Advantages of open source processes for reliability: clarifying the issues

D. Bosio, B. Littlewood, L.Strigini
Centre for Software Reliability

M. J. Newby
Department of Actuarial Science and Statistics
City University, London, England

Abstract

Some authors maintain that open source software processes are particularly well-suited for delivering good reliability. We discuss this kind of statement, first clarifying the different measures of reliability and of a process's ability to deliver it that can be of interest, and then proposing a way of addressing part of it via probabilistic modelling. We present a model of the reliability improvement process that results from the use of the software and the fixing of reported faults, which takes account of the effect on this process of the variety of software use patterns within the user community. We show preliminary, interesting, non intuitive results concerning the conjecture that a more diverse population of users engaged in reporting faults may give OSS processes an advantage over conventional industrial processes, in terms of fast reliability growth after release, and discuss further possible developments.

1 Introduction

Many claims have been made about the dependability of Open Source Software (OSS), some of them contradicting each other (OSS is generally better than Closed Source Software (CSS), or vice-versa), some of them presenting a challenge to intuition (OSS is more secure because of the accessibility of its source code to all, including would-be intruders). We find that these arguments often fail to clarify the claims made and the reasoning supporting them.

We propose to add some clarity to some of the issues. Specifically, we wish: to separate different qualities of possible interest, corresponding to

Workshop on Open Source Software Development, Newcastle upon Tyne, 25-26 February 2002

different precisely defined measures related to reliability; then to give formal expression to some of the conjectured laws which support the claims made; to discuss to which extent these conjectures are consistent with the common understanding of the processes that produce software reliability. We do so via an example of probabilistic models of the effects on software dependability of factors in the software production process. In this paper, we use, as examples, models of a class which we developed earlier to weigh claims about the merits of different testing methods [1, 2, 3].

1.1 Different aspects of reliability

One can identify many attributes of a product that are components of its *dependability* (an umbrella word we will use for all the meanings of "reliability" in its common, non-mathematical sense, to avoid confusion with the specialist term "reliability").

For instance, we can discriminate between the ability of a system to deliver continuous correct service, described by "reliability" in its strict technical sense, from its ability to provide a correct service at any given moment. E.g., the probability of surviving a mission is a reliability measure while the average uptime is an availability measure. This distinction is important because the two measures describe requirements whose relative importance varies between users and circumstances; because a system can exhibit a high level of availability without a high level of reliability; and because the best system design for a given application (e.g. using static redundancy vs rollback recovery) varies depending on which requirement is most important.

Other important distinctions centre on the characteristics of the failures of interest, characterised by severity along a unidimensional scale, or in terms of the level of specification that they violate (specifications internal to the development process vs expectations of the users), or in terms of differences of kind: e.g. related to maintaining data integrity or privacy against intentional attacks (two attributes of *security*) or to avoiding failures that cause outcomes classified as hazards or as accidents (attributes of *safety*). It is obvious that these distinctions also matter to a debate about OSS processes, as exemplified by the debate over achieving security via openness vs via obscurity of code or algorithms.

In this paper we choose to talk, for ease of exposition, about a single reliability attribute, measured by the probability of a system completing a demand (e.g. a user session for interactive software) without failures.

We also wish to point out two further important distinctions:

- reliability of a specific release of a product vs the evolution of reliability during the lifetime of the product. One can have software that is highly reliable at release but improves slowly (as an extreme example,

many safety-critical products will exhibit few failures but be subject to exceedingly time-consuming procedures for any update), just as software that is initially unreliable but improves quickly. From the users' viewpoint, in many critical applications it is necessary to know that software is sufficiently dependable when first deployed; in other applications, teething problems are tolerable, especially if the software can be expected to improve rapidly, or to be exempt from reliability deterioration in the long run. Some of the alleged advantages of open processes, e.g. prompt response to problem reports, seem more plausibly to aid fast reliability improvement than reliability at release time;

- average reliability over all users, vs reliability seen by individual users or groups of users. The manner of use of a product determines its reliability; so, the failures affecting different users differ in kind and frequency. Even software that performs well for most users may be ruinously unreliable for some of them. These unfortunate users are hidden in statistical averages, but the risk of becoming one of them should weigh heavily on the mind of anyone planning a software purchase. A software vendor is often interested in the average reliability among all users (and possibly the number of users so wretched that they might endanger the reputation of the vendor). A user is interested in reliability for him/herself. For instance, this concern contributes to the insistence of some military customers on being able to take over maintenance of the products they buy.

