Preprints of the 19th World Congress
The International Federation of Automatic Control
Cape Town, South Africa. August 24-29, 2014

Operating Experience of Programs and Changing Demand Profile – Consideration of Paths

Wolfgang D. Ehrenberger

Hochschule Fulda, Marquardstraße 35, D - 36039 Fulda
(Tel ++49 661 9640 325; email: Wolfgang.D.Ehrenberger@informatik.hs-fulda.de

Abstract: Since more and more software exists, it is economically important to estimate whether or not operating experience gained with earlier software applications can be used in new applications. Normally new applications have another demand profile than the earlier applications had. For safety-related applications quantitative relationships are required. This contribution derives formulae that can be used to estimate the failure probability of the software in the new environment. In contrast to the work of other authors the present considerations are not based on software modules, but on execution paths. The related inaccuracies are taken into account. An example is given, as well as a method for getting and storing the path characteristics. The pre-requisites that have to be met in order to make the derived formulae applicable are mentioned.

1. INTRODUCTION

In safety-related software applications the question about the confidence that can be placed in pre-existing software becomes more and more important: Should software that requires licensing be developed for the current project from scratch, or is it better to use software that has been used for a certain time in other applications? It seems to be clear that certain standard functions, as they are usually provided by compilers, should not be re-written, but rather taken from the related library. Similar views are common on operating systems. Meanwhile a large amount of frequently used application software exists and therefore some general thoughts seem to be in order. We ask: What data are needed to accept that software in another application? One will usually accept software if the new demand profile is identical or at least very similar to the pre-existing application profile and whose number of successful executions or runs is large. If the new profile differs from the old one, a quantitative estimation about the effect of the differences is helpful.

1.1 Characteristic of this paper

This contribution discusses the arising questions. It is based on the theory of stratified sampling, which has been known for a long time as e.g. given by (Saifuddin, 2009) or (de Vries, 1986) but seems to have not been recognized yet by the software community. In this contribution software is considered as a set of paths. A path is a possible execution of the software from its starting point to its end point. *Each path is recognized as a stratum.*

If we consider probabilistic software verification it is problematic to see software as a composition of individual modules, e.g. subroutines, functions, methods or objects. I think it is better to consider software as a composition of paths: Because it is the paths that are really executed. The modularised view is probably more suitable for hard wired equipment. But software has a clear advantage over hardware: It does not necessarily get less reliable, if it gets larger. So we are better off, if we code complicated functions in software. The disadvantage of software in contrast to hardware is the possibly far reaching effect of one programming fault along any possibly extended computing path. The considerations of this contribution take care of that because they focus on execution sequences and not on code parts.

1.2 Literature

Statistical testing and operating experience and the possible conclusions that can be drawn from them have fascinated researchers since a long time.

(Littlewood and Strigini, 1993) consent with a view of the British authority who is responsible for licensing nuclear power plants, which says, it is impossible to verify or validate failure probabilities per demand, that are lower than 10^{-4} for software. The application area was reactor protection systems, which have to operate basically on demand, e.g. for shutting the reactor down, and are called only rarely. The limitation was not claimed for frequently called functions[1]

[1] "function" is used for something that happens or has to happen, not as it is used by some programming languages, as e.g. C

Copyright © 2014 IFAC

such as the storing of an editor or the starting of a passenger car.

(Littlewood, 2013) and (Butler and Finelli, 1999) explain that it was impossible to demonstrate high reliabilities of software by probabilistic testing. Their results are based on a pure black-box view. The mathematical foundations are correct and demonstrated carefully; so are the conclusions. The present contribution, however, does not rely on a *black-box view, but assumes a certain knowledge on the internals of the software, the knowledge of its paths.* If the paths are known, more precise and more optimistic statements can be made and one can conclude from the software behaviour in one demand profile about its behaviour in another. As far as I know, only few computer scientists have dealt with the inner structure of software that is to be certified probabilistically. Among these are (Kuball, May and Hughes a, b, c 1999) and (Söhnlein et alii, 2010). The earlier quoted reservations against the demonstration of high reliabilities by probabilistic means mainly rest upon the infeasible high numbers of required test cases or testing times. (Littlewood and Wright, 1997) describe thoroughly how these numbers or times are to be derived. In contrast to the following they also consider the appearance of failures. Of particular interest is their proof of the equivalence of Bayesian and frequentistic thinking. A related demonstration is also found in (Ehrenberger et alii 1985).

