N$). The relative frequency $p_i = m_i/M$, thus gives the probability that i versions will fail simultaneously on a randomly chosen input from these populations of inputs and versions (for $i = 0, 1,\ldots, N$). Various probabilities of interest can now be calculated.

First, consider the probability that one of the versions (chosen randomly) will fail on one of the inputs (also chosen randomly)

$$p(1) = \Pr\{\text{one randomly chosen version fails on the input}\}$$
$$= \sum_{n=1}^{N} \Pr\{\text{exactly } n \text{ versions fail on this input and the chosen version is one of the failures}\}$$
$$= \sum_{n=1}^{N} \Pr\{\text{chosen version fails} \mid \text{exactly } n \text{ versions fail}\} * \Pr\{\text{exactly } n \text{ versions fail}\} = \sum_{N=1}^{N} \frac{n}{N} * p_n$$

Similarly,

$$P(2) = \Pr\{\text{two randomly chosen versions simultaneously fail on an input}\} = \sum_{n=1}^{N} \frac{n}{N} * \frac{(n-1)}{(N-1)} * p_n$$

and, in general,

$$p(r) = \Pr\{r \text{ randomly chosen versions simultaneously fail on an input}\}$$
$$= \sum_{n=1}^{N} \frac{n}{N} * \frac{(n-1)}{(N-1)} \ldots * \frac{(n-r+1)}{(N-r+1)} * p_n$$

for $r = 2,\ldots, N$.

Similar arguments can be used to calculate probabilities of a majority of versions being correct for a randomly chosen input. For example,

$$p(maj3) = \Pr\{\text{majority out of 3 randomly chosen versions are correct for the input}\}$$
$$= \Pr\{\textit{either } (0 \text{ out of } 3) \textit{ or } (1 \text{ out of } 3) \text{ versions fail on this input}\} = \sum_{n=0}^{N} \Pr\{\text{none of the chosen versions fail} \mid \text{exactly } n \text{ fail}\}$$
$$* \Pr\{\text{exactly } n \text{ fail}\} + \sum_{n=1}^{N} \Pr\{\text{one of the chosen versions fails} \mid \text{exactly } n \text{ fail}\} * \Pr\{\text{exactly } n \text{ fail}\}$$
$$= \sum_{n=0}^{N} \frac{(N-n)}{N} * \frac{(N-n-1)}{(N-1)} * \frac{(N-n-2)}{(N-2)} * p_n$$
$$+ 3 \sum_{n=1}^{N} \frac{n}{N} * \frac{(N-n)}{(N-1)} * \frac{(N-n-1)}{(N-2)} * p_n$$

Analogous expressions can easily be derived for the probability that the majority out of r randomly chosen versions are correct, for any r between 3 and N. However, these expressions become progressively lengthier and involve more sets of summations as r increases, so we do not present them here.

4. Coincident-failure diversity

Coincident-failure diversity of versions is generally considered operationally desirable (and usually thought to be the only important form of software diversity), so we now consider how it might be measured. Coincident-failure diversity is the property of different versions failing on different inputs, i.e. lack of coincident failure (with respect to input) within a version set.

Suppose that A and B are two randomly chosen versions and write

$$p(AB) = \Pr\{\text{both } A \text{ and } B \text{ fail on a randomly chosen input}\}$$

$$p(A|B) = \Pr\{A \text{ fails given that } B \text{ has failed}\}$$

$$p(A) = \Pr\{A \text{ fails}\}$$

Then from standard probability theory,

$$p(AB) = p(A|B)p(B) = p(B|A)p(A)$$

Now maximum diversity occurs when failure of one randomly chosen version is always accompanied by non-failure of another randomly chosen version. In this case $p(A|B) = 0$ for any A and B. Thus for maximum diversity $p(AB)$ is always zero.

Minimum diversity occurs when failure of one of the versions is always accompanied by failure of the other, and in this case $p(A|B) = 1$ for any A and B. Thus for minimum diversity we have $p(AB) = p(A) = p(B)$. Since each of the latter terms just represents the probability that a randomly chosen version fails on a given input, we can thus write $p(AB) = p(1)$ in the notation of Section 3.

The range of possible values of diversity for a given population of versions is thus $p(1) - 0 = p(1)$. Now the actual probability that A and B both fail for a given population is $p(AB) = p(2)$ in the notation of Section 3, which is $p(1) - p(2)$ below the value that corresponds to minimum

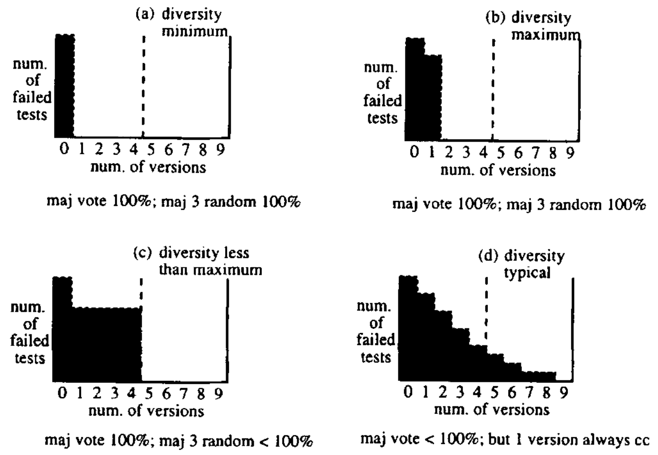

Fig. 1. Coincident failure, diversity and reliability.

diversity. Thus a standardized measure of diversity could be defined as

$$GD = \frac{p(1) - p(2)}{p(1)} = 1 - \frac{p(2)}{p(1)}$$

taking values between 0 (minimum diversity) and 1 (maximum diversity). This measure has been defined and explored in the context of neural computing [7].