  Again, we see a plausible conjecture that some aspects of open processes improve the lot of the least favoured users, e.g. the possibility for minorities of users to modify the source code to fix their own specific problems, possibly leading to great advantages, from a buyer's viewpoint, even if the average reliability to be expected were less than that obtainable from a competing, less open process.

Interestingly, this last conjecture is related to an assumption that seems to underlie many of the claims for the usefulness of "open" processes: their ability to exploit diversity among developers and among maintainers. For instance, the saw "given enough eyes, all bugs are shallow" is obviously wrong if all the eyes have blind spots for the same bugs. What matters is that some eyes naturally see bugs that are hidden to other eyes. Diversity between eyes is a common principle in all development processes: from the use of independent V-and-V staff to "dual programming". The issue is which forms and extents of diversity work better, from the various viewpoints of interest. We will outline some useful research questions and conjectures for shedding light on possible advantages and disadvantages of open processes in this area.

1.2 Probabilistic modelling

We wish to shed some light on the possible contributing factors to the claimed greater (or lower) reliability of OSS. The questions of practical interest we want to answer are of the form "Does factor X in the development process tend to improve dependability measure Y"?

We choose here one aspect of statements like "the greater diversity between participants enjoyed by OSS processes causes better reliability in OSS products", often used either to argue that OSS processes favour dependability or to explain the good dependability observed in some products of OSS processes.

The first advantage of mathematical modelling is that it forces us to explain what we mean by "diversity" and "greater degree of diversity", to express formally which results we wish to compare, etc. After being so specific, we can often check whether it is plausible that the factor alleged to improve dependability actually improves it, and under which additional conditions we should observe this effect.

Some use of modelling is usually necessary for supporting a claimed causal effect between aspects of the software production process and its achieved results, even given some empirical evidence. What we do here is to make the modelling formal and explicit. The only alternative would be appealing to bare statistical evidence of correlation between the two. This could prove to be prohibitively difficult. Checking empirically even a simple statement like "OSS products are more reliable than the others, all things being equal" is difficult in practice, for various reasons: paucity of products with documented reliability, difficulty of choosing "equal" terms of comparison, expected high variability of the effects so that it may be exceedingly difficult to reach conclusions with any level of confidence.

We have started applying this approach to a specific scenario – we model the reliability growth of software while in use after release – and initially to a single conjecture of interest: that the diversity of users involved in fault-reporting in an OSS environment may give it an advantage, from this viewpoint, over comparable software that enjoys less of such diversity.

We do *not* inquire whether OSS products do improve faster than comparable non-OSS products (a worthwhile investigation if the investigator overcomes the difficulties cited above), nor whether OSS products do in general enjoy more diverse fault reporting. Instead we study how this kind of diversity would affect software reliability growth if the plausible assumptions of our model were true. Essentially, we wish to produce conjectural laws that link this kind of user diversity to reliability growth.

These laws are of interest to decision-makers, e.g. project managers who can influence the make-up of the user community engaged in failure reporting, or procurement managers who have to choose between products with visibly different make-ups of this community. Using a mathematical

model also clarifies which statistical evidence would support or refute the idea that these laws are at work in the real world, indicate confusing factors that may affect the measurements, and so on. Even without experimental support, a model that decision makers can recognise as consistent with their experience will allow them to scrutinise the less formal, intuitively appealing arguments proposed to them.

Our modelling is thus not framed in terms of OSS vs non-OSS processes: it applies to comparing any processes whose differences can be described in terms of its parameters.

2 A model of reliability growth during use

2.1 Description and basic assumptions

We start with an intuitive description of the process of finding faults in software while using it, and of removing them. A program has well-identifiable defects ("bugs", "faults"), which may cause it to fail. There is a set of users using the software. Some of them may actually be intentionally testing it, some just using it normally. We do not need at this stage to discriminate between the two sets. What matters is that they use the software, and they may experience failures, as a random process due to their (different) sequences of use of the software. If a failure occurs, the user may notice it and report it. If it is reported, someone may attempt to identify and remove the fault that caused it (and the attempt may succeed or fail).