1.3 Overview

The following chapter 2 discusses the principles of stratified sampling of software. Chapter 3 considers the effect of inaccuracies in the data that form the basis of the calculations. Chapter 4 gives an example, chapter 5 deals with the acquisition of the necessary data and Chapter 6 contains the conclusive remarks, and indicates limitations of the method. The appendix lists the prerequisites that are necessary to do the mentioned calculations. I believe that these prerequisites are so demanding that the related effort will only pay off, if the software has to deal with safety applications.

2. MONOLYTHIC AND COMPOSED SOFTWARE

2.1 Prerequisites and basic formula

Ideal assumptions are made, in particular: No failure has occurred in the past. Then we get for the upper limit $\tilde{p}$ of the failure probability per demand p after n operational runs, such that $p < \tilde{p}$ with a known probability, i.e. a known degree of significance α:

$$\tilde{p} = \frac{-\ln \alpha}{n}$$

α = 1 - level of confidence. The confidence interval refers to one side. See also (IEC 61508-7, 2010).

2.2 Monolithic System

We start with a system that is taken as a unit, as a black box; it does not have any known sub structure like modules or paths. It holds for the failure probability of the **t**otal system after n_t successful runs to the degree of significance α:

$$\tilde{p}_t = \frac{-\ln \alpha}{n_t} \quad ; \qquad (1)$$

ideal conditions are assumed. The subscript t indicates that the whole software and all runs are meant. It is expected that the view on the system does not influence its failure probability; i.e. that the failure probability given by (1) is also received as calculation result, if we consider the software as being composed of paths.

2.3 Composed System, stratified sampling

Each program can be thought of as being composed of a set {N} of paths. See also formulae (7) as an example.
Definition: A path consists of the statements that are traversed during a possible run through a program from its start to its end; it ends, when it has no further effects on other code parts; if a path ends, a new path can begin.
Assumption: The demand profile of a program or program part is described by the usage of its paths.
We define further:

N number of paths,
n_i number of runs (or traversals) of path i.

The total number of runs n_t of the system equals the sum of the number of runs of all paths i

$$n_t = \sum_{i=1}^{N} n_i \ .$$

The probability of running path i is $\pi_i = \frac{n_i}{n_t}$; $\sum_{i=1}^{N} \pi_i = 1$.

The upper limit of failure the probability of path i is

$$\tilde{p}_i = \frac{-\ln \alpha}{n_i} \qquad (1a)$$

If a path is executed without any failures, we get, considering the one-sided confidence interval:

The number of (fictitious) failures of path i during n_t runs
equals
the probability of selecting that path
times the probability of failure per run
times the number of runs of that path.

So number_of_failures_of_path_i = $\pi_i * p_i * n_i$. See also the theory of stratified sampling, e.g. in (Saifuddin, 2002) or (de Vries, 1986).

The total number of failures of the system is the probability of failure of the individual run p_t, times the total number of runs n_t; it is also the sum of the failures of the individual paths; therefore we get for a software system that consists of N paths:

$$n_t * \tilde{p}_t = \sum_{i=1}^{N} n_i * \tilde{p}_i * \pi_i \qquad (2)$$

This relation holds for the case of "no failure". We always consider the upper limits from (1a) for the probabilities. We get:

$$\tilde{p}_t = \sum_{i=1}^{N} n_i * \tilde{p}_i * \pi_i / n_t = \sum_{i=1}^{N} \pi_i^2 * \tilde{p}_i \qquad (3)$$

During the derivation of (2) and (3) no assumption has been made about any old or new profile. Both formulae are valid during the phase of gaining operating experience and during any new application of the software. Normally all $\tilde{p}_i$ are larger than $\tilde{p}_t$, because we assume 0 failures and a one-sided confidence interval and because the number of runs for gaining $\tilde{p}_t$ is larger than the number of runs for gaining any of the $\tilde{p}_i$.

