Software diversity as a measure for reducing development risk

Peter Popov, Andrey Povyakalo, Vladimir Stankovic, Lorenzo Strigini
Centre for Software Reliability
City University London
Northampton Square, London EC1V 0HB, U.K.
{ptp, andrey, v.stankovic, l.strigini}@csr.city.ac.uk

***Abstract*—Despite the widespread adoption of software diversity in some industries, there is still controversy about its benefits for reliability, safety or security. We take the prospective of diversity as a *risk reduction* strategy, in face of the uncertainty about the dependability levels delivered by software development. We specifically consider the problem faced at the start of a project, when the assessment of potential benefits, however uncertain, must determine the decision whether to adopt diversity. Using probabilistic modelling, we discuss how different application areas require different measures of the effectiveness of diversity for reducing risk. Extreme values of achieved reliability, and especially, in some applications, the likelihood of delivering "effectively fault-free" programs, may be the dominant factor in this effect. Therefore, we cast our analysis in terms of the whole distribution of achieved probabilities of failure per demand, rather than averages, as usually done in past research. This analysis highlights possible and indeed frequent errors in generalizations from experiments, and identifies risk reduction effects that can be proved to derive from independent developments of diverse software versions. Last, we demonstrate that, despite the difficulty of predicting the actual advantages of specific practices for achieving diversity, the practice of "forcing" diversity by explicitly mandating diverse designs, development processes, etc., for different versions, rather than just ensuring separate development, is robust, in terms of worst-case effects, in the face of uncertainty about the reliability that the different methods will achieve in a specific project, a result with direct applicability to practice.**

Keywords- software diversity; multiple version software; software fault tolerance; forced diversity; diversity-seeking decisions; distribution of reliability; fault-freeness

I. INTRODUCTION

In fault-tolerant systems, the risk of design faults, replicated in the redundant components, causing common failures can be reduced by *diversity* among the redundant components. In the simplest case, two or more *versions* of these components are built independently, and connected in an architecture such that the system will perform correctly, or safely, if a certain quorum of them does. In addition to trying to ensure independence between the developments of the version, measures are usually applied for making the development processes and the designs of diverse programs as different as possible [1-4]. We will call these latter techniques, collectively, "*forcing*" diversity, and diversity obtained by just keeping the version developments separate, "*unforced*" diversity.

Diversity is an established principle in some critical applications of computers, e.g. in nuclear safety and in aviation; the recent automotive safety standard ISO26262 [5] lists "diverse design" as "highly recommended" as a "Mechanism for error detection at the software architectural level" at ASIL D (the highest "Automotive Safety Integrity Level"). There have been extensive studies about its effectiveness, to address questions like: how much advantage can one expect from a redundant configuration using diversity, compared to a single program? However, the usefulness of software diversity is still a controversial topic in many forums.

There are really two, quite distinct, decision problems with *any* technique for improving dependability:

- deciding which techniques to *apply*, in which form. That is, predicting what results should we expect from applying the technique. The actual dependability of the

final product is *a priori* uncertain, but do the likely results – e.g. the range of likely improvement – justify "betting" the cost of applying a specific technique (as an additional cost, or rather than spending on another dependability-enhancing technique – e.g., choosing between investing in massive extra V&V of a single program, or in multiple versions)?

- once a product is finished, when *assessing* how dependable this specific product is, how can we use the fact that the technique was applied to improve our confidence in the product's dependability, or reduce the cost of achieving this confidence?

The two questions are obviously related. For critical applications, no developer would make important design choices without an expectation that the end client, and/or the regulators for the application sector, will accept them; the developer will generally try to make choices that will gain "credit" with these judges, e.g. choices recommended by safety-related standards. In the probabilistic assessment of the final product, if applying a rigorous Bayesian process a pre-development assessment would contribute to *prior* distributions, to be updated with detailed knowledge about the final product. However, this pre-development assessment can only be imprecise. Establishing a software development process, a set of techniques to apply, and a product architecture and even choosing the development team and organisation does not determine the exact level of dependability of the final product. There are known counterexamples to the hypotheses that additional dependability techniques *always* reduce bugs, improve dependability, or achieve the dependability levels they are meant to achieve [6, 7]. To be considered good for dependability, a software engineering technique should certainly deliver an improvement in the *statistics* of achieved dependability, but cannot reasonably be expected to deliver a specific level, or specific degree of improvement, at every use.

Developers have to live with the impossibility of precise answers, "applying this technique, in this organization, will deliver *this* failure rate", to the first question; but even rather imprecise answers will often be enough. For instance, knowing that a technique tends to achieve its intended result in 99 out of 100 products may be enough to justify the cost of using it, e.g. because the gamble is no worse than other business gambles; or because the extra cost of the 1 product in 100 that requires extra work to satisfy its customers can be subsidized by the successful ones.

Here we address this first question: what reliability improvements will diversity deliver? As often with techniques for high reliability, an *empirical*, *usefully precise* answer is now unfeasible: there are not enough data. Even less feasible would be an empirically based answer for those "u-high reliability" products [8, 9], for which estimating the reliability of even one product with the desired precision is unfeasible, and which are one of the intended areas of application for diversity. We describe instead mathematical considerations that we believe useful for practitioners, although – obviously – mathematical truths can only be trusted to apply if the assumptions on which they are based are true.

We aim to achieve various benefits: gaining clarity about the relationship between commonly used estimates of the benefits of diversity and the risk measure of actual interest; improving the methodological approach to analyzing experimental results in this area; and direct indications to help decisions about whether and how to apply diversity.

For a start, how should one *describe* the range of improvement from diversity: which measures should one use? A good way of framing the problems is that of the *risk* that the developer takes, and how much this risk is reduced by precautions like diversity. Uncertainty about what the development will produce naturally leads to seeing product development as a stochastic process, followed by operation, another stochastic process. One invests in dependability-enhancing techniques to reduce the probability of undesired events, e.g., of experiencing too many system failures in operation.

The early proposals for software diversity did not attempt to quantify the advantages to be expected, but some claims were made that independent development of the versions would produce independence between their failures [10]. In the 1980s, an important

experiment by Knight and Leveson [11] proved this wrong by showing a counter-example, and probabilistic models developed by Eckhardt and Lee [12] and Littlewood and Miller [13] explained this lack of independence as a general pattern. These models consider that there is uncertainty about both the reliability of the individual programs that can be combined into fault-tolerant systems, and about that of the fault-tolerant systems themselves. Their probabilities of failure are thus random variables. If we call q_{single} the probability of failure per demand (*pfd*)[1] of a program, and q_{pair} the probability of common failure (i.e., of both versions in the system failing on the same demand) of a "1-out-of-2" pair of programs (the pair fails only if both programs fail[2]), Eckhardt's and Lee's result was about the expected values of these two random variables, and stated $E(q_{pair})>(E(q_{single}))^2$. If all pairs of versions failed independently, we would instead have $E(q_{pair})=(E(q_{single}))^2$. The result that $E(q_{pair})>(E(q_{single}))^2$ is explained by similarities between which demands are "difficult" for the developers of the diverse versions. Littlewood and Miller pointed out that it was possible for such similarities of "difficulty" not to exist, and actually this is the goal of the many ways that diversity is pursued [1, 4]. It is possible in principle not only to achieve $E(q_{pair})=(E(q_{single}))^2$ but even better results, with diverse versions usually having negatively correlated failures, even to the point of *no two versions ever failing on the same demand*, so that $E(q_{pair})=0$. However, it is not feasible to tell a priori that the means applied to achieve diversity have been so successful, or even successful enough to achieve $E(q_{pair})=(E(q_{single}))^2$, and thus it would be prudent (pessimistically) to expect $E(q_{pair})>(E(q_{single}))^2$.

Valuable as the insight from such models is, both to an assessor and in suggesting ways of making diversity more effective [14],[15], they do not go very far in addressing the developer's decision problem. If for instance we were to combine two versions with $q_{single}=10^{-4}$ *pfd* into a 1-out-of-2 system, the inequality above leaves open a range of possibilities, from diversity being extremely useful (e.g., $q_{pair}=10^{-7}$), to it being useless (e.g., $q_{pair}=10^{-4}$).