We wish to describe the reliability growth patterns taking place as a result of all these factors, from the viewpoint of any one user or group of users.

We now go into more formal details of this model, as previously described in [1, 2, 3].

For simplicity, we restrict ourselves to a demand-based model of program execution. A program is given a demand, computes a result and terminates. In other words, we characterise the "extent of exposure" of the program to failure as a discrete variable represented by the number T of executions of (i.e., demands applied to) the software. A demand is characterised by the values of all the input parameters and machine state that determine the behaviour of the program in one execution. This model is very general, applying, for instance, even to interactive programs if we consider a "demand" and a "result" to include the sequences of all user inputs and of all the program's outputs during a session (cf [2, 4]).

The collection of all the possible demands is called the *demand space*. The demands which will cause the program to fail (*failure points)* form its *failure set*, which we describe as composed of multiple, non-overlapping *failure regions*, each a collection of failure points corresponding to a specific defect (or *fault*) in the code. If a failure point is found (through observing

a failure), and if an attempt is made to remove the fault that caused it, then either the failure region to which it belongs is completely eliminated (successful fix) or not at all (this is a simplistic description, commonly accepted in the literature; in reality, even the definition of what is "a fault" is ambiguous, as two different people trying to eliminate the cause of the same failure may well change different parts of the code; cf the discussion in section 2.2, *"So-Called Faults"*, of [2]).

Users use the software in many different ways. The users' ways of using the software are described by their *usage profiles*. A user's usage profile is the set of the probabilities of each possible demand being chosen by that user. It determines the different probabilities of the program failing[1] for that user due to each bug in the software (in a given number of demands by that user), and thus also the reliability of the software for that user.

We also assume that the demands chosen on different executions (by the same or different users) are statistically independent. Making this assumption realistic requires a judicious choice of what is defined as "one demand", and excludes some kinds of testing from our model. These assumptions can model "operational", "statistical" or "stress" testing – each regime is modelled by different usage profiles and different rates of bug reporting – but not partition testing. We will not be modelling the reliability growth that arises from early stages of testing by developers, which is of little present interest to us.

Different users may use the software more or less frequently. Individual users also differ in their probabilities of reporting the failures they observe.

2.2 Parameters of the model

The parameters in the model are

- $q_{i,j}$, the probability of the fault i causing a failure for the user j on a randomly selected demand,
- $r_{i,j}$, the conditional probability of a failure caused by the fault i being reported by user j if it occurs in an execution for that user,
- f_i, the conditional probability of the fault i being successfully fixed given it has been reported,
- T_j, the number of demands applied by user j by the moment in time at which we study the achieved reliability of the software, and total number of demands by all users $T = \sum_j T_j$.

[1] Note that we do not consider the consequence of a failure: the severity of a bug is only given in terms of the probability of selection in operation of a demand from the failure region associated to it.

All the probabilities just described, $q_{i,j}, r_{i,j}$ and f_i, are considered constant over successive demands[2]. In other words, we are assuming that the number of users, and their behaviour in terms of software execution and bug reporting, are constant over time. This constrains the generality of the model, but not as severely as it may appear. For instance, we can describe users recruited later as users who were always present but perform no executions until a certain time. We can also describe a step change in a user's usage profile in terms of two virtual users, one of which stops executing the software when the other starts.

The $q_{i,j}$ parameters are determined by each user's usage profile, and in turn they determine both the effects of each fault on the reliability experienced by that specific user, and the probability that that user will be able to report the fault. For a given fault i, larger $q_{i,j}$s correspond to users with higher probabilities of being affected by failures caused by fault i, thus users who are better at finding that fault: if these users are intentionally testing the software, they are better testers, and if they are using the software, they are the less fortunate users.

The model parameters describe measures of practical interest in a debate about software processes. About OSS processes, it is often stated that:

- their products tend to have better (or worse: opinions differ) initial quality than with alternative processes. This can be modelled by sets of $q_{i,j}$ parameters giving lower (conversely higher) $\sum_i q_{i,j}$s for the users whose experienced reliability we consider.
- they enjoy better failure reporting: higher values of [some] $r_{i,j}s$.
- they offer more responsive bug-fixing for at least some faults: higher f_is.