However, it should be noted that while a version set with a GD value of 1 will always deliver a reliability of 100% when used in conjunction with a majority vote strategy, version sets exhibiting less than maximum diversity can be just as reliable. All that is required for a 100% majority-vote performance is that all common errors occur within a *minority of versions at most.* Fig. 1 illustrates some possibilities: the illustrations are of a system composed of nine versions and each bar chart is a record of the performance of this system on a set of test data. Each chart records how many of the test cases failed on precisely zero, one,..., nine versions. In bar chart (a) we see that all nine versions perform in an identically perfect manner (no tests fail on any versions). Notice that if some of the tests failed on all nine versions, the diversity would still be zero (as all nine versions are still behaviourally identical), but the voting performances would be less than 100%. Bar charts (b) and (c) illustrate versions with very different test performances and different diversities, but the same majority vote performance. Bar chart (d) illustrates a more typical distribution with a monotonic decrease in coincident failure. It does, however, exhibit no test failures on all nine versions, i.e. every test case was correctly computed by at least one version. The scope for exploitation of this situation will be discussed at the end of this paper.

These observations suggest that a coincident-failure diversity measure, for use as a reliable indicator of system performance using a voting strategy, should take coincident-failure distribution into account—large numbers of versions failing coincidently would be a negative indicator and small numbers of versions falling coincidently would be positive. In addition, we can see that diversity measures are voting-strategy sensitive, e.g. in the change from bar chart (b) to (c) the coincident failure distribution changes and the performance of the 'majority of 3 random choices' changes but that of the majority vote does not.

First, we require the probability that a test failure will fail on exactly n versions, f_n.

$$f_n = \frac{\text{number of tests failed on n versions}}{\text{the number of tests that fail on at least one version}}$$
$$= \frac{m_n}{m_1 + m_2 + \ldots + m_N} = \frac{m_n}{M - m_O}$$
$$= \frac{m_n/M}{1 - m_O/M} = \frac{p_n}{1 - p_O}$$

(providing that $p_O < 1$, i.e. that there is at least one failure in the set).

Now we can define Coincident Failure Diversity by:

$$CFD = \sum_{n=1}^{N} \frac{(N-n)}{(N-1)} f_n$$
$$= \frac{1}{1 - p_O} \sum_{n=1}^{N} \frac{(N-n)}{(N-1)} p_n \text{ (if } P_O < 1)$$

and $CFD = 0$ (if $p_O = 1$)

The maximum value of CFD is 1 when $f_1 = 1$ i.e. when all test failures occur on exactly one version (in this case $p_0 + p_1 = 1$). The minimum value of zero is obtained either when tests fail on zero and N versions only ($f_n = p_n = 0$, for $0 < n < N$) i.e. when all the versions are identical, or when no test failures occur ($p_0 = 1$), i.e. when all versions are identically perfect.

5. Inter-set diversities

Next suppose that there are two distinct sets of versions (e.g. different development environments, different testing regimes, different languages, i.e. different methodologies in Littlewood and Miller's terminology [6]), with N_A versions in the first set and N_B versions in the second set. If these two sets of versions are tested on the same M inputs, under the same conditions as before, the raw data now consist of joint frequencies m_{ij} representing the number of inputs for which i versions from the first set and j versions from the second fail simultaneously ($i = 0, 1, *, N_A; j = 0, 1, *, N_B$). From these frequencies we then obtain the joint probabilities $p_{ij} = (m_{ij})/M$ that i versions from one set and j versions from the other simultaneously fail on a randomly selected input.

Littlewood and Miller use the correlation coefficient computed from this probability distribution to quantify the diversity between the two sets of versions, but this measure is only appropriate for infinite populations (or sampling with replacement). For our finite population case, we require a generalization of the measure CFD developed above.

If we write

$$f_{ij} = \frac{\text{number of tests failed on } i \text{ versions in 1st set and } j \text{ in 2nd}}{\text{number of tests failing on at least one version in either set}}$$

then it readily follows that

$$f_{ij} = \frac{m_{ij}}{M - m_{00}} = \frac{m_{ij}/M}{1 - m_{00}/M} = \frac{p_{ij}}{1 - p_{00}}$$

in our previous terminology.