Several authors have tried to extrapolate from the results of experiments the size of likely gains in industrial use. We will discuss serious difficulties with such generalization. The problem is not just whether an experiment can be trusted to be representative of the industrial application about which we wish to learn. More importantly, it is unclear which measures of the effects of diversity in one experiment should be chosen as likely to hold in other projects. In looking at these results, it is easy to rely on intuitions that are only appropriate for simple distributions of the variables involved, like narrow bell-shaped curves; but in reality these distributions are likely to be discrete and irregular. Instead of trying to generalize from these data, we state purely mathematical results, that are true if the assumptions used are true. We use an example data set from an experiment just to *illustrate* these mathematical facts.

In the rest of this paper, we discuss these issues with reference to 1oo2 architectures, which despite their simplicity are widespread in practical applications, e.g. self-checking pairs in safety critical architectures or duplex servers with fail-silent elements. Section II proposes various measures of risk, applied to different environments, and introduces a number of probabilistic results, illustrated on the data from a well-known experiment on diverse software. Section III focuses on the effects of "forcing" diversity, introducing a useful theorem about its effect on the likelihood of achieving target levels of reliability. Section 0 sketches scenarios of how the statistics of the *pfd* distribution affect the gains from diversity and their predictability. Our conclusions follow in the last section.

II. RISK AND DIVERSITY. AN EXAMPLE

To illustrate the motivations of this paper, we take a set of experimental data and subject them to some new analyses. For this example we use the published numerical data

---

[1] These models refer to "on demand" operation, where the probability of failure concerns a single call or "demand" on the system considered – the probability of interest is thus the *pfd* – although extensions to the continuous-time case are not problematic. We also use the "on demand" scenario.

[2] We will use the abbreviation 1oo2 for "1-out-of-2".

from Knight's and Leveson's widely cited experiment (which we will call "the KL experiment" for brevity), funded by NASA to check the conjecture that diverse software versions, independently developed for the same specification, would fail independently [11], a claim that if true would certainly support the usefulness of diversity. In this experiment, 27 program versions, required to recognize in a set of radar echoes the presence or absence of an incoming missile (the "launch interceptor" problem), were developed independently (9 versions developed at the University of Virginia (UVA) and 18 at the University of California at Irvine (UCI)), and then tested on 1 million random test cases. The test results refuted with high confidence the conjecture that *all* software versions failed independently. Our calculations below use the published data [11].

If we look at how these results are interpreted by those quoting them, we see multiple viewpoints:

- many (simplistically and wrongly) believe that the experiment proved software diversity useless, e.g.: "N-version programming rests on the assumption that software bugs in independently-implemented programs are random, statistically-uncorrelated events. Otherwise, multiple versions are not effective at detecting errors [...] John Knight and Nancy Leveson famously debunked this assumption on which N-version programming rested" [http://leepike.wordpress.com/2009/04/27/n-version-programming-for-the-nth-time/];
- some point out that the reported results showed the average frequency of common failures of two versions was about 60 times less than that of an individual version, a massive improvement, and roughly in line with measures in some other experiments;
- some tried to extrapolate to the probability of masking all faults, discussing which specific features of the ''launch interceptor'' problem could enhance or reduce the benefits of diversity [16]; or reasoned about likely effects of diversity given more or less reliable versions, with a model based on the fault density of the versions [17].

We will *not* use the data to claim any general property of diversity, but to illustrate general mathematical facts, and some possibly surprising consequences, on a real example.

A. *Generalizations about pfd*

As Knight and Leveson themselves pointed out, all generalizations from a single experiment, or few experiments, are suspicious. First, one may doubt whether the conditions of the experiments are representative of those of the projects for which predictions are sought. Secondly, there is no theory for deciding *what* we should generalize, if we wished to do so. We are in the same situation as an aeronautical engineer would be who tried to predict an aircraft's performance from wind tunnel tests on small scale models, without a theory about how the results scale up. For instance, in this experiment the average *pfd* of a version[3] was 7.02×10^{-4} (4.33×10^{-4} for the UVA subset and 8.37×10^{-4} for the UCI subset); for pairs formed from one version from each subset, the average *pfd* was 1.09×10^{-5}. This could be seen as "30 times worse than independence" (i.e., than the product of the average *pfd*s of the two subsets), or "64 times better than the average for a single version". Such ratios are sometimes cited as indicative of the likely results of applying N-version programming; but they really only show that such results *can* be achieved (because they *were*, in the experiment). There is no theory to justify using them as a *likely* prediction, even as order-of-magnitude guidance. They certainly cannot both be given this status (be considered invariant characteristics of the technique), because this would be self-contradictory: the first makes the expected system *pfd* proportional to

[3] When referring to measures of *pfd*s in this experiment, we will use the "observed" *pfd* values, i.e., empirical frequencies of failures, unless otherwise noted. In *assessing* a safety-critical system, instead, one would want a pessimistic confidence bound. We will highlight which steps of reasoning are affected by choosing one or the other of these measures.

the product of the versions' expected *pfd*s; the second to their average. If, as a thought experiment, we improve the processes producing the two versions in a system so that both have expected *pfd* 100 times better than the two subsets in this experiment, the expected *pfd* of the system would become 10^{-9} based on the first assumed invariant, or 10^{-7}, based on the second. There is no clear theory to justify a choice between the two, or indeed a belief that either is the sought-for invariant.[4] All experiments have shown diversity to improve average reliability in 1ooN systems. This *is* an invariant, because it can be mathematically proved to be, without empirical support.

B. *Measures of risk and of risk reduction*

However, in interpreting the results of experiments, we should also ask whether the average *pfd* is indeed the measure of interest. We are interested in how the decision whether to apply diversity affects the (dependability-related) risk accepted in developing the system. We now discuss how the undesired event to be avoided differs between different application scenarios: its probability is what diversity mainly attempts to reduce, and thus the way we assess the risk reduction that can be achieved must also change between these scenarios. We reason about three types of undesired event: failures; the event of having a least one failure in a mission or operational lifetime; and the production of a system with *pfd* exceeding a required upper bound.

The real-life random process in which the risk is incurred is as follows: if a 1oo2 system is needed, two versions are developed or bought; they are combined into a system; the system is then operated, receiving a sequence of random demands, with probabilities defined by the operational profile under which it operates. Given this operational profile, the system has a certain *pfd*.

The results of the development – the specific two programs developed, their *pfd*s and the *pfd* of the resulting 1oo2 system – would vary from case to case, according to some probability distribution; therefore, in software engineering experiments numerous versions are developed, to obtain a sample from this distribution. The statistics of this sample are used to estimate the probabilities of various outcomes when developing a single 1oo2 system.

C. *Measures of risk: expected pfd*

In the experiment just outlined, the mean *pfd* of a 1oo2 system from the population of versions produced estimates the probability of failure in a real-life "experiment" consisting of developing *two* versions ("randomly sampled" from the probability distributions describing the development processes for the two versions) and then testing the resulting 1oo2 system on a single demand ("randomly sampled" from the operational profile assumed in the experiment). The mean *pfd* observed in an experiment with many versions produced, multiplied by a future number of demands, predicts the *expected* number of failures in operating the system over that many demands. If we do expect the system *to fail a number of times in its lifetime*, and given a cost per failure, the mean *pfd* determines the expected cumulative cost of failures (*if* faults are not corrected after each failure; *cf* [18] for a more general treatment). In this limited sense, the reduction in average *pfd* between the set of versions and the set of 1oo2 systems (built from these versions) is an indication of risk reduction – in the KL data, by a factor of about 60.

D. *Measures of risk: reliability*

However, diversity is typically used for safety critical systems with requirements like "[catastrophic failures must be] not anticipated to occur during the entire operational life of all aircraft of one type" [19]. Then, what really matters is the probability of surviving for the intended operational life without such failures – formally, a reliability function. How did diversity improve this in the KL experiment?