2.3 Reliability improvement as a result of use

This model represents the fact that the reliability improvement process is a stochastic process. The fixing of faults depends on when (and whether) they are found during execution, and their being reported, and the report prompting an action to fix the bug, and the fix actually removing the fault. The model describes statistically how this process will evolve.

For instance, the initial reliability of the program as seen by user j corresponds to $T = 0$, i.e. after no executions of the software, and is described by its probability of failure on demand (pfd) pfd_j

$$pfd_j = \sum_{i \in \{failure\ regions\}} q_{i,j} \ . \tag{1}$$

[2]Note the underlying simplifying assumption that the probability of a user reporting a fault does not depend on how many times that user has observed failures caused by that fault before.

We now consider the case where multiple users have executed the software, each user k having executed T_k demands. The probability of a fault having being removed is the probability of the fault having been reported (at least once) and fixed. The probability of the fault being reported at least once is $1 - P(\text{fault } i \text{ not reported})$. The probability of the fault not being reported by any user is given by

$$P(\text{fault } i \text{ not reported}) = \prod_{k \in \{users\}} (1 - r_{i,k} q_{i,k})^{T_k} \ , \tag{2}$$

Recalling that f_i is the probability of fixing bug i once it has been reported, we have that the probability of the fault i being removed is

$$P(\text{fault } i \text{ removed}) = f_i(1 - P(\text{fault } i \text{ not reported})) \ .$$

Every user will experience an improvement in reliability as a result of the faults fixed when revealed in this multi-user "testing" activity, but this improvement will be different for different users (as, indeed, would be their initial perceived reliabilities before testing). The mean reliability as seen by user j as a result of the multi-user testing, after a total number of executions $T = \sum_k T_k$, depends on the usage profiles of the other users, in the following way:

$$pfd_j = \sum_{i \in \{failure\ regions\}} q_{i,j} \left(1 - f_i (1 - \prod_{k \in \{users\}} (1 - r_{i,k} q_{i,k})^{T_k}) \right) \ . \tag{3}$$

Thus, the associated expected increase in reliability for user j as a result of exposure to the T tests will be

$$I_j = \sum_{i \in \{failure\ regions\}} q_{i,j} f_i (1 - \prod_{k \in \{users\}} (1 - r_{i,k} q_{i,k})^{T_k})) \ . \tag{4}$$

All the following factors will decrease pfd_j (and hence increase I_j, the increase in reliability of the software for user j): adding more users, performing more tests, increasing the value of any of the $r_{i,k}$ or $q_{i,k}$, or f_i parameters. In other words, the more users executing (and thus testing) the software, the greater the benefit for the community; the more likely each person is to report any bug, the more bugs will be exposed, allowing for more bugs to be corrected. The more likely the bug is to be removed if reported, the higher the resulting reliability of the software.

We note that users for whom $r_{i,j} = 0$ for all i are "free riders": they experience reliability improvement without reporting any bugs, and they benefit from bugs being reported by and fixed for others.

All these are somewhat unsurprising properties, simply confirming that the model captures "common-sense" understanding of how people interact

in fixing bugs. It is interesting to see how competing claims about OSS processes can be represented in this model. In a CSS process, one would expect there to be a small community of *special* users, the in-house testers, who put in a big "lump" of executions early on in the life-cycle and after major changes. These users have very high $r_{i,j}$ and aim to have high $q_{i,j}$ as they often *try* to cause failures. Yet theory shows [1, 2, 3] that if their $q_{i,j}$ for certain faults are lower than for "ordinary" users, the latter may see much worse reliability than the in-house testers. There is anecdotal evidence that this happens with many products. In an OSS process, this nucleus of heavy duty testers may well be smaller. On the other hand, it is plausible that "ordinary" users of OSS have $r_{i,j}$s that are often much higher than with "ordinary" users of CSS, due to perceived higher chance of obtaining a fix; visibility of the source code and higher number of potential "fixers" should give higher $f_i s$ than for many commercial products (at least after some time from release).