2.4 The new profile

What is different between the old and the new operation is just the set of the π_i. This set represents the operation profile; it changes between the old and the new operation. Regarding the operating experience the π_i of the old operational profile have to be taken, for any new application the π_i of the new application have to be taken. The $\tilde{p}_i$, however, do not change between the old and the new operation. So, (3) can also be used for the new profile.

There are some conditions, however: *All paths must be known explicitly*, as well as their transition numbers in both the old and the future application. If these numbers are not known, a *conservative estimate* is needed. Therefore the paths as such and the number of their traversals must be recorded during previous operations. If they are not known for the future operation, they must be conservatively estimated.

Also: *The data values occurring within one path execution must be sufficiently closely similar* in the old and the intended application. If this does not hold, *sub paths* must be defined that reflect the differing data values.

3. INACCURACIES

If the individual n_i are not exactly known, we have to deal with related uncertainties. These can occur to both the old and the new operation profile. We are interested in a conservative estimation. From (1a) we get for each path of the experienced profile:

$$\tilde{p}_{i_\max} \leq -\frac{\ln\alpha}{n_{i_\min_old}} = -\frac{\ln\alpha}{n_i * \delta_{i_\min_old}} \quad (4)$$

For the new profile we take the largest possible δ_i and the smallest possible δ_t and we estimate:

$$\pi_{i_\max_new} = \frac{n_{i_\max_new}}{n_{t_\min_new}} = \frac{n_{i_new} * \delta_{i_\max_new}}{n_{t_new} * \delta_{t_\min_new}}$$

$$\cong \frac{n_{i_new}}{n_{t_new} * \delta_{t_\min_new}{}^2} \cong \frac{n_{i_new} * \delta_{i_\max}{}^2}{n_{t_new}} =$$

$$= \Delta^2 * \pi_{i_new_estimated}$$

The three last equalities are valid, if:

$$\frac{1}{\delta_{t_\min_new}} = \frac{\delta_{i_\max_new}}{1} = \Delta$$

$\pi_{i_new_estimated}$ stands for the new selection probability of path i without consideration of the uncertainties. We always assume $\delta_{min} < 1$ and $\delta_{max} > 1$. For the new operation we get from (4):

$$\tilde{p}_{t_new_\max} = \sum \pi^2_{i_new_\max} * p_{i_\max} \cong$$

$$\sum (\Delta^2 * \pi_{i_new_estimated})^2 * \left(\frac{-\ln\alpha}{n_i \delta_{i_\min_old}}\right) \quad (5)$$

This makes it possible to consider the influence of inaccuracies of the observation of the past operational runs and of the estimation of the future demands. If also $|\delta_{i_\min_old}| = \Delta$, it holds:

$$\tilde{p}_{t_new_\max} = \sum \pi^2_{i_new_\max} * \tilde{p}_{i_\max} =$$

$$= \Delta^5 * \sum \pi^2_{i_new_estimated} * \tilde{p}_{i_estimated} \quad (6)$$

Obviously the inaccuracies of the knowledge of the new demand profile dominate the inaccuracies of the result, as they occur to the 4th power. Table 1 shows examples, based on (5), i.e. without taking into account the influence of the $\delta_{i_min\text{-}old}$.

Table 1. Effect of inaccuracies of the knowledge of the new demand profile (π_i s), factor

Δ	1.1	1.2	1.5	2	3	
Effect on $\tilde{p}_{t_new}$	1.5	2	5	16	81	

The table gives the very worst case, as it assumes derivations to the worse by all paths. But in reality an overestimation of the number of runs of one path will result in an underestimation of the number of runs of another path. See also Table 3 versus Table 2.