---

[4] Readers may object that this weakness is common to most extrapolation from experiments in software engineering. We agree.

Referring again to the real-life random process in which the risk is incurred, the risk that has to be controlled is that the *one* system developed fail in its lifetime. The systems that may be produced may exhibit a broad range of *pfd* values, from some that will fail almost certainly to some that will almost certainly *not* fail. To obtain the probability of no failures over the system's life, we need to extend the thought experiment of the previous section to operating each system through future demands. If the *i*-th system, in the set of all possible systems, has *pfd* q_i and probability p_i of being developed, the probability of this *i-th* possible system surviving *T* demands (its reliability function) is

$$R_i(T) = (1 - q_i)^T \qquad (1)$$

The risk from the uncertain process of developing and then using a system is then the risk of the randomly selected system actually failing

$$1 - R(T) = 1 - \sum_i p_i (1 - q_i)^T \qquad (2)$$

This calculation for the KL data set, taking the sample as representing the whole population of possible versions, is shown in Figure 1. The solid lines represent the probability that a version, or a 1oo2 pair, randomly chosen from this set, operates without failures for T consecutive demands. In other words, they represent the average of the reliability functions of all individual versions. We also show that averaging in the wrong order would be misleading: the dashed lines represent the reliability that one would – wrongly – predict by assuming the average *pfd*: 7.02×10^{-4} for single-version systems and 1.09×10^{-5} for diverse 1oo2 systems.

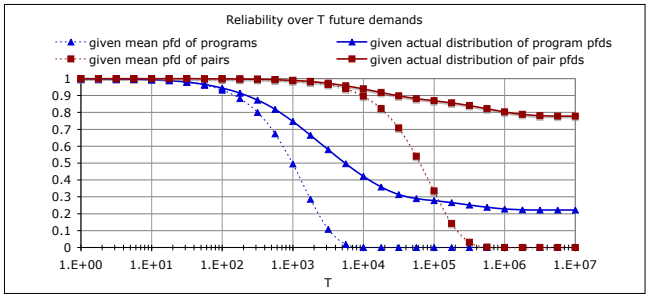


Figure 1 Reliability curves (solid lines) for the set of programs developed in the Knigth-Leveson experiment. The dashed lines instead represent the reliability of a (non-existent) "average" version or "average" 1oo2 system.

These curves illustrate some important *mathematical* facts, and provide a few *empirical* observations:

- each version and system that *could* be developed has in principle different *pfd*. A reliability curve here does not represent the unknown reliability of the one system developed; it represents the gamble taken in developing a system through a process with inherently variable results, and then using this system.[5] The graph compares this risk when developing a two-version system with the risk when developing a single-version system from the same development process;

[5] The distribution that matters is that of the *pfd* of programs that reach the operational stage, that is, takes into account the stringent V&V process, with possible rejections and modifications, that precedes acceptance into service. In experiments this is typically simulated (and was in the KL experiment) by a preliminary acceptance testing phase for each version. A real-life developer has also the extra risk of producing a "dud" system that will not even pass the acceptance phase into operation, a generally low risk for specialized developers. This economic risk actually *increases* with diversity, everything else being equal; for the 1oo2 system to fail in operation, it is necessary that *both* versions fail, hence its advantage over a non-diverse system; but for it to be accepted, in many regulatory regimes it would be necessary that both versions *pass*. Using diversity trades a reduction in safety risk against a – possibly mild– increase in project (financial) risk.

- given a system for which there is uncertainty about its *pfd*, assuming the average *pfd* (as though the actual system were a hypothetical, and non-existent, "average" system) always yields pessimistic reliability predictions [20], as shown here by the dashed lines;
- indeed for the KL data, in the long run (for T tending to infinity), the hypothetical version or version pair with average *pfd* (dashed lines in Figure 1) *will* fail, with probability 1; in reality (solid lines in the figure), the probability of a randomly chosen version ever failing is 0.78, while that of a randomly chosen pair ever failing is 0.22[6]: diversity increases the probability of never experiencing failures at all on a long system lifetime from (1-0.78) to (1-0.22), or, equivalently, reduces the risk of failures by a factor (0.78/0.22)=3.5;
- the average reliability over a set of programs (or a probability distribution of the *pfd*) is heavily influenced by the fraction of programs that have low enough *pfd* that they are very unlikely to fail over the number of demands of interest (and by those that are almost certain to fail): the tails of the distribution of system *pfd* matter;
- the *pfd* of a 1oo2 system can be no greater than that of any of its components, as illustrated in Figure 2;
- so, given any required upper bound on *pfd*, the probability of achieving it is greater for a 1oo2 (or, in general, 1ooN) system than for any of its component versions (*cf* Figure 3). Given any distribution of *pfd* for the individual versions, the distribution of *pfd* of 1oo2 systems obtained from them will be "compressed" towards lower values;
- in particular, the worst-case probability of the *pfd* of a 1oo2 pair of *independently* developed versions exceeding a given value is the *product* of the probabilities of this value being exceeded by the two versions. Specifically, calling F the cumulative distribution function of a *pfd*, we have for a 1oo2 system built from versions A and B:

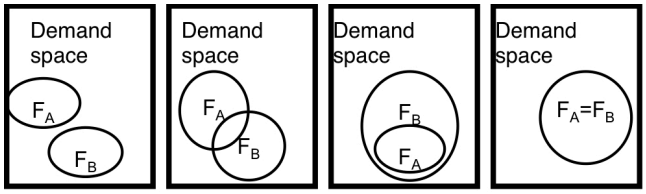
$$F_{1oo2}(q) \geq 1 - \big(1 - F_A(q)\big)\big(1 - F_B(q)\big) \qquad (3)$$

  Intuitively, this equation says that any version with high *pfd* has a good chance of being paired with a version with lower *pfd*. *In particular*, any version with *unusually high pfd* (say a *pfd* that occurs with a small probability ε) is very likely to be paired with a lower-*pfd* version (since these occur with much higher probability, 1 - ε): any "thin, long tail" of high *pfd* values that is present in the distribution of *pfd* of individual programs will be much smaller in the distribution of *pfd* of 1oo2 systems. In (3), the limiting case of equality gives a worst-case, *minimum assured* improvement in risk. This worst-case improvement level due to diversity is illustrated by dashed lines in Figure 3[7]

---

[6] Readers may notice that these two numbers add up to 1. This is just a coincidence. Out of 27 versions, 21 (7/9, or 0.777...) had observed *pfd*>0. The number of version pairs with observed *pfd*>0 happened to be 36 out of 162 pairs (2/9). But as we discuss later, this latter fraction could have been anywhere between 0 and the product of the corresponding fractions of failing versions – 6 out of 9 UVA versions and 15 out of 18 UCI versions – for the two sets from which the versions were selected independently to form pairs: so, between 0 and (6/9)(15/18)=5/9.

[7] That diversity guarantees improved *pfd* is true for all 1-out-of-N systems; the larger the value of *N*, the greater the reduction in probability of high *pfd*. For a voted system, this is only true if the probability of a version exceeding that *pfd* is below a threshold: e.g., for a 2-out-of-3 system where all versions' *pfd*s have the same distribution, this threshold is 0.5.



a. Mutually exclusive failures, pfd_{pair}=0. Failures are negatively correlated.

b. Failure sets overlap. Failures may be negatively or positively correlated, or independent.

c. If A fails, then B fails as well. Failures are positively correlated.

d. Identical failure sets: A and B fail on the same demands. Failures are positively correlated.

Figure 2 Possible relationships between non-null failure sets, F_A and F_B, of two versions A and B. A fifth case (e), in which F_A and/or F_B is the empty set, turns up frequently regarding *observed* failures in experiments.