Extending the ideas of Section 4 to the two-set case, we want a measure of coincident-failure diversity between sets A and B, CFB_{AB} say, that satisfies

(a) $CFB_{AB} = 1$ when $\sum_{i=1}^{N_A} f_{i0} + \sum_{j=1}^{N_B} f_{0j} = 1$

(b) $CFB_{AB} = 0$ when $f_{N_A N_B} = 1$

There are various ways to satisfy these criteria, for example:

$$CFB_{AB} = \sum_{i=1}^{N_A} f_{i0} + \sum_{j=1}^{N_B} f_{0j} + \sum_{i=1}^{N_A}\sum_{j=1}^{N_B}\left[\frac{i(N_B-j)}{N_A N_B} + \frac{j(N_A-i)}{N_A N_B}\right] f_{ij}$$

$$= \frac{1}{1-p_{00}}\left\{\sum_{i=1}^{N_A} p_{i0} + \sum_{j=1}^{N_B} p_{0j} + \sum_{i=1}^{N_A}\sum_{j=1}^{N_B}\left[\frac{i(N_B-j)}{N_A N_B} + \frac{j(N_A-i)}{N_A N_B}\right] p_{ij}\right\}$$

(providing p_{00} 1, with CFB_{AB} defined to be zero when $p_{00} = 1$).

When $K > 2$ sets of versions exist, the raw data will consist of the joint frequencies $m_{ij\ldots k}$ giving the numbers of inputs jointly failed by i versions from the first set, j versions from the second set,..., k versions from the Kth set. For diversity, the above formulation of inter-set CFD can be extended to multi-set generalization in a direct way. For example, the three-set measure (in obvious notation) would be

$$CFB_{ABC} = \sum_{i=1}^{N_A} f_{i00} + \sum_{j=1}^{N_B} f_{0j0} + \sum_{k=1}^{N_C} f_{00k}$$

$$+ \sum_{i=1}^{N_A}\sum_{j=1}^{N_B} (i(N_B+1-j) + j(N_A+1-i))\frac{f_{ij0}}{(N_A N_B)}$$

$$+ \sum_{j=1}^{N_B}\sum_{k=1}^{N_C} (j(N_C+1-k) + k(N_B+1-j))\frac{f_{0jk}}{(N_B N_C)}$$

$$+ \sum_{k=1}^{N_C}\sum_{i=1}^{N_A} (k(N_A+1-i) + i(N_C+1-k))\frac{f_{i0k}}{(N_C N_A)}$$

$$+ \sum_{i=1}^{N_A}\sum_{j=1}^{N_B}\sum_{k=1}^{N_C} (i(N_B-j)(N_C-k)$$

$$+ j(N_C-k)(N_A-i) + k(N_A-i)(N_B-j)$$

$$+ ij(N_C-k) + jk(N_A-i) + ki(N_B-j))\frac{f_{ijk}}{N_A N_B N_C}$$

6. Two-level systems

One special case of particular interest is that of two-level systems, e.g. a system composed of three sets, each containing five versions (Littlewood and Miller [6] refer to these as "diverse" systems as opposed to single set or "homogeneous" systems). In such two-level systems, we may require the majority of K separate majority decisions (one for each set). Earlier results can also be generalized to obtain relevant probabilities in this case.

The particular statistics which we require are ones that give an estimate of the reliability of a two-level system: as an example, we consider a two-level system composed of three separate sets of versions, A, B and C, from each of which a majority is taken. The final result is then the majority decision from these separate majority decisions.

It is easiest to start with simple majority decisions and to assume that all sets contain an odd number of versions, each of which is either correct or incorrect (a failure) for all inputs.

Let $p(maj)_{n_X}$ denote the probability that a majority of randomly selected versions are correct when n versions in set X fail on a given input.

If $p(majmaj)_{ABC}$ is the probability that a majority of majorities, one majority each from sets A, B and C, is correct, then[1]

$p(majmaj)_{ABC} = prob((0$ out of three majorities)

or (one out of three majorities) fail)

$$= \sum_{n_A n_B n_C = 0}^{N_A N_B N_C} prob(\text{exactly } n_A,\ n_B \text{ and } n_C \text{ versions fail}$$

on this input *and* either zero *or* one of the individual majorities fail)

$$= \sum_{n_A n_B n_C = 0}^{N_A N_B N_C} prob(\text{either zero or one of chosen majorities}$$

fail | exactly n_A, n_B and n_C versions fail)

[1] In the interest of brevity, all triple summations have been contracted to a single summation, thus $\Sigma_{n_A=0}^{N_A}\Sigma_{n_B=0}^{N_B}\Sigma_{n_C=0}^{N_C}$ is written $\Sigma_{n_A n_B n_C=0}^{N_A N_B N_C}$

$$= \sum_{n_A n_B n_C = 0}^{N_A N_B N_C} prob\text{ (none of chosen majorities fail exactly } n_A,\ n_B \text{ and } n_C \text{ versions fail)} * p_{n_A n_B n_C}$$

$$+ \sum_{n_A n_B n_C = 0}^{N_A N_B N_C} prob\text{(none of chosen majorities fail exactly } n_A,\ n_B \text{ and } n_C \text{ versions fail)} * p_{n_A n_B n_C}$$

$$= \sum_{n_A n_B n_C = 0}^{N_A N_B N_C} prob\text{(none of chosen majorities fail exactly } n_A,\ n_B \text{ and } n_C \text{ versions fail)} * p_{n_A n_B n_C}$$

$$+ \sum_{n_A n_B n_C = 0}^{N_A N_B N_C} prob(A \text{ majority fails and } B,\ C \text{ majorities correct} | \text{exactly } n_A,\ n_B \text{ and } n_C \text{ versions fail)} * p_{n_A n_B n_C}$$