2.4 Brief discussion of assumptions.

Of the assumptions of this model, the ones that seems most seriously unrealistic are those of a constant r_{ij} for given i, j and of constant f_i. I.e., a user who observes a failure due to a certain fault more than once has the same probability of reporting it each time; and the probability of a fault being fixed depends only on whether it is ever reported. There are many plausible factors that would suggest other behaviours. For instance, users may be unlikely to report a failure that they recognise as similar to one already reported, especially by themselves; however, this does not matter as the model is only affected by a failure being reported *at least once*. For certain failures, users who have not reported them previously may become more willing to report them when they observe them again; but for others (or other users for the same failures) they may be most likely to report the failure at the first observation, and later discount it as a known nuisance. Many such psychologically plausible assumptions are possible. They could be represented in the model at the cost of added complexity, but this is premature during this first investigation. We will need later to examine under which circumstances they would, if true, cause serious changes in the results derived from the model.

We will not discuss here, for brevity, the other, more standard assumptions we made.

3 Diversity is useful

We now illustrate how models of this kind can be used to investigate the plausibility of quite general hypotheses. The hypothesis we investigate here

is: "Diversity is a good thing: all things being equal, it is better for users to have diverse demand profiles than for them to have the same profile."

We want to compare a situation where users have many diverse profiles with one in which all the users have the same profile (we will see below which one), if the total number of executions of the software is the same in both cases: $\sum T_k = T$. Starting with the former situation, we define our "ideal average equivalent user", where "average" refers to the effectiveness in fault reporting, measured by its effect on the potential reliability improvements if all faults reported are fixed. For each bug i, it is a user who has the average of all the considered profiles, weighted with their respective number of executions. Mathematically this corresponds to a user with associated parameters $r'_{i,j}$ and $q'_{i,j}$ such that

$$r'_{i,j}q'_{i,j} = \frac{\sum_{k\in\{users\}} T_k r_{i,k} q_{i,k}}{\sum_k T_k} \ . \tag{5}$$

Let us then consider a situation in which all the users are "average" users: all have identical parameters $r'_{i,j}$ and $q'_{i,j}$ satisfying equation (5) for all is. The theorem of arithmetic and geometric means [5] (see appendix 5) now tells us that, for each bug i, its probability of not being reported is smaller in the case of diverse users than in the case of all users having this ideal average profile

$$\prod_{k\in\{users\}} (1 - r_{i,k}q_{i,k})^{T_k} < \left(1 - r'_{i,j}q'_{i,j}\right)^{\sum_k T_k} \ , \tag{6}$$

unless all the products $r_{i,k}q_{i,k}$ are equal, in which case equality occurs. Combining the effect over all the failure regions i, we obtain

$$\sum_{i\in\{failure\ regions\}} q_{i,j} f_i \left(1 - \prod_{k\in\{users\}} (1 - r_{i,k}q_{i,k})^{T_k}\right) \ > \\ \sum_{i\in\{failure\ regions\}} q_{i,j} f_i \left(1 - \left(1 - r'_{i,j}q'_{i,j}\right)^{\sum_k T_k}\right) \ . \tag{7}$$

What does this tell us? *Any* user (with profile $q_{i,j}$, $r_{i,j}$) would prefer the previous exposure of the software to have been diverse rather than uniform, because diversity gives them higher reliability. There are two conditions here to represent "all things being equal" in our comparison of diverse-profile testing with uniform-profile testing. They are (i) that the same number of demands are executed in *each* case and, (ii) that the uniform profile is, in an intuitively appealing way, the mean of the different profiles used in the diverse-profile testing. Subject to all things being equal in this natural way, we have thus shown that diverse-profile testing is superior to uniform-profile testing, in the sense it can be expected to deliver greater reliability improvement to *every* user (no matter what their profile is).

4 Discussion

4.1 Further implications of the model — conjectures

We consider here a few questions of clear interest, and speculate about which answers the model would give to these questions. I.e., we formulate mathematical conjectures; if in the future we manage to prove them to be consequences of the model, they will gain the status of conjectures about the actual evolution of software reliability. We rely heavily on our understanding of similar models described in [1, 2, 3].