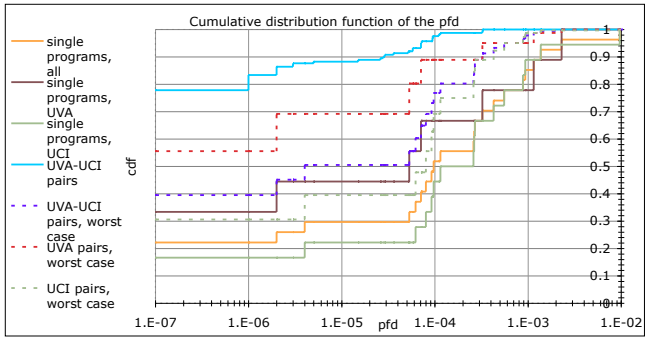


Figure 3 Cumulative distribution functions (*cdf*s) of the observed version *pfd*s in the KL experiment. The dashed lines are pessimistic bounds for the *cdf*s of *pfd*s of 1oo2 systems obtained from these versions (eq. (3)), both when combining two versions from the same "process" (university), and when "forcing" diversity by choosing one from each. The top solid line represents the distribution of 1oo2 *pfd*s actually observed in the experiment. All observed *pfd* values to the left of 10^{-6} are 0.

- this worst case assumes that in *every possible pair* of versions, whenever the better version fails, the other one also fails on the same demand: the "unlucky" cases (c) or (d) in Figure 2. In reality, one hopes often to observe cases (a) or (b), which would yield higher reliability for the pair. *Empirically* in the KL experiment, for inter-university pairs the distribution of *pfd* was as the top solid line in Figure 3, indeed much better than the worst-case curve, also shown;
- with this measure of risk, the risk reduction achieved varies with the "projected lifetime" (here measured in number of demands). The reduction observed in this experiment is by a factor of 64 for one demand, as mentioned, of 50 over 100 demands, 10 over 10,000 demands, and tends to 3.5 at infinity (Figure 1);
- another factor to consider is "forced" diversity. The KL data concern 1oo2 systems made from "heterogeneous" pairs (one version produced at UCI and one at UVA). This is a very weak form of "forced" diversity. In industrial practice, much stronger measures are applied to "force" diversity, hoping to achieve lower probability of common failures [4]). We will study the role of forced diversity in Section III.

We briefly discuss some assumptions made above:

1) Role of probability of fault-freeness or very low pfd

In Figure 1, the reliability tends to a non-zero asymptote due to a non-zero probability of the *pfd* being 0. Some may question the plausibility of *pfd*=0: software, they would maintain, *always* has bugs. But what matters here is simply having versions with low enough *pfd* to imply probability close to 0 of failures over the "projected lifetime" (number of demands here; time, for continuously operating systems) of interest. We could call these "effectively fault-free" with reference to this "projected lifetime". For instance, if here we attributed to those versions that never failed a *pfd* that equals not the observed

frequency of failure, 0, but a one-sided 95% confidence bound on this probability, 3×10^{-6}, then for what concerns the first few thousand demands, these versions would be "effectively fault-free": Figure 1's reliability function would not change significantly.[8]

Achieving such low *pfd* values with non-negligible probability is a plausible scenario for systems that are made simple by construction for the sake of safety, and developed according to stringent standards, and if one considers only those faults that are likely to endanger safety. Various authors have studied how to assess and exploit the probability of *pfd*=0 [21 , 22-25] towards demonstrating extremely high reliability for critical applications.

If such low *pfd* values can be achieved with reasonable probabilities, it is then an important fact that the probability of achieving them in a diverse system is substantially higher than in the individual versions. E.g., if the requirement "not anticipated to occur during the entire operational life" is formalized as "the probability of occurrence over the operational life must not exceed 10%", one way of ensuring this is to have at least 90% probability of a *pfd* equal to 0 or so low that it ensures practically certain survival over the operational life of the system. This goal may be challenging; however, if the system can be developed as a 1oo2 system with independently developed versions, that goal can be attained if we achieve 68% in each version; or if we achieve 46% per version in a 1oo3 system; or 80% in a voted, 2-out-of-3 system; etc.[9] These targets may or may not be attainable, but they are more so than the initial 90%.

2) Independence between pfds

While we believe that failures of two faulty versions are typically not independent, we assumed above that the *pfd*s achieved in developing two versions are independent random variables; a scenario of *independent developments*. Studying diversity has made us wary of blanket assumptions of independence: we should examine this one carefully. "*Independent developments*" means that if we consider, for instance, the project that develops one of the two specific versions that will be used in this specific system – say, project A –the probability of this project achieving a certain version *pfd* is the same irrespective of exactly what value of *pfd* is achieved by project B, developing another version. This seems reasonable when both projects are in the past and we are buying from independent vendors that can prove the two products to have separate histories. With bespoke procurement, still if the developments are carefully isolated, the assumption would be easily believable with respect to the parts of the process that can be kept isolated (for instance, this independence would only be believable *conditional* on a specific set of high-level requirements[10] [15]).

To conclude the observations on this example, we cover two more viewpoints from which a system can be assessed.

E. Measures of risk: risk of exceeding a required pfd

In many applications, a reliability or safety requirement is stated simply as a *pfd* level that must not be exceeded. It makes sense to ask what this requirement means, in view of the uncertainty that affects any assessment [26]. Supposing that the requirement is for a *pfd* not exceeding 10^{-5}, does it mean that the *mean pfd* should be 10^{-5} or lower? Or is this

---

[8] This "few thousand demands" limit does *not* mean that these versions *will* later fail: they *could* have *pfd*=0. It is just an acknowledgment of the limits of statistical inference from any finite experiment. If fault-free versions exist, they will indeed never fail. If they existed in a set of programs subjected to statistical testing on a finite number N of tests, we could never infer from testing that they are actually fault-free with certainty. We could however conclude that their *pfd* is so close to 0 that for predicting reliability over some number N'<N of future demands, they can be assumed 0 *without substantial error*. For a more complete study of "effective fault-freeness" see [21].

[9] These numbers are obtained by solving the familiar formulas for the probability of failure of redundant systems with independent failures, applied to independent *processes* of version development rather than to the versions themselves. For the three scenarios listed, if the probability of a single version being too unreliable is p_V, the probabilities of the system produced being too unreliable are respectively p_V^2, p_V^3, $(3p_V^2-2p_V^3)$.

[10] This independence is conditional on these requirements being *set*, not on their being *correct*. [15].

interpreted as a (Bayesian) confidence bound – "the *pfd* should be 10^{-5} or lower with high probability" – and if so, what is this required high probability: 90%, 95%?

For the KL data, the improvement in mean *pfd* has been quoted before; the effect of diversity on the probability of satisfying a certain bound on system *pfd* is fully described by the *cdf*s in the above plots; the gain varies with the bound chosen. This is evident in Figure 4, showing the reduction in the risk of producing a system that is too unreliable.[11] .

F. *Probability of failure independence*

Independence between failures of diverse versions is still frequently discussed or sometimes claimed. Indeed, being able to claim independence of failures would be extremely useful in *assessment* of a system: after paying for the separate assessment of the *pfd*s of the diverse versions in it, the system *pfd* would come at no extra cost as a simple function of these *pfd*s, and would be very good. But since independence is impossible to believe *a priori*, discussing the chances of achieving it is just like discussing the chances of achieving any other, arbitrarily chosen value of *pfd*. What matters is the *pfd* of the system delivered, not whether it is more or less than this arbitrary point. E.g., in the KL data, a mean 1oo2 *pfd* "30 times worse than independence" is a substantial improvement over a single version; in a system of two versions with $pfd=10^{-6}$, it would deliver a quasi-unbelievable system *pfd* of 3×10^{-11}. Lack of independence is irrelevant.