$$+ \sum_{n_A n_B n_C = 0}^{N_A N_B N_C} prob(B \text{ majority fails and } A,\ C \text{ majorities correct} | \text{exactly } n_A,\ n_B \text{ and } n_C \text{ versions fail)} * p_{n_A n_B n_C}$$

$$+ \sum_{n_A n_B n_C = 0}^{N_A N_B N_C} prob(C \text{ majority fails and } A,\ B \text{ majorities correct} | \text{exactly } n_A,\ n_B \text{ and } n_C \text{ versions fail)} * p_{n_A n_B n_C}$$

$$= \sum_{n_A n_B n_C = 0}^{N_A N_B N_C} p(maj)_{n_A} p(maj)_{n_B} p(maj)_{n_C} * p_{n_A n_B n_C}$$

$$+ \sum_{n_A n_B n_C = 0}^{N_A N_B N_C} (1 - p(maj)_{n_A}) p(maj)_{n_B} p(maj)_{n_C} * p_{n_A n_B n_C}$$

$$+ \sum_{n_A n_B n_C = 0}^{N_A N_B N_C} p(maj)_{n_A} (1 - p(maj)_{n_B}) p(maj)_{n_C} * p_{n_A n_B n_C}$$

$$+ \sum_{n_A n_B n_C = 0}^{N_A N_B N_C} p(maj)_{n_A} p(maj)_{n_B} (1 - p(maj)_{n_C}) * p_{n_A n_B n_C}$$

The simplest case is when three versions are randomly chosen from each set. The probability that a majority of three, randomly selected, versions are correct when exactly n versions fail (assuming N versions in total), $p(maj3)_n$, is

- 1 if $n \le 1$, because if only zero or one versions fail a majority of three must be correct;
- 0 if $(N - n) < 2$, because if only zero or one versions are correct then a majority of three must fail;
- a value between 1 and 0 otherwise, because it will depend on the chances of randomly selecting three versions within which at most one fails when n out of N versions actually fail.

The required expression is the probability that a majority of three randomly chosen versions is correct when exactly n versions fail.

$prob$(majority out of three are correct when exactly n fail)

$= prob$(at most one of chosen versions fails when exactly n fail)

$= prob$(either exactly zero fail or exactly one fails|exactly n fail)

$=$ prob(none of chosen versions fail|exactly n fail) $+ prob$(one chosen version fails|exactly n fail)

$$= \frac{(N-n)}{N} \frac{(N-n-1)}{(N-1)} \frac{(N-n-2)}{(N-2)} + 3 * \frac{n}{N} \frac{(N-n)}{(N-1)} \frac{(N-n-1)}{(N-2)}$$

$$= \frac{(N-n)}{N} \frac{(N-n-1)}{(N-1)} \frac{(N+2n-2)}{(N-2)}$$

where N is the number of versions in the set.

The probability that a majority of three versions is correct when exactly n versions fail, $p(maj3)_{n_X}$, (for set X), is then simply:

$$p(maj3)_{n_X} = \frac{(N_X - n_X}{N_X} \frac{(N_X - n_X - 1)}{(N_X - 1)} \frac{(N_X + 2n_X - 2)}{(N_X - 2)}$$

Notice that this expression equals 1 when $n_X < 1$, and it equals 0 when $(N_X - n_X)$ is 0 or 1.

If $p(majmaj3)_{ABC}$ is the probability that a majority of three majority-of-three votes, one majority each from sets A, B and C, is correct, then

$p(majmaj3)_{ABC}$

$$= \sum_{n_A n_B n_C = 0}^{N_A N_B N_C} p(maj3)_{n_A} p(maj3)_{n_B} p(maj3)_{n_C} * p_{n_A n_B n_C}$$

$$+ \sum_{n_A n_B n_C = 0}^{N_A N_B N_C} (1 - p(maj3)_{n_A}) p(maj3)_{n_B} p(maj3)_{n_C} * p_{n_A n_B n_C}$$

$$+ \sum_{n_A n_B n_C = 0}^{N_A N_B N_C} p(maj3)_{n_A} (1 - p(maj3)_{n_B}) p(maj3)_{n_C} * p_{n_A n_B n_C}$$

$$+ \sum_{n_A n_B n_C = 0}^{N_A N_B N_C} p(maj3)_{n_A} p(maj3)_{n_B} (1 - p(maj3)_{n_C}) * p_{n_A n_B n_C}$$

7. Engineering multiversion software

The prohibitive cost of generating a set of versions, added to the discouragingly small increase in the resulting system reliability, has meant that multiversion systems have been seldom used. However, with the development of inductive programming techniques (such as inductive logic programming [8], decision-tree induction [9] and neural computing [10]) as practical software engineering technologies, the case for multiversion software engineering emerges in a much more positive light.

This is primarily for two reasons. First, inductive techniques are largely automatic (once the initial data have been analysed and organized). Hence the cost argument against a multiversion approach is undermined—automatic version generation is much cheaper than manual algorithm design and development. Second, because the characteristics of the final implementation (including its failure points) are determined by the initial conditions (to automatic development), there is scope for engineering version sets to contain the required level of diversity. This can be done either by assessment of the diversity-generating potential of each variable of the initial conditions and systematic exploitation of the most powerful ones, or by overproducing the numbers of versions required and selecting a maximally diverse set to constitute the final system. Most probably, both techniques should be used.