4.1.1 The individual user's viewpoint

Studies in software reliability usually refer to the *average* reliability over all users. In other words, they apply to predictions of the total number of failures observed by the whole population of users. Of course, if users have very diverse profiles, as is the case for many products, even a very good average will not avoid very poor reliability for some users. In other words, just because a product is known to be very reliable on average, I cannot trust that it will be very reliable for me. Our model is a step in the right direction for considering this issue: it refers to the reliability for each specific user, and thus it will allow one to study distributions rather than averages.

In a practical situation, what would the model predict for an individual user, or set of users, who benefit from the collective fault reporting and fixing effort?

A simple (approximate) analogy with the previous work cited is that it is "as though" there were two users, user 1 whose viewpoint we are taking, and user 2 representing all the other users and executing many more demands, $T_2 >> T_1$ (or $T_1 = 0$, as when user 1 is deciding whether to adopt the software). We can compare two scenarios with equal total fault-detecting and reporting efficacies, i.e., two scenarios which, if one tried to estimate the reliability of the program (averaged among all users) by looking at the rate of generation of fault reports, would give identical estimates.

Suppose that user 1's profile is very different from that of all others. We can expect that after any amount of time and use, pfd_1 will be better than if user 1 were alone to report failures; yet much worse than if all users had the same profile as user 1 (this is the standard argument in favour of "operational" testing). Yet the situation is probably more complex. In [2] it was found that if software is tested by user 2 with a constant profile, there is a single profile that delivers the best reliability for user 1, among all profiles that yield the same initial probability of failure per execution, in other words under the constraint

$$\sum_{i \in \{failure\ regions\}} q_{i,1} = \sum_{i \in \{failure\ regions\}} q_{i,2} \ , \tag{8}$$

This optimal profile that user 2 should apply for maximum benefit to user 1 actually depends on the amount of use/testing, T. Its parameters $q_{i,2}$ satisfy the following relation ([2], eq. 28):

$$q_{k,1}\left(1-q_{k,2}\right)^{T-1} = q_{l,1}\left(1-q_{l,2}\right)^{T-1} \quad . \tag{9}$$

for any two faults k and l, under the constraint (8).

What does our "diversity is good" result adds to this previous result? If I (user 1) have to choose, for my software, among testing regimes with constant profile (as though executed by the single other user 2), then equation (9) tells me how to choose this profile for maximum reliability. But on top of that, if I then split my total number T of test cases among many users, and give them more varied profiles, of which this optimal profile is the average according to equation (5), I will get even better reliability for myself. Software that has been used by a diverse community can reach me with better reliability than software that has been used by "the best possible" (from my viewpoint) uniform community with comparable total usage and fault reporting efficiency.

Other interesting questions deserve investigation, e.g.: under which conditions could a user be aware of being too "special" to benefit greatly from diversity in the user community? When is it that an apparently very good fault finding process (large rs and qs) is less useful for me than a "worse" one (generally smaller rs and qs)?

And if "diversity is good", is there a sense in which "more " diversity is better than "less"? We need to characterise what "more" means in terms of the model's parameters before we can ask whether differences between OSS and other approaches matter in this respect. For instance, we may ask under which conditions a law of diminishing returns would apply for diversity.

4.1.2 Evolution over time

All the results so far have been discussed in terms of reliability at a certain, arbitrary moment in the history of use of the program. All results contain parameters T_k, or their sum $T = \sum_k T_k$.

We are really interested in how the program's reliability evolves over time. We showed in [3] a phenomenon whereby testing with a profile similar to one's usage profile yields better reliability growth in the short term, but in the long term different profiles, with some emphasis on the "less important" bugs, are more beneficial (i.e., in the long run, the "important" bugs will have been found and fixed no matter what; but the other ones are difficult to get rid of).

In our model we can therefore conjecture two main contributing factors to the reliability growth as observed by user j. In the short term, corresponding to initial rapid reliability growth, it is affected mostly by those users with similar profiles to user j's own. In the long run, the profiles

which differ from user j's will contribute more to improving reliability as seen by j.

4.1.3 Predictability of results, dependability of process

We have so far referred to the average, or expected value of reliability measures for a given user. This acknowledges that reliability growth is a stochastic process: for instance, a fault i with high $q_{i,j}$s is likely to be discovered early on, but it may well (with low probability) go undetected for a long time. The probabilities of different histories of reliability growth are determined by our model parameters. The averages that we have been discussing are defined over all the possible histories of reliability growth. So, they are useful indicators, but they may be misleading as they hide the potential variation between different histories.