4. EXAMPLE

Fig. 1 gives an example of a code fragment. The fragment has 4 paths:

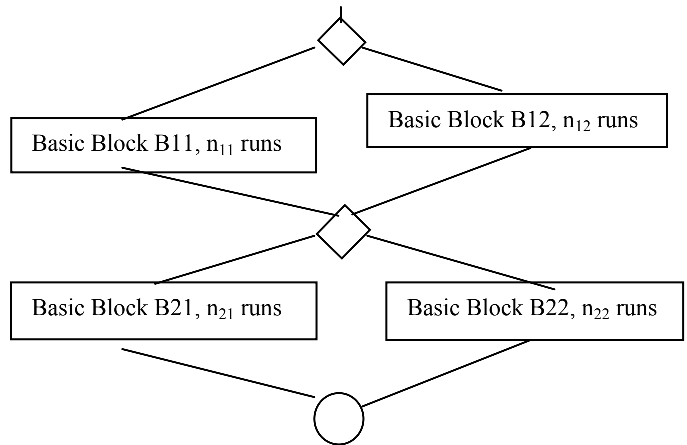

Fig. 1. Flow Diagram: Code fragment with 2 branches consisting of 4 basic blocks, each traversed n_{ij} times; 4 paths; $n_{11} + n_{12} = n_t = n_{21} + n_{22}$.

Path 1 = {B11, B21}, Path 2 = {B11, B22},
Path 3 = {B12, B21}, Path 4 = {B12, B22}. (7)

In total 30000 operational runs are considered; α is assumed to be 0.05. (1) gives $\tilde{p}_t = 10^{-4}$. We assume the individual paths have the number of runs of Table 2. The π_i and $\tilde{p}_i$ are calculated; the latter ones at a level of significance of 0.05 after (1a); the end result is calculated by (3), leading to the same value of $\tilde{p}_t$.

Table 2. Operating experience of the code fragment of Fig. 1

	Path 1	Path 2	Path 3	Path 4	Total
n_i	12 000	6 000	9 000	3 000	30 000
$\pi_{i\ old}$	0.4	0.2	0.3	0.1	1
$\tilde{p}_i$	$2.5*10^{-4}$	$5*10^{-4}$	$3.3*10^{-4}$	10^{-3}	10^{-4}

If the demand profile of the new application differs from the old one, the n_i of Table 3 might apply, resulting in the other figures of Table 3. Note that the $\tilde{p}_i$ of the paths do not change, $\tilde{p}_t$ however increases significantly.

Table 3. New operation profile of the code fragment of Fig.1

	Path 1	Path 2	Path 3	Path4	Total
n_{inew}	300	3000	2700	24000	30 000
π_{inew}	0.01	0.1	0.09	0.8	1
$\tilde{p}_i$	$2.5*10^{-4}$	$5*10^{-4}$	$3.3*10^{-4}$	10^{-3}	$6.9*10^{-4}$

If the concerned π_{i_new} are not well known, the new p_t might still require a correction according to Table 1. If they were inaccurate by 20%, $\tilde{p}_t$ would be too optimistic by a factor of 2.

It should be noted that the results of both Table 2 and Table 3 are not gained by a calculation based on the failure probabilities of the individual basic blocks as they can be calculated by using (1a) in connection with their traversal numbers. See Table 4.

Table 4. Failure probabilities of the Basic Blocks, old profile

	B11	B12	B21	B22	Total
n_{Bj}	18 000	12 000	21 000	9 000	60 000
$\tilde{p}_{Bj}$	$1.67*10^{-4}$	$2.5*10^{-4}$	$1.4*10^{-4}$	$3.3*10^{-4}$	
π_{Bj}	0.6	0.4	0.7	0.3	

Table 4 demonstrates: There is no easy way to derive a failure probability for the total software from the failure probabilities of basic blocks or modules.