Suffice to say, there is a variety of inductive programming techniques, and, although not without problems, they have proved their worth in a wide range of practical applications. The firm expectation is that usage of such techniques in practical software engineering will increase in number and expand in scope [11]. A further important consequence of the shift in multiversion software economics revealed by the introduction of inductive programming is that the choice of system architectures (arrangement of version sets and decision strategies) is enormously expanded. This means that design, construction and enhancement of multiversion software must be guided by more than measures of overall system reliability. There are too many options for such a "try it and see" strategy to be effective.

In order to engineer such systems (or system enhancements) properly we must exploit sound knowledge of the underlying technology—one major component of which is "diversity". In the following section we shall demonstrate some of the complexities of the nature and role of useful diversity in multiversion software, and we shall present some strategies to support multiversion software development as an engineering discipline.

8. Illustrative examples

8.1. The diversity of diversities

The varieties of useful diversity, as mentioned earlier, are not well understood. We have, however, explored three distinct types beyond coincident-failure diversity.

8.1.1. Distinct-failure diversity

All formulae, developed above, are based on the binary distinction between success and failure as the output of a classifier. When the target functions are Boolean then success and failure is all there is, but in general there may be c (> 1) distinct wrong answers. In this latter case, the current measures will still be applicable—all wrong answers are simply failures.

Application of the current statistics to a problem with c (> 1) distinct wrong answers will produce a minimum diversity and a worst-case estimate of performance improvement. It will only be accurate if the classifiers do always produce identical wrong answers which is a worst case situation (because any differentiation in the failure category will be potentially exploitable by an appropriate voting strategy). Differentiation of the "failure" category into c distinct alternatives can add considerable extra scope for diversity to be exploited. Diversity measures gain a second dimension: n coincident failures may be all distinct or all identical (or any intermediate distribution), where $n \leq c$. Elsewhere [4], distinct-failure diversity has been defined and explored. Briefly, if t_n is defined as (the number of times that n versions fail identically) divided by (the total number of distinct input failures), then distinct-failure diversity, DFD, can be defined:

$$\mathrm{DFD} = \sum_{n=1}^{N} \frac{(N-n)}{(N-1)} t_n$$

if (total number of identical test failures) > 0, otherwise DFD $= 0$.

Similarly, voting strategies further proliferate. For example, with c distinct failures majority-in-agreement is no longer the same as simple majority-vote (as it is when $c = 1$). So if 15 classifiers, say, produce just two correct answers together with 13 distinct failures, then a majority-in-agreement strategy will deliver a correct overall result despite the fact that correct answers are in a

Table 1
Two types of useful diversity

System	CFD	DFD	Average	Majority vote	Majority in agreement
$pick_{\mathrm{CFD}}$	0.84	0.95	79.81%	85.87%	91.10%
$pick_{\mathrm{DFD}}$	0.73	0.97	69.92%	81.57%	89.42%

Table 2
A comparison of differently diverse multiversion systems

Nine-version system	Average	Majority vote	Expert ensemble	CFD
*LIC*4$_{\text{exp}}$	(60.7%)	(51.2%)	98.3%	0.28
*LIC*4$_{\text{mvs}}$	71.9%	96.2%	–	0.75

substantial minority. In addition, coincident-failure diversity may be zero (i.e. every test failure involves ($N - 2$) versions) which demonstrates that DFD may be useful diversity in its own right, quite independent of coincident-failure diversity.

In the context of a multiversion neural-net system for handwritten character recognition, the OCR problem, each version has 26 output units (one corresponding to each of the 26 output classifications, i.e. the letters "A" to "Z"), and 16 input units (one for each feature value). The 16 numerical features are defined in Frey and Slate [12]. Ideally, each of the input feature vectors should result in a value of 1.0 in the correct classification for that input and a value of 0.0 in the other 25 output units. Actual output values are never this extreme; the maximum of the 26 output values may be selected as the category computed by each version. For this task, c, the number of distinct wrong answers, is 25.

Two 15-version systems were constructed [3]: in one the construction principle was to maximize CFD, this was the $pick_{\text{CFD}}$ system; in the other, DFD was maximized, this was the $pick_{\text{DFD}}$ system. Table 1 illustrates the diversity values measured and the reliability of these systems under two decision strategies. The column headed "average" records the average performance of the 15 versions, and the fifth and sixth columns record system performance under simple majority-vote and majority-in-agreement decision strategies, respectively.

As can be seen, proper exploitation of distinct-failure diversity can enhance system reliability over and above the reliability gain from coincident-failure diversity.

Using the results of a multiversion study of Prolog and Modula2 versions [1], we show [4] that exploitation of distinct-failure diversity could further reduce the residual error reported by as much as 56% for the set of six Modula2 versions.

8.1.2. Specialization diversity

Exploitation of both CFD and DFD relies on voting decision strategies that are based on indifference with respect to version selection. However, within a set of classifiers there is potential for specialization: some subset of the classifiers may be high-performance specialists on some subset of the problem inputs. In which case, the best strategy is one of dynamic prioritization, i.e. "favour" a specific subset of versions dependent upon some characteristic of the specific input. Bar chart (d) in Fig. 1 illustrates a situation in which a version-specific decision strategy can (potentially) deliver a 100% performance, whereas voting will not.