In reality, what matters in a project is the reliability growth history that actually takes places. When I am stuck with an unreliable product, it does not matter much that, if I consider all other possible histories, the average of all the reliability levels that I *could* have obtained would be much better than the one I actually see. So, in project decisions the probabilities of "bad" reliability growth histories - thus, the probabilities of histories that are "much worse" than average - matter.

In [3] we studied probability distributions of reliability growth histories, accounting also for the fact that the initial set of faults is unknown. We could show some counterintuitive examples of how comparatively bad failure reporting rates, from usage with a user's own usage profile, would be better defences against the risk of very poor reliability growth than even much higher reporting rates based on someone else's, different profile.

We would like to explore how this translates into predictions, for a new prospective user, of the risk of very bad reliability growth after adopting a new product, and how the degree of "openness" of the process and diversity of the user base should affect them.

4.2 Limits of this model and possible extensions

This model is not exhaustive: there are many aspects of OSS processes that we have not described. Some of these could be studied through extensions to the model, some require different methods. To give some examples:

- in OSS an individual user might make available to the other users a fix which would work only in his particular profile, which is often commercially unviable in CSS;
- having additional users reporting bugs may be generally useful as it will increase the chances to improve the reliability in any case. However, in this model there is no explicit representation of any resource

bottleneck, like scarcity of bug fixing staff, or simply delays due to the need to coordinate many fixes. As an example, consider the situation when bug fixing is a competing activity. Bug fixers may need to abandon one bug for another, so that increasing the likelihood of user k reporting bug i implies a higher probability of fixing bug i, but at the expense of fixing bug l. Mathematically this can be described by saying that increasing $r_{i,k}$ increases f_i, which in turn decreases f_l.

- the model lacks any notion of consequence of a failure, or failure "severity", yet it seems likely that users will be influenced by this (as well as their perception of its frequency) in deciding whether or not to report it. It may well be that attitudes to failure severity are different between OSS and commercial developments.

5 Conclusions

We started this work to clarify to ourselves some open issues about the relationship between software processes and software dependability, using the tools we know, i.e. a simple, formal, probabilistic approach. We were motivated by interest in open source processes, but a reader may well say that this paper is not about"open source". Our considerations about dependability attributes, and the model we propose, apply to any scenario in which the important factors at work can be described by the model's parameters. It so happens that some of these parameters are plausibly influenced by "openness" factors. E.g., if a development community appears responsive to failure reports, we may well expect users to report failures more often than they would otherwise. Such thorough reporting of failures might of course be produced by appropriate management decisions or cultural factors even without open source development.

So, we have not studied "open source" per se. Yet this way of modelling does seem useful for decisions involving the choice between more or less open project styles, both in managing development projects and in software procurement. The model's universe includes many of the factors that seem important: the initial quality of the software, various factors affecting its growth, the different reliability levels experienced by different users, etc.

In fact, we would argue that only questions of the kind we have chosen, "how does factor X affect dependability attribute Y" are likely to produce interesting general answers. Questions of the form "how reliable is open source software?" may elicit highly specific, useful answers of the form "product X achieved an average time between crashes of y days for user community Z", but no generalisation without the support of theories – i.e., of models.

As to useful answers that this model can give to our questions, we have only proven a first theorem. This theorem does resolve an intuitively unclear

issue, but does not seem especially useful in the practical situations, in which two processes are likely to differ in all the $r_{ij}q_{ij}$ in arbitrary ways. However, even this modest theorem has some practical implications. E.g., given a community of similar users who are not likely to report all failures, it would pay off to encourage different users to focus on reporting different kinds of failures, provided their reporting frequencies, taken together, still satisfy equation (5).

We will continue to interrogate this model, e.g. about the relationships between reliabilities observed by different users, as outlined in the previous sections.

We plan to obtain clarifications of what "one should believe" given plausible assumptions, and thus about the consistency of various claims about the differences between processes that differ in the extent of some aspect of "openness". We shall investigate which predictions can be checked empirically to decide whether these models, however stylised, are useful approximations for important, dependability-related aspects of real, complex processes.