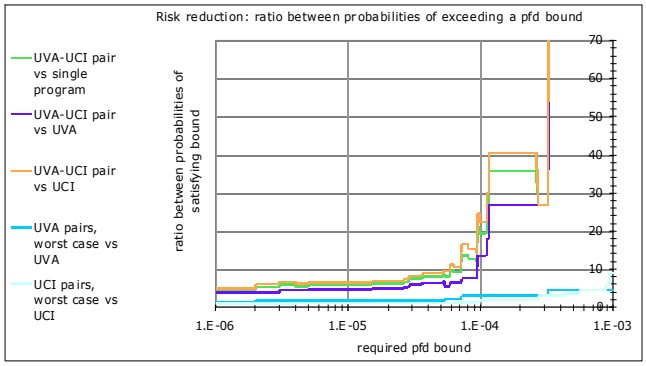


Figure 4 Reduction in risk of violating a requirement on *pfd*, in the KL data. Some curves shoot to infinity towards the right end of the plot because in this data sample no 1oo2 pair had *pfd*>0.000323: for required bounds greater than this value, the probability of failing to satisfy them is 0.

However, if we *did know* what relationship to expect between the average true system *pfd* and that which would follow from failure independence, this knowledge would be useful. In the KL data, the former was much higher than the latter.

A reasonable position, in view of the probabilistic models [14], is that *a conservative assessor has no choice but to assume positive correlation* between failures of the versions in a 1oo2 system. However, this does not mean that *positive correlation is the norm* or that achieving, or doing better than, independence of failures will be rare. In fact, in the KL data, "independence or better" between failures of two versions occurs often. The observed frequencies are in Table I. *Only 2/9 of the pairs had positive observed correlation* between failures[12]. All cases of negative correlation had *disjoint* failure sets: the two versions never failed together. These data show how misleading it may be to

---

[11] One can choose to describe the reduction in risk of failure or the increase in chance of success. We show the former because it is the usual way of reasoning about small risks. Of course describing the increase in chance of success would show smaller increases. Reducing a probability of failure from – say 0.1 by a factor of 10^6 only increases the chances of success by a factor of roughly *1.1*, from 0.9 to 0.999999.

[12] We note again that this table refers to observed *frequencies* of failures, as estimators of probabilities. Again, we are not doing hypothesis testing on *probabilities* of common failure: as some versions have individual failure rates in the order of 10^{-5}, an experiment with 1 million test cases is still too small to discriminate with any confidence, if two such versions exhibit no common failures, between the hypotheses that the *pfd* for common failures is less or more than 10^{-10} (better or worse than independence).

reason on the basis of the means alone. Of course, we *do not* recommend that one assume similar frequencies to apply in general.

TABLE I FAILURE CORRELATION IN 1OO2 PAIRS FROM KL EXPERIMENT

class of failure correlation	number of pairs in class	%
independent	72	44.44%
positively correlated	36	22.22%
negatively correlated	54	33.33%

III. EFFECTS OF FORCED DIVERSITY

We now study further, from the viewpoint of reducing risk, the common notion of the desirability of "forcing" diversity, i.e., choosing products with different designs, produced with different development methods and tools, rather than just products that were developed independently ("unforced" diversity in our terminology). The advantages one can hope to achieve by forcing diversity have two aspects:

- the diversity of process and designs may well have the result that the errors most likely to be made in developing the versions will be different; that these will cause faults that affect different demands; and thus that (with reference to Figure 2) cases (c) or (d) are less probable than cases (a) and (b), and within case (b), the intersection of the two failure sets is likely to be smaller (all in comparison with what these probabilities would be with "unforced" diversity). However all this is a *plausible* (possibly common) scenario of advantages; perhaps instead, in a specific project, the differences that we "force" have no effect on which errors are likely at all; or perhaps there is only one error that in the production of either version is likely enough to have an effect on risk, and scenarios (c) or (d) are the norm;
- more subtly, equation (3) indicates that even in the worst case just highlighted, there is a guaranteed worst-case level of stochastic improvement. This is present irrespective of whether the diversity is "forced" or "unforced". That is, whether the distributions $F_A(q)$ and $F_B(q)$ are identical or different does not affect the existence of this guaranteed advantage of a two-version system over the a single-version system; but we can show that *it affects its size*.

We now first consider this *worst-case* improvement, from equation (3), in the chance of achieving a certain *pfd*. This matters for both the measures of risk considered in sections II.D and II.E, the former in view of how the probability of "effectively fault free" systems affects the lifetime risk of failure. We prove that forced diversity is a remarkably effective strategy over a broad range of scenarios. We then discuss what can be said about risk reductions outside the worst-case scenario.

A. *Forced diversity, worst-case distribution and probability of fault-freeness*

We refer to previously described models of multiple-version programming [12 , 13]: two versions are sampled independently, from a "population of all possible versions", according to a single probability distribution, in the case of "unforced" diversity, but according to two distributions, produced by the development processes A and B, in the case of "forced" diversity.

As pointed out earlier, the probability of a 1oo2 system, with *independent* developments of the component versions, exceeding any given bound on *pfd* is *at worst* (i.e., at most) the product of the corresponding probabilities for the two processes (cf. equation (3) and cases c and d in Figure 2).

With "forced" diversity, these two probability distributions (normally unknown, when one makes the choice between forced and unforced diversity) are different. Forced diversity is known to improve *mean pfd* under certain conditions of *indifference* between processes A and B [13]. We ask whether, without this assumption, it improves the probability of satisfying a *bound* on *pfd*. In view of the importance of "fault-free" or "effectively fault-free" systems, we study in what follows the probability of not exceeding

the bound *pfd*=0: "fault-free" pairs, i.e., 1oo2 pairs in which *at least one* member is fault-free. However, the proofs *do not* depend on which *pfd* bound one considers.

We first introduce our notation. Without a policy of forced diversity, developers will have their own preferences and thus probabilities of choosing either process. We call α this "unforced" probability of developers choosing process A, so their probability of choosing process B will be (1-α).[13] We call q_A the probability of process A producing a faulty program, i.e., a program with *pfd*>0, q_B the same probability for process B, and for convenience parameterize by setting:

$$q_A = kq_B \qquad (4)$$

Without loss of generality, we assume $k \in [0,1]$: B is the "worse" process. The decision maker does not know *k*, which describes how well the two processes A and B perform on the current development; and α is also often unknown. With forced diversity, the probability of a randomly chosen pair being faulty is:

$$q_f = q_A q_B = kq_B^2 \qquad (5)$$

while with unforced diversity, the probability of a randomly chosen pair being faulty is the probability of selecting a faulty program, squared:

$$q_u = (\alpha q_A + (1-\alpha)q_B)^2 = q_B^2(\alpha k + 1 - \alpha)^2 \qquad (6)$$

Forced diversity guarantees better worst-case probability of fault-freeness than unforced diversity **iff** $q_f < q_u$. One can note that:

- if process A only produces fault-free programs (*k=0*), then q_f=0, and thus unforced diversity can only be as good as forced diversity if developers *always* choose the "perfect" process A (α=1).
- If processes A and B give the same probability of fault-free programs (*k=1*), then unforced and forced diversity give the same probability of faulty pairs.
- In the general case, the decision maker may not know α or *k* and may be concerned that forcing diversity might make things worse. Indeed, if developers - left free to choose - tended to choose the "better" process A over process B often enough, the resulting frequency of pairs with at least one A program might guarantee a higher frequency of fault-free pairs than guaranteed by forced diversity, i.e., by requiring each pair to contain one A and one B program. Of course, for any given $\alpha < 1$, we have seen that as $k \to 0$, forced diversity will eventually become the better choice.

We measure the advantage of forced diversity via the ratio q_u/q_f (so that a larger ratio indicates a larger advantage). Figure 5 shows plots of q_u/q_f for a range of scenarios, which illustrate how forced diversity is a guarantee against surprises in the quality of one of the processes used.

The more process B is "worse" than process A (the smaller *k* is), the more the gain from ensuring – by "forcing" diversity – that every pair contains an "A" version. The curves can be read as follows: if "unforced" developers have a 50-50 chance of choosing the process with the higher probability of faulty programs (process B), then forcing diversity *always* has an advantage over not forcing it; even more so if developers tend to choose the worse process, B, more frequently than the better process, A.