In the limiting case, each version is an "expert" on a circumscribed subset of the domain of the function computed. If the intersection of all such subsets is empty, and the union is the complete function, then a 100% reliable system can be constructed. It requires that each "expert" version is 100% reliable (but only on a subset of the complete function), and that the appropriate expert can be identified from the input data. Notice that in this case of "ideal" specialization diversity, both coincident-failure diversity measures (i.e. GD and CFD) collapse to $1/(N-1)$, because $p_{N-1} = 1.0$ and all other $p_n = 0.0$. The previous diversity measures can thus also provide an indication of the degree to which we have specialization diversity in a given version set.

This type of multiversion system has been explored using neural computing technology, where the systems are typically called "expert ensembles" [5]. This exploration of expert specialization shows it to be a useful technique which produces multiversion systems that exhibit higher reliability than voting multiversion systems when the problem can be decomposed into appropriate subproblems. In addition, it was shown that coincident-failure diversity measures did provide a guide to specialization diversity as well.

In Table 2, results from [5] show two alternatively diverse multiversion systems for the same problem. The first system, *LIC*4$_{\text{exp}}$, was designed as an expert ensemble. The two reliabilities in parentheses are obtained when a majority-vote decision strategy is applied, rather than the switching strategy for which it was designed. The second system, *LIC*4$_{\text{mvs}}$, was designed as a majority-voting system with the emphasis being on maximizing the CFD measure within the nine versions. These systems illustrate particularly well the desirability of matching diversity characteristics with decision strategy—if the expert ensemble, *LIC*4$_{\text{exp}}$, is (mis)treated as a majority voting system it delivers only 51.2% reliability, but if treated as an expert ensemble it delivers 98.3% reliability. Notice that the CFD values are indicative of these results, an "ideal" CFD value for a nine-version expert ensemble would be 1/8, or 0.125.

8.1.3. Random scatter diversity

In addition to all such voting decision strategies (indifferent with respect to version selection or not), another class of decision strategy and associated diversity has proven to be useful. Certain types of classifier (e.g. hardware circuits and neural networks) can be used to perform continuous, rather than discrete, computations. Thus a Boolean result of 0 or 1, for example, may be generated as a real value in the

Table 3
Architectural choices and diversity characteristics

Majority of three performance		3 × 5 system diversities			
Flat 15	3 × 5	CFB_{ABC}	Average CFDs	Inter − intra difference	Relative increase
97.60%	97.41%	0.67	0.92	− 0.25	− 0.19%
98.41%	98.43%	0.80	0.83	− 0.03	0.02%
98.43%	98.52%	0.83	0.71	0.12	0.09%

range 0.0–1.0 which is typically thresholded to obtain the required discrete result. However, classifiers that do produce such real-number approximations to the discrete targets may be treated with an averaging strategy (rather than threshold and voting) followed by thresholding to obtained the overall system performance. Although alien to conventional computing, this particular approach to multiversion software has been widely used in neural computing.

In the context of a multiversion neural-net system for hand-written character recognition, OCR, an appropriate decision strategy is to sum the output value in each category across all versions, and select the maximum as the system outcome. If this decision strategy is applied to the two multiversion systems illustrated in Table 1, $pick_{\mathrm{CFD}}$ and $pick_{\mathrm{DFD}}$, we obtain reliability values (using the same test set as previously) of 92.95 and 93.47%, respectively [3].

8.2. Diversity as a guide to system construction

Several examples in the previous section demonstrate clearly that the class of decision strategy and the type of diversity exhibited by the version set must match, otherwise a suboptimal multiversion system may be constructed. This use of diversity measures (e.g. whether CFD $\rightarrow$ 1.0 or whether CFD $\rightarrow 1/(N-1)$) requires that a version set already exists, but we can use diversity measures to guide the process of version set development as well, as the first point below describes.

8.2.1. Forcing diversity

Analytical and empirical study of diversities achieved by controlled version development processes can lead to strategies for engineering version sets that are maximally diverse (of one type or another).

Such a study of the GD measure [7] produced the following ranking of the coincident-failure diversity generating potential of some important parameters in neural computing:

$$\text{net type} > \text{training set} > \text{training set structure}$$
$$> \text{number of hidden units} \simeq \text{weight seed}$$

This information was subsequently used in a study to engineer reliable multiversion neural-net systems [13]. For example, a system composed of the 15 best (in terms of correctness on a test set) networks averaged 97.97% correct (on a further test set) and exhibited a GD of 0.53. A further system was composed of 15 networks generated to exploit the above-listed sequence of network generating potential; the average (on the same further test set) was 97.83% and the GD was 0.65.

However, the majority vote performance of the first system was 98.48%, but of the second, more coincident-failure diverse system it was 98.85%. Systematic exploitation of the diversity generating potential of the major parameters controlling version production had produced a 24% decrease in residual error.

8.2.2. Systematic version selection

Given a set of versions, the best multiversion system may be constructed from a maximally diverse subset. Diversity measures can be used to select such subsets.