We have chosen to study aspects of the software life-cycle for which our modelling approach seems appropriate. We are sure that there are other aspects for which it would be ineffective. For instance, some will argue that a prestigious open source project will produce high initial dependability, or long-term maintainability, simply because it will attract highly skilled and dedicated contributors. In our model, this result would be represented by low values of the sum in (1) for all js. The process it describes is not represented in the model, nor would we expect any advantage from trying to represent it. To investigate how this conjecture should affect one's decisions, e.g. in setting up an open-source project, one would need to use knowledge from psychology, sociology or economics, and still would expect rather imprecise answers. For someone intent on procuring a software product, the conjecture is probably irrelevant, as what matters if whether that specific product *has* been built by especially skilled individuals, and/or *is* very reliable.

We expect our style of modelling to complement such other methods of investigating software engineering problems, in terms of general laws or of specific cases.

Acknowledgements

This work was supported in part by EPSRC, within the Interdisciplinary Research Collaboration in Dependability of Computer-Based Systems, DIRC.

Appendix: theorem

In this appendix we describe the theorem of arithmetic and geometric means. Let us consider two set of non-negative numbers $a_1, a_2, \ldots, a_n$ and $p_1, p_2, \ldots, p_n$. We will call the ps weights. The weighted arithmetic means of the as is defined as

$$A(a,p) = \frac{\sum_{k=1}^{n} p_k a_k}{\sum_{k=1}^{n} p_k} \ ,$$

whereas the weighted geometric means is defined as

$$G(a,p) = \left(\prod_{k=1}^{n} a_k^{p_k} \right)^{1/\sum_k p_k} \ .$$

The theorem ([5], p.17) says that $G(a,p) < A(a,p)$ unless all the as are equal. Raising both sides to the power $\sum_k p_k$ the theorem gives the equation

$$\prod_{k=1}^{n} a_k^{p_k} < \left(\frac{\sum_{k=1}^{n} p_k a_k}{\sum_{k=1}^{n} p_k} \right)^{\sum_k p_k} \ . \tag{10}$$

Replacing a_k with $(1 - r_{i,k} q_{i,k})$ and the weights p_k with the number of executions T_k in equation (10) yields equation (6). Indeed, we find

$$\prod_{k=1}^{n} (1 - r_{i,k} q_{i,k})^{T_k} < \left(\frac{\sum_{k=1}^{n} T_k \, (1 - r_{i,k} q_{i,k})}{\sum_{k=1}^{n} T_k} \right)^{\sum_k T_k} \ .$$

In the right hand side term we can recognise our definition of the "ideal average user". Indeed, by the definition of "ideal average user"(5), we have

$$\left(\frac{\sum_{k=1}^{n} T_k \, (1 - r_{i,k} q_{i,k})}{\sum_{k=1}^{n} T_k} \right) = 1 - \frac{\sum_{k=1}^{n} T_k r_{i,k} q_{i,k}}{\sum_{k=1}^{n} T_k} = 1 - r'_{i,j} q'_{i,j} \ .$$

References

[1] P. Frankl, D. Hamlet, B. Littlewood, and L. Strigini. Choosing a testing method to deliver reliability. In *Proceedings 19th International Conference on Software Engineering ICSE'97*, pages 68–78, Boston, USA, 1997. IEEE Computer Society Press.

[2] P. Frankl, D. Hamlet, B. Littlewood, and L. Strigini. Evaluating testing methods by delivered reliability. *IEEE Transactions on Software Engineering*, SE-24(8):586–601, 1999.

[3] M. Pizza and L. Strigini. Comparing the effectiveness of testing methods in improving programs: the effect of variations in program quality. In *Proceedings 9th International Symposium on Software Reliability Engineering, ISSRE '98*, pages 144–153, Paderborn, Germany, 1998. IEEE Computer Society Press.

[4] L. Strigini. On testing process control software for reliability assessment: the effects of correlation between successive failures. *Software Testing Verification and Reliability*, 6(1):36–48, 1996.

[5] G. Hardy, J.E. Littlewood, and G. Polya. *Inequalities*. Cambridge University Press, 1952.