---

[13] When selecting off-the-shelf products, "unforced" diversity amounts to choosing randomly among all those that appear to satisfy the requirements (functional requirements, required safety or quality certification, etc). If *e.g.* 10 such products are available, 7 of which were produced by process A, then if the customer chooses with uniform distribution, α=0.7; forced diversity means choosing separately one product from the 7 type A products, and another one from the 3 type B products.

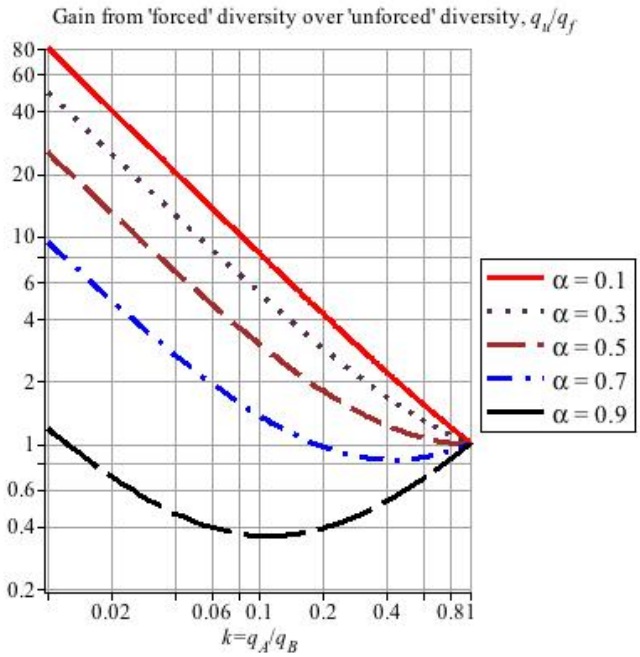


Figure 5**.** Ratio between worst-case probabilities of non-fault-free systems with unforced (q_u) and forced (q_f) diversity, for different values of i) the probability α of developers choosing the development process, A, that is better at producing fault-free programs; ii) the ratio k between the probabilities of faulty programs from process A and from process B. Points *above* the horizontal line q_u/q_f=1 indicate situations in which forced diversit gives better worst-case probability than unforced diversity.

The plots also show that *even if developers "favour" the better process, A* (that is, if α > 0.5), forcing diversity has potentially large advantages, for ranges of values of α and k (or comparatively moderate disadvantages for other ranges). This is because if k is small – the "worse" process B is substantially worse than the "better" process A – then even a small risk $(1-\alpha)^2$ of ending up with a pair of versions from process B makes $q_u/q_f>1$: unforced diversity incurs a greater worst-case risk (of producing a system with *pfd*>0) than forcing every pair to contain one version from the "better" process and one from the "worse" process. The worse B is, the higher α must be for unforced diversity to "beat" forced diversity from this viewpoint.

In other words, forced diversity tolerates undesired outcomes – choices of development process or design that happen to perform badly on a specific project – that we cannot predict and avoid. It is true that this protection has a possible "flip side": *if we knew* that developers "wisely" tend to prefer process A, that produces fewer faulty versions, then *not* forcing diversity could be advantageous; but we would need to know that developers' preferences are commensurate to the actual difference between the two processes. The required preference level is obtained by solving the equation

$$q_u = q_f \qquad (7)$$

with the results that unforced diversity yields better worst-case probability of fault-free pairs than forced diversity *if* developers choose the "better" process A with probability:

$$\alpha > \frac{-1+\sqrt{k}}{k-1} = \left(1+\sqrt{k}\right)^{-1} = \left(1+\sqrt{q_A/q_B}\right)^{-1} \qquad (8)$$

For instance, if process B produces faulty versions with probability *10 times lower* than process A ($k= q_A/q_B$=0.1), Figure 5 shows that for α=0.7, forced diversity is still the better choice ($q_u/q_f>1$). That is, if developers choose the better process A 70% of the time, this is *still not enough* to make unforced diversity superior; from (8), we can calculate that for q_A/q_B=0.1, the minimum required "preference" for process A is α=0.76. Another way of reading the equations and plots is: if developers use the "better" process A with probability

α=0.7, we see that the curve labelled α=0.7 reaches the value 1 for k=0.2. This tells us that α=0.7 only makes "unforced" diversity superior (in terms of worst-case risk of system *pfd*>0) *if* process B is no more than "5 times worse" than process A: $q_B < 5\ q_A$.

We also note the asymmetry of the consequences of developers "preferring" one or the other process: strong preferences for A, the "better" process, make the worst-case for forced diversity mildly worse than for unforced diversity (see the minima on the curves for α<0.5); strong preferences for B, the "worse" process, make the worst case for unforced diversity radically worse than for forced diversity. So, the desirability of forcing diversity (from this viewpoint of worst-case improvement in the risk of producing a faulty system) seems a remarkably robust property, over a broad range of situations (the values of α and k, generally unknown). Of course, in the extreme case α=1 (any system developed contains two versions from the "better" process A, unless we "force" each to include one version from process B), $q_u/q_f = 1/k < 1$. We discuss this scenario in the next section.

Last, we recall that these results apply for *any* bound on *pfd*, that is, not only for the case of fault-freeness or "effective fault freeness" for a given lifetime (or mission) number of demands, but for the whole curve of worst-case *cdf*. In other words, these results hold if by q_A, q_B, q_u and q_f we designate not probabilities of certain *pfd*s not exceeding 0, but probabilities of their not exceeding any specific bound of interest (for instance, one required as the maximum acceptable *pfd* for an application). Recall that this is a worst case in that the advantage proved for forced diversity if inequality (8) holds will apply, even if the intended main advantage of forced diversity – that of "diversifying faults", making cases (c) and (d) in Figure 2 less likely than with "unforced" diversity – failed to materialize in a specific project. If inequality (8) holds, for the probabilities q_A and q_B of the two processes producing versions that exceed the required bound on *pfd*, all the equations and graphs in this section apply to the worst-case probability of diverse pairs of versions exceeding that bound, and "forcing" diversity is still the preferred choice unless the developers "preferred" the "better" process with the probability given by (8).

B. *Departures from the worst case*

The discussion above concerns how forcing diversity affects a *pessimistic* bound on the probability of satisfying any given requirement on the *pfd* (and therefore also pessimistic bounds on lifetime reliability). The actual *pfd* achieved cannot be worse, but may often be better than this: the pessimistic bound assumes case (c) or (d) in Figure 2, but the goal pursued via "forced diversity" is to avoid common failure points as far as possible, making case (a), or at worst (b), the likely outcomes (*if* both versions have faults). Indeed, in the KL data, for instance, 1/3 of the inter-university pairs (or 3/5 of those with two faulty programs) exhibit behavior (a), and this produces the top *cdf* curve in Figure 3, substantially better than the worst case.

So, a conclusion about forced diversity is that if it were to achieve its primary goal (making case (a) of Figure 2 the most likely case) it is to be preferred to unforced diversity. If it failed in this primary goal, it would still deliver, as shown in the previous section, protection against the risk that one of the two processes happens to be especially ineffective at achieving low *pfd* for this project. The system designer should always, it seems, choose to force diversity.

A dilemma arises in the special case of knowing (or, more realistically, having strong evidence about) which process has the lower risk of high version *pfd* – which one is process A. Should one still choose to combine a version from the "better" process with one from a "worse" process? This gives a hope of high probability of case (a), disjoint failure sets. By contrast, combining instead two versions from process A (we can see this as "unforced diversity" with α=1) would reduce by a factor k the worst-case risk of a high *pfd*: a tempting, seemingly risk-averse choice if one is unsure about the effectiveness of the ways of "forcing" diversity available for the current project, and suspected that k may be small.

We have no mathematical theory to predict the magnitude of the departure from the worst case. The insight from probabilistic models [13 , 27] is simply that it depends on

whether the different processes share some area of "high-difficulty" (and frequently occurring) demands for the application being developed.