In one study, two such selection techniques were explored: one was a heuristic selection algorithm, known as pick, the other was a genetic algorithm, and both attempted to maximize a given diversity measure [13].

When 15 versions were selected from a pool of 500 (averaging 95.34%) with the goal of maximizing the GD measure, the pick heuristic system exhibited an average performance of 95.44% and a GD value of 0.78. The 15 versions selected by the genetic algorithm averaged 84.79% and exhibited a GD value of 0.79. In the former case the majority vote performance (on the same test set as the example above) was 99.31%, and in the latter it was 99.08%. In both cases the result is better than that for the system composed of the 15 best versions mentioned above (i.e. 98.48%) with which it is directly comparable.

8.2.3. System architecture choice

The construction of a multiversion system requires choice between single- or two-level architectures (homogeneous or diverse, in Littlewood and Miller's terminology). As a simple illustration we can compare performance and diversity measures of 15 versions, configured either as a single set or as three sets of five versions—a "flat" 15 or a 3 × 5 system.

Several decision strategies might be considered, for example, majority-vote of the "flat" 15 system compared with "majority of three majorities" for the 3 × 5 system. However, in this case the single-level system is never worse than the two-level system because in the former architecture the "best" eight versions can always deliver an optimal result. Yet in the two-level architecture, any subset of five versions can only contribute, at most, three versions to the final outcome.

However, if we confine the decision strategy to a simple "majority of three" vote, then we can directly compare the system performances when the three versions are selected totally at random from the "flat" 15 system and when the three are selected, at random, one from each of the five-version subsets.

In Table 3 some comparative results are given [13]. The fourth column in the table gives the average CFD value of the three versions sets in a 3 × 5 system. The fifth column gives the difference between inter-set diversity and the average of the three intra-set diversities, and the final column gives the percentage improvement in system reliability obtained by the architectural rearrangement from the one- to a two-level system.

In the first two of the three systems listed the diversity within each subset is greater than the diversity between each subset, although in the second they are close to equal. This characteristic indicates that the two-level system is likely to be inferior to the single-level one. The results support this in the case of the first system where the difference is (relatively) large. In the first system, because the diversity within the component five-version sets is high (actually, 0.92, 0.92 and 0.91, compared with 0.87 for the "flat" 15), the chosen decision strategy can be more advantageously applied to the most diverse component set to obtain a performance of 98.40%. In addition, it can be seen that the greatest reliability enhancement (7% reduction in residual error) achieved by the architectural switch from single-level to a two-level system configuration, is obtained in the last system illustrated. This result is also indicated by the diversity measures. The (relatively) large positive difference between inter- and intra-set diversities indicates that there is more diversity to exploit between sets than within sets, and the chosen decision strategy exploits only inter-set diversity, whereas a "majority of three majorities" decision strategy would exploit both sources of diversity (it yields a 98.82% performance, but requires the evaluation of up to 15 versions rather than just three).

8.2.4. Methodological diversity choice

Littlewood and Miller [6] argue persuasively that diversity of process (which they term "methodology") should lead to diversity of product. However, given the almost limitless options for process diversity, in practice there is a need to determine the most cost-effective processes, or methodologies, to employ in multiversion software construction. The first of the techniques described above (i.e. diversity generating potential of the major process parameters) is one way to approach this practical requirement. We need to know which methodologies generate most diversity (of a particular type), and diversity measures, such as we suggest, provide a means to answer this question.

For example, in a recent study we have confirmed Adams and Taha's conclusion that the different implementation languages, Prolog and Modula2, are methodologies that generate sufficient coincident-failure diversity to cancel the negative effect of input data variance [14], i.e. the resultant product diversity is sufficient to nullify the adverse effect of inherently difficult inputs causing clusters of version errors on an acknowledged benchmark problem. However, further study of this problem using neural-net versions suggests that neural computing is not as methodologically diverse with respect to Prolog or Modula2 as these two conventional methodologies are to each other [14].

Table 4
The interaction of coincident-failure diversity and average performance

Version average	CFD	Majority-vote
97.97%	0.735	98.48%
97.09%	0.909	98.53%
96.31%	0.887	98.53%
95.59%	0.867	98.51%
95.08%	0.871	98.51%

8.3. Diversity as a guide to system reliability enhancement

In this subsection we consider the use of diversity measures to guide the systematic enhancement of a given multiversion system. The task is thus not to build a system from scratch, but to determine how best to improve the performance of a system that already exists. This might involve substituting or adding new versions, or a complete version set. It might also involve a major architectural rearrangement, or less drastically a change of decision strategies used. This use of diversity measures is a special case of that considered in Section 8.2, and, until such time as the overall technology is better understood, it is also usefully viewed as a component of the system-building task. Given the incomplete state of our knowledge with respect to the diversity generating potential of methodologies, a strategy of "overproduce and select" might be usefuly employed, as we have demonstrated in the context of inductive technologies [13].

8.3.1. Diversity extremes

With respect to the choice between expert ensemble and voting systems, we have already demonstrated that extreme values of coincident-failure diversity can indicate optimal decision-strategy choice. In addition, diversity measures that approach 1.0 (either CFD or DFD) would indicate that attempts to generate further versions to improve these diversity characteristics would be fruitless. Conversely, very low values of these measures would indicate that new versions, appropriately generated, could be used to enhance system performance.