We can remark: Since $n_{11} + n_{12} = n_t = n_{21} + n_{22}$, it holds for the upper part and the lower part of the basic blocks of the figure:

$$\tilde{p}_t = \frac{-\ln \alpha}{n_t} = \tilde{p}_{B11withB12} = \tilde{p}_{B21withB22}$$
$$= \sum_{j=1}^{2} (\pi_{1j})^2 * \tilde{p}_{1j} = \sum_{j=1}^{2} (\pi_{2j})^2 * \tilde{p}_{2j} \quad (8)$$

5. COLLECTION OF DATA

A program has usually thousands, if not millions of paths. In order to use the here mentioned theory, the data collection has to be nearly exhaustive. It has at least to be able to consider all paths in principle. Each path has to be characterized as such and the number of its traversals counted. It is suggested to store the characterisations and the traversals in a tree.

5.1 Storing

All basic blocks of the code are instrumented with an operation that can characterize the related path. Such an instrumentation can use a floating point number for each basic block that is connected with the so far gained result by one of the primitive operators ρ Є {+,-,*,/}. The so gained path characteristics are used to address a node of an AVL tree (Adelson-Velskii, 1962) during the phase of gaining the operation experience. The node of the tree could have the following shape:

```
struct pathCharacteristic {
   double characterizingNumber;
   unsigned long numberOfPathRuns;
   struct pathCharacteristic *left;
   struct pathCharacteristic *right;
};
```

As AVL trees are always well balanced, the effort of inserting into the right place in the tree increases only logarithmically with the size of the tree. It would only take about 17 steps for 100 000 paths and only about 20 steps for one million paths. The same applies for finding a path-related node for counting the runs. After each path traversal its `numberOfPathRuns` is increased by 1.

The new operation profile should then be simulated using the same software. A comparison of the gained new number of runs to the old ones would enable to estimate upper limits of the failure probabilities in the new environment.

5.2 Example of an instrumentation

The software of Fig. 1 gets the following code lines in addition to the already existing ones.

Starting point of the program, at the beginning of the `main` function:
`double characterizingNumber = 1.0;`

And then in

Basic Block 1: `characterizingNumber += 2.0;`

Basic Block 2: `characterizingNumber -= 3.0;`

Basic Block 3: `characterizingNumber *= 5.0;`

Basic Block 4: `characterizingNumber /= 7.0;`

The resulting value of the `characterizingNumber` forms the argument of the subroutine that inserts into the tree at path end. May be, it is not necessary to use only prime numbers for calculating the characterising number.

6. CONCLUSIONS

It may well happen that the effort needed to implement the considerations of this contribution comes up to the effort for a "normal" verification procedure on the basis of a software analysis, related tests and formal proofs. Never the less even high reliability claims can be supported by this method. But it is not thought that white-box testing strategies could be completely omitted.

Should failures occur during the operating experience and have they not been removed, a special consideration is needed. Related one-sided intervals can be gained by applying the tables of the Poisson distribution. But a program that is known to contain faults will normally not be allowed for safety applications.

Meeting all the requirements that are connected with the method presented can be costly. If they cannot be met, deterministic reasoning is required to demonstrate that the violation does not have any effect or only a limited and tolerable one. As far as I know, the number of pre-use runs or test runs required by (1) can never be reduced. Using this method does not guarantee success in licensing at lower cost. Using it, however, always results in a warm feeling supplementing the results of other verification efforts; and sometimes it leads to a quantitative reliability claim. Its main area of application is probably allowing widely used software packages in new environments.