"Unforced" diversity data for the KL experiment are not published; we give just one example from another, recent set of empirical data [28] from programs developed in an open programming context.

TABLE II COUNTS OF FAILURE-FREE AND FAULTY PROGRAMS (WITH *PFD* ≤ 0.01) AND 1OO2 PAIRS FOR THE *FACTOVISORS* PROBLEM:
COMPARING "FORCED" AND "UNFORCED" DIVERSITY

				unforced	forced
	C	Pascal	*1oo2 pairs made up of a failure-free and a faulty program*	9,504	1,686
Failure-free single programs	172	26	*1oo2 pairs made up of two failure-free programs*	19,701	4,472
Faulty single programs	45	3	*1oo2 pairs made up of two faulty programs*	1,176	135
Total count of single programs	**217**	**29**	***Total count of 1oo2 pairs***	**30,381**	**6,293**

One of the many scenarios analyzed concerns diversity between programs in C and Pascal that solve a mathematical problem called *Factovisors*. Table II gives data about failure-free and faulty single programs (among those with $pfd \leq 0.01$). Here, $\alpha=0.118$, and $k=0.499$; these values make the advantage of forced diversity $q_u/q_f = 1.8044$ (*cf* the topmost curve in Figure 5). In detail, the pessimistic bound analysis (III.A) gives *worst-case probabilities* of a randomly chosen pair being faulty, given unforced and given forced diversity, $q_u=0.0387$ (1,176 pairs out of 30,381), and $q_f=0.0215$ (135/6,293). The *actual*, lower observed values were: $q_u=0.0232$ (705/30,381); $q_f=0.0162$ (102/6,293) because only a fraction of 1oo2 pairs made of two faulty programs failed themselves (Table II). One may note that while the ratio of the worst-case values would be $q_u/q_f = 1.8044$, that of the *observed* values gives a smaller gain, 1.4317: it so happened that unforced diversity resulted in a higher fraction of failure-free 1oo2 pairs made up of faulty programs (case (a) from Figure 2):~40% (471/1,176), than the corresponding value with forced diversity: ~24% (33/135).

IV. DISCUSSION: EFFECTS OF THE DISTRIBUTION OF *PFD*

The example data that we used show but one of a broad range of possible scenarios. We briefly discuss alternative scenarios about the distribution of the *pfd* values of single versions and pairs. Any such distribution will be a set of discrete probability masses rather than a continuous distribution, because there are only a finite number of possible demands (any possible demand is a long digital number of finite length, because for any digital system the number of input bits, the memory size and the lifetime are finite). In practice this distribution might well approximate a continuous one, or, at the other extreme, be reduced to a few discrete probability masses. If the distribution is very sparse, the effects of diversity may well be especially dramatic. If we think of the process that generates faults in each version and thus this probability distribution, we can identify a range of possibilities between two extreme scenarios [27]:

- *complex, good quality product* scenario: many faults are possible, with low but non-negligible probability of being present, each with a modest contribution to *pfd*. Then, both versions are almost certain to be faulty, but it is unlikely that they share many faults. This scenario is plausible for complex, mature, commercial quality software, and some evidence for it has been observed in off-the-shelf DBMS products, and in operating systems from the viewpoint of security flaws [29, 30]. Thanks to the central limit theorem, the distribution of *pfd* for these products would resemble a continuous, bell-shaped, unimodal distribution, and the *pfd* for 1oo2 systems made from them would be similar, but narrower and shifted towards

pfd=0. With "forced diversity" (e.g. if the two versions belong to different product families from different vendors) it is possible that although many likely faults are common to the two versions, many are not, and it may well happen that $E(q_{pair})<(E(q_{single}))^2$;

- *simple, likely ultra-reliable product* scenario: very high quality production processes make fault-freeness achievable with non-negligible probability; very few faults are practically possible. The distribution of *pfd* for the versions produced is made up of very few discrete probability masses (one of them for *pfd*=0). Combining two such versions produces another discrete distribution, and what shape this distribution happens to take is unpredictable: it depends on the effects of the individual faults on *pfd* and how many of these "least unlikely" faults are the same for the two versions.

V. CONCLUSIONS

If we accept that software reliability techniques are a form of protection against the variability of the software production process – a way of controlling the risk of an individual development resulting in an inadequate product – we need to assess their effect on the distribution of results achieved. We have moved a first step in this direction, regarding software diversity.

Decisions about whether to use, and how to pursue, diversity are generally driven by consensus within an industrial sector, or by judgment, or educated intuition. A proper description of how the various measures one may consider are related to the risk measure of interest will help to shape this judgment and intuition. To this end, we have shown a number of mathematical results, and some empirical facts, that go beyond, and often contradict, frequently voiced intuition.

Our summary of this paper's contributions and conclusions includes:

- different application require different measures of development risk, which imply quantitatively different benefits from diversity. Identifying this range of different risk measures, appropriate for different scenarios of use, is useful and in particular can avoid some wrong extrapolations from published results of experiment as guidance to achievable risk reduction;
- extrapolation from experimental results must be taken with extreme caution, but *some* experimental results are actually instances of mathematical truths that one can trust to hold *without empirical demonstration* whenever certain sufficient conditions hold;
- when the requirement is a low probability of a system *ever* failing, the measure of interest is a reliability function rather than a reliability parameter (*pfd*);
- we have shown some aspects of how diversity reduces risk. Among these, the probability of fault-free or "effectively fault-free" programs or pairs may have great importance towards the likely reliability of the system; and diversity radically "shrinks" the low-reliability tails of the distribution of system *pfd*;
- "forcing" diversity is a more robust strategy than usually acknowledged. Its primary goal is "diversifying faults" (making it likely that any failures of the diverse versions in a system are mutually exclusive, as in Figure 2a). Even in the worst-case scenario that it does not succeed in this, it improves the chances of meeting a bound on the *pfd* of a 1oo2 system (including a bound of *pfd*=0), for a broad range of scenarios. This advantage in worst-case results holds if developers choose between a better and a worse process with 50-50 probability; but even, counterintuitively, in a range of scenarios in which they choose the better process more often than the worse process;
- the actual (not worst case) effects of forcing diversity cannot be predicted from existing models.

The models we have used give insight with practical consequences. They allow "what if" analyses to compare options before developing a system. They do not predict dependability figures for a specific system: one could not now estimate parameter values, and derive such system-specific predictions with any confidence. Whether this may become feasible after extensive empirical experience remains to be seen: there is now no indication that some uniform pattern will emerge from empirical data.

On a related note, we have been asked whether our "mathematical facts" could be validated for a specific real-world project. But the single data point provided by one project cannot validate or reject a probabilistic prediction, no matter whether generated by theorems or by extrapolating from empirical data. The advantage of theorems is that they can be trusted without empirical validation, provided that their assumptions (here, for most results, independence between development processes and true 1-out-of-2 behavior of the architecture) hold.

The details of our mathematical results depend on the specific architecture (1-out-of-2) examined. We expect that most results will extend rather directly to 1-out-of-N systems; and many observations, e.g. about the importance of the probability of "effectively fault-free" versions and systems, are valid in general. On the other hand, K-out-of-N systems, e.g. a triple modular redundant, 2-out-of-3 voted system, will require more complex models.

We have *not* considered the problem of assessing the reliability of a specific system. Our results are about probability distributions over the possible outcomes of a development project, important for decision before and during development. For the client adopting one product, instead, or the regulator approving it, what matters is the dependability of that one product. Especially with critical applications, one wants high confidence that sufficient dependability *has been* achieved. Although, with stringent dependability requirements, acquiring this confidence may be difficult [8, 9], about the finished product the client or regulator can rely on detailed specific evidence. They can examine the finished product itself and, perhaps most importantly, they can run operational testing. The risk reduction considerations we have discussed may *contribute* to form prior probability distributions that an assessor combines with new evidence, possibly exploiting techniques for simplifying their combinations with the results of testing [21, 25 , 31, 32].