8.3.2. Diversity within and between sets

We have shown (Table 3) that a consideration of the relative magnitudes of diversity within and between versions sets can indicate the relative advantages of single- and two-level architectures. They can also, as we have also pointed out, indicate the relative advantages of competing decision strategies, such as "majority of three majorities" as opposed to "majority of three random."

8.3.3. Diversity vs. average performance

It is clear that multiversion system reliability (for a given architecture and decision strategy) is primarily dependent upon the diversity characteristics of the versions and on some "average" performance level of the individual versions. In general, both "features" should be as high as possible. Both can be measured for a given system, and (other things being equal) whichever appears to offer most scope for improvement might be taken as the guide for further version generation in order to improve system performance—emphasis on a new version to increase diversity will suggest a different version-generation process from that suggested by emphasis on increased individual performance.

As the data [15] in Table 4 illustrates, substitution of inferior performance version, that improve diversity, can more than compensate for the decrease in average performance of the version set. The first row gives the performance and diversity of a 15-version system. In successive rows the best individual version in the set is exchanged for an inferior (but more diverse) version.

9. Conclusions

We have shown that multiversion software development is not simply a matter of collecting a minimum-coincident failure set of versions and applying a majority vote strategy to see how reliable the resultant system happens to be. The possibilities are many (especially when inductively generated versions are considered) and the relationships between diversity, architecture, individual version performance and decision strategy do not appear to be straightforward. Multiversion software design is not simple, but if we aspire to engineer such systems efficiently, then a "try and see" strategy is not good enough. We must develop an understanding of the essential components of reliability enhancement and determine their inter-dependencies.

Design must be through demonstrated processes of maximum diversity generation. This presumes knowledge of diversity generation potential of the "parameters" of a methodology which will be discovered through experimentation with diversity measures. The starting point is methodologically diverse subsets of versions. The measures of diversity achieved both within and between these subsets (together with average performance measures) provides the basis for the major architectural decisions, e.g. single- or two-level system. Diversity measures of the resultant version set (or sets) will indicate the appropriate class of decision strategy to employ, as well as provide specific guidelines for version set enhancement (e.g. more reliable or more diverse extra versions).

However, the overall picture is far from clear. What is clear is that coincident-failure diversity is not the whole story, nor even a necessary part of every attempt to build multiversion software. The conventional wisdom, founded on the severely cost-limited possibilities for conventional programming of multiversion systems, is badly wrong. We hope to have made this clear as well as made a positive contribution to a future discipline to exploit the full scope of multiversion software systems.

Acknowledgements

This work was partially supported by a grant (number GR/H85427) from the EPSRC/DTI Safety-Critical Systems Programme and a subsequent grant from the EPSRC (number GR/K78607). In addition, Phillis Jones is thanked for her help in generating new results as well as checking our accuracy.

References

[1] J.M. Adams, A. Taha, An experiment in software redundancy with diverse methodologies, Proceedings of the 25th Hawaii Conference on Software Systems, January, 1992.

[2] G. Dahll, M. Barnes, P. Bishop, Software diversity: way to enhance safety?, Information and Software Technology 32 (10) (1990) 677–685.

[3] D. Partridge, W.B. Yates, Data-defined problems and multiversion neural-net systems, Journal of Intelligent Systems 7 (1-2) (1997) 19–32.

[4] D. Partridge, W.J. Krzanowski, Distinct failure diversity in multiversion software, Research Report number 348, Department of Computer Science, University of Exeter, 1996.

[5] N. Griffith, D. Partridge, Self-organizing sets of experts, Research Report number 349, Department of Computer Science, University of Exeter, 1996.

[6] B. Littlewood, D.R. Miller, Conceptual modelling of coincident failures in multiversion software engineering, IEEE Transactions on Software Engineering 15 (12) (1989) 1596–1614.

[7] D. Partridge, Network generalization differences quantified, Neural Networks 9 (2) (1996) 263–271.

[8] N. Lavra, L. De Raedt, Inductive logic programming: a survey of European research, AI Communications 8 (1) (1995) 3–19.

[9] D. Michie, Methodologies from machine learning in data analysis and software, The Computing Journal 34 (6) (1991) 559–565.

[10] B. Widrow, D.E. Rumelhart, M.A. Lehr, Neural networks: applications in industry, business and science, Communications of ACM 37 (3) (1994) 93–105.

[11] D. Partridge, The case for inductive programming, IEEE Computer 30 (1) (1997) 36–41.

[12] P.W. Frey, D.J. Slate, Letter recognition using Holland-style adaptive classifiers, Machine Learning 9 (1991) 161–182.

[13] D. Partridge, W.B. Yates, Engineering multiversion neural-net systems, Neural Computation 8 (1996) 869–893.

[14] D. Partridge, N. Griffith, D. Tallis, P. Jones, An experimental evaluation of methodological diversity in multiversion software reliability, Research Report number 358, Department of Computer Science, University of Exeter, 1997.

[15] W.B. Yates, D. Partridge, Use of methodological diversity to improve neural network generalisation, Neural Computing and Applications 4 (2) (1996) 114–128.