REFERENCES

Adelson-Velskii, G.M, and E.M. Landis (1962). *An Algorithm for the organisation of information* (Wikipedia 2013)

Butler and Finelli, R.W.Butler and G.B.Finelli (1999). *The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software*; IEEE Transactions on Software Engineering, Vol 19, No1

Bishop P.G., D.G.Esp, F.D.Pullen, M.Barnes, P.Humphreys, G.Dahll, B.Barlan, J.Lahti, H.Valisuo (1987). *STEM a peoject on Software Test and Evaluation Methods* in Proceedings Safety and Reliability Symposium SARS 87, pp 100-117

de Vries, P.G. (1986). *Sampling Theory for Forest Inventory Stratified Sampling*, pp 31-55, taken from internet 2013 Springer Verlag

IEC 61508 (2010). ISO/IEC 61508-7 *Functional Safety of electrical/electronic/programmable electronic safety-related systems*, to be ordered via IEC Geneva or Beuth-Verlag Berlin

Ehrenberger, W., J.Märtz, G.Glöe and E.-U.Mainka (1985). *Reliability Evaluation of a Safety-related Operating System*, Safecomp 1985, Pergamon Press, editor B. Quirk

Kuball, S., J. May and G. Hughes, a (1999). *Building a system failure rate estimator by identifying component failure rates*; ISSRE 99, Proceedings, IEEE Computer Society Press

Kuball, S., J.May and G. Hughes b (1999): *Structural Software Reliability Estimation*; Safecomp 99, Lecture Notes in Computer Science, Vol. 1698 LNCS, Springer-Verlag Heidelberg

Littlewood, B. (2013) *The Problem of Assessing Software Reliability …when you really need to depend on it*; no date, no written source, from internet 2013

Littlewood, B. and L.Strigini, (1993). *Validation of Ultra-high Dependability for Software-based Systems*, Communications of the ACM, 36(11)

Littlewood, B. and D. Wright (1997). Some Conservative Stopping Rules for the Operational Testing of Safety-Critical Software, IEEE Transactions on Software Engieering, Vol 23, No 11 November

May, J., S. Kuball and G. Hughes c (1999). *Test statistics for system design failure*; International Journal of Reliability, Quality and Safety Engieering, Vol 6, No3, pp. 249-264

Saifuddin, A. (2009). *Methods in Survey Sampling Biostat 140.640* – Lecture4StratifiedSampling.pdf, John Hopkins University, Bloomberg, school of public health (from internet 2013)

Soehnlein, S., F. Saglietti, F. Bitzer, M. Meitner and S. Baryschew (2010). Software Reliability Assessment Based on the Evaluation of Operational Experience, Proc. 15th International GI/ITG Conference on Measurement, Modelling and Evaluation of Computing Systems, Dependability and Fault Tolerance (MMB & DFT 2010), Lecture Notes in Computer Science, Vol. LNCS 5987, Springer-Verlag

APPENDIX A
COLLECTION OF PRE-REQUISITES AND ASSUMPTIONS

The following rules, requirements and assumptions are not in systematic order, as their criticality is usually project dependent.

R1 The code of the pre-existing version(s) and the code of the version for the future application shall be identical.

R2 No failures must occur during pre-operation.

R3 Sequence and number of runs of any path must not influence any future run.

R4 The distribution of input data that are processed in one path is approximately equal between the experience gathering period and the future operation period.

R5 If the operational experience is simulated by tests, the individual test runs must be independent from each other.

R6 Observation of information gathering is so strict and complete that any possible failure is recognised.

R7 A specification existed that allowed to decide whether or not any result was correct of incorrect.

R8 The paths shall be identified for the old and the future demand profile.

R9 For estimating the demand profile for a new application model checking can perhaps help.

R10 Each path shall have at least 2 runs.

R11 The $\tilde{p}_i$ *and* π_i shall be evaluated or conservatively estimated for each path i.

R12 Concatenation of paths or modules is allowed, if they do not interact; in this case the largest $\tilde{p}_i$ of the chain shall be taken.

R13 If interacting modules are concatenated, the paths from start to end shall be taken.

R14 Paths whose correctness has been proven, may be counted with $p_i = 0$.

R15 It is recommendable to verify the effect of loops with varying repetition number deterministically.

R16 Separate considerations are required for complicated logical expressions or for complicated algorithms.

R17 Some aspects, e.g. events that shall be triggered by the software at a specific future date, need deterministic verification and white box testing.

R18 During the pre-use phase where the experience is gathered, no failure masking is allowed. See e.g. (Bishop 1987)