ACKNOWLEDGMENT

This work was supported in part by the Artemis Joint Undertaking and the U.K. Technology Strategy Board (ID 600052) through the SeSaMo project, and by the DISPO project, funded under the CINIF Nuclear Research Programme by EDF Energy Limited, Nuclear Decommissioning Authority (Sellafield Ltd, Magnox Ltd), AWE plc and Urenco UK Ltd. ("the Parties"). The views expressed in this paper are those of the author(s) and do not necessarily represent the views of the members of the Parties. The Parties do not accept liability for any damage or loss incurred as a result of the information contained in this paper.

REFERENCES

[1] R. T. Wood, R.Belles, M. S. Cetiner, D. E. Holcomb, K. Korsah, A. S. Loebl*, et al.*, "Diversity Strategies for Nuclear Power Plant Instrumentation and Control Systems," NRC, U.S. Nuclear Regulatory Commission, NUREG/CR 7007, 2010.

[2] G. G. Preckshot, "Method for Performing Diversity and Defense-in-Depth Analyses of Reactor Protection Systems," NRC, U.S. Nuclear Regulatory Commission, NUREG 6303, 1994.

[3] P. Popov, L. Strigini, and A. Romanovsky, "Choosing effective methods for design diversity - how to progress from intuition to science," in *SAFECOMP '99, 18th International Conference on Computer Safety, Reliability and Security*, Toulouse, France, 1999, pp. 272-285.

[4] B. Littlewood and L. Strigini, "A discussion of practices for enhancing diversity in software designs," Centre for Software Reliability, City University London, DISPO project technical report LS-DI-TR-04, 2000 http://openaccess.city.ac.uk/275/.

[5] ISO, "ISO 26262 Road vehicles -- Functional safety," ed, 2011.

[6] D. Niedermeier and A. A. Lambregts, "Fly-by-wire augmented manual control - basic design considerations," in *ICAS 2012, 28th Congress of the International Council of the Aeronautical Sciences*, Brisbane, Australia, 2012.

[7] A. German and G. Mooney, "Air vehicle software static code analysis—Lessons learnt," in *Ninth Safety-Critical Systems Symposium*, Bristol, U.K., 2001.

[8] B. Littlewood and L. Strigini, "Validation of Ultra-High Dependability for Software-based Systems," *Communications of the ACM,* vol. 36, pp. 69-80, November 1993.

[9] B. Littlewood and L. Strigini. (2011, May) 'Validation of ultra-high dependability...' - 20 years on. *Safety Systems, Newsletter of the Safety-Critical Systems Club*. Available: http://www.csr.city.ac.uk/people/lorenzo.strigini/ls.papers/2011_limits_20yearsOn_SCSC/

[10] L. J. Yount, "Architectural solutions to safety problems of digital flight-critical systems for commercial transports," in *6th Digital Avionics Systems Conference, Baltimore, Maryland, December 1984*, 1984, pp. 28-35.

[11] J. C. Knight and N. G. Leveson, "An Experimental Evaluation of the Assumption of Independence in Multi-Version Programming," *IEEE Transactions on Software Engineering,* vol. SE-12, pp. 96-109, 1986.

[12] D. E. Eckhardt and L. D. Lee, "A theoretical basis for the analysis of multiversion software subject to coincident errors," *IEEE Transactions on Software Engineering,* vol. SE-11, pp. 1511-1517, December 1985.

[13] B. Littlewood and D. R. Miller, "Conceptual Modelling of Coincident Failures in Multi-Version Software," *IEEE Transactions on Software Engineering,* vol. SE-15, pp. 1596-1614, December 1989.

[14] B. Littlewood, P. Popov, and L. Strigini, "Modelling software design diversity - a review," *ACM Computing Surveys,* vol. 33, pp. 177-208, June 2001.

[15] K. Salako and L. Strigini, "When does 'Diversity' in Development Reduce Common Failures? Insights from Probabilistic Modelling," *IEEE Transactions on Dependable and Secure Computing,* vol. 99, (in print).

[16] P. G. Bishop, "Software Fault Tolerance by design diversity," in *Software Fault Tolerance*, M. Lyu, Ed., ed: John Wiley & Sons, 1995, pp. 211-229.

[17] L. Hatton, "N-Version Design Versus One Good Version," *IEEE Software,* vol. 14, pp. 71-76, November-December 1997.

[18] P. G. Bishop, "Does Software have to be Ultra Reliable in Safety Critical Systems?," in *SAFECOMP 2013, 32nd International Conference on Computer Safety, Reliability and Security*, Tolouse, France, 2013, pp. 118-129.

[19] FAA, "Federal Aviation Regulations FAR 25.1309," Federal Aviation Administration, Advisory Circular AC 25.1309-1A, 1985.

[20] L. Strigini and D. Wright, "Bounds on survival probability given mean probability of failure per demand; and the paradoxical advantages of uncertainty," City University London, Centre for Software Reliability Technical Report, 2013 http://openaccess.city.ac.uk/1644/.

[21] L. Strigini and A. A. Povyakalo, "Software fault-freeness and reliability predictions," in *SAFECOMP 2013, 32nd International Conference on Computer Safety, Reliability and Security*, Tolouse, France, 2013, pp. 106-117.

[22] W. E. Howden and Y. Huang, "Software Trustability Analysis," *ACM Transactions on Software Engineering and Methodology,* vol. 4, pp. 36-64, 1995.

[23] A. Bertolino and L. Strigini, "Assessing the risk due to software faults: estimates of failure rate vs evidence of perfection," *Software Testing, Verification and Reliability,* vol. 8, pp. 155-166, 1998.

[24] D. Hamlet and J. Voas, "Faults on Its Sleeve: Amplifying Software Reliability Testing," in *1993 International Symposium on Software Testing and Analysis (ISSTA), in ACM SIGSOFT Software Eng. Notes, Vol. 18 (3)*, Cambridge, Massachusetts, U.S.A., 1993, pp. 89-98.

[25] B. Littlewood and J. Rushby, "Reasoning about the Reliability of Diverse Two-Channel Systems in which One Channel is 'Possibly Perfect'," *IEEE Transactions on Software Engineering,* vol. 38, pp. 1178 - 1194, Sept.-Oct. 2012.

[26] R. E. Bloomfield, B. Littlewood, and D. Wright, "Confidence: Its Role in Dependability Cases for Risk Assessment," in *DSN 2007, 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, Edinburgh, U.K., 2007, pp. 338-346.

[27] P. Popov and L. Strigini, "The Reliability of Diverse Systems: a Contribution using Modelling of the Fault Creation Process," in *DSN 2001, International Conference on Dependable Systems and Networks*, Goteborg, Sweden, 2001, pp. 5-14.

[28] P. Popov, V. Stankovic, and L. Strigini, "An Empirical Study of the Effectiveness of 'Forcing Diversity' Based on a Large Population of Diverse Programs," in *23rd International Symposium on Software Reliability Engineering (ISSRE 2012)*, Dallas, Texas, USA, 2012.

[29] I. Gashi, P. Popov, and L. Strigini, "Fault Tolerance via Diversity for Off-The-Shelf Products: a Study with SQL Database Servers," *IEEE Transaction on Dependable and Secure Computing,* vol. 4, pp. 280-294, October-December 2007.

[30] M. Garcia, A. N. Bessani, I. Gashi, N. Neves, and R. R. Obelheiro, "OS diversity for intrusion tolerance: Myth or reality?," in *DSN 2011, 41st International Conference on Dependable Systems & Networks*, Hong Kong, 2011.

[31] B. Littlewood, P. Popov, and L. Strigini, "Assessing the Reliability of Diverse Fault-Tolerant Software-Based Systems," *Safety Science,* vol. 40, pp. 781-796, 2002.

[32] B. Littlewood and A. Povyakalo, "Conservative Bounds for the pfd of a 1-out-of-2 Software-Based System Based on an Assessor's Subjective Probability of 'Not Worse Than Independence'," *IEEE Transactions on Software Engineering,* vol. 39, pp. 1641-1653, December 2013.