Paper 5.

Robert Feldt. *Forcing Software Diversity by Making Diverse Design Decisions - an Experimental Investigation*, Technical Report no. 98-46, Department of Computer Engineering, Chalmers University of Technology, Gothenburg, Sweden, December 1998.

Forcing Software Diversity by Making Diverse Design Decisions – an Experimental Investigation

Robert Feldt
Department of Computer Engineering
Chalmers University of Technology
S-412 96 Göteborg, Sweden
Tel: +46 31 772 5217, Fax: +46 31 772 3663
E-mail: feldt@ce.chalmers.se

Abstract

When developing software versions for a multi-version system, the probability for coincident failures may be decreased by forcing the development efforts to be different by making diverse design decisions. There are theorems showing that the probability is minimized by making as diverse design decisions as possible but it is not known if the assumptions made in proving the theorems are valid in practice. To investigate this we have developed 435 versions of a software controller for an aircraft braking system. The versions were developed using genetic programming. Analyses of the failure behavior of these versions showed that the assumptions of failure independence among the decisions were valid, on average, for 74% of the test cases. The assumption of indifference between methodologies were not valid in a single case which seems to be the major cause invalidating the theorem. Thus, if we are not indifferent between design decisions, it is not guaranteed that increased diversity of design decisions will decrease the probability of coincident failures.

1. Introduction

N-version programming (NVP) has been proposed as a technique to develop multiple versions of the same software for fault-tolerant software systems [Avizienis95b]. In the case in which the versions fail independently of each other the reliability of a multi-version system could be significantly larger than the reliabilities of single versions. However, empirical and theoretical results have shown that independent failure behavior can not be expected; apart from the difficulties of making the development efforts independent varying difficulty of the input data guarantees that the versions will not fail independently [Knight86] [Bishop95]. In [Littlewood89], Littlewood and Miller proved theorems showing that the attainable reliability levels can be higher than if independence is assumed if we force the development efforts to be different. Thus, by actively making diverse design decisions in the different development efforts we can force software diversity, i.e. diversity in the structure and / or failure behavior of the resulting software ver-

sions.

The theorems of Littlewood and Miller further indicate how the diversity of the development efforts should be forced: the design decisions should be as diverse as possible to decrease the probability of coincident failure [Littlewood89]. In order to prove the theorems about the effectiveness of forced design diversity, Littlewood and Miller made a number of assumptions. They pointed out that these assumptions might not be valid in practice. In this paper we study if these assumptions are valid on a particular application and assess how the validity of a theorem of the effectiveness of forced diversity is affected when the assumptions are not fully valid.

To carry out these analyses in a statistically rigorous way we need failure data from a large number of versions developed with a number of different design decisions. The cost of conducting such an experiment would be prohibitively high. In a previous study we have introduced a procedure for forced design diversity using genetic programming to develop the software versions [Feldt98b]. Genetic programming (GP) is a technique for searching for computer programs with desirable properties. In the previous study we showed that failure diversity can be forced by varying parameters to a GP system and proposed that this technique can be used as a research tool in software fault-tolerance [Feldt98a]. In this paper we have used the technique as a tool to develop software controllers for an aircraft braking system.

In section 2 we recapitulate the results on forced diversity by Littlewood and Miller and state the research questions we investigate in this study. Section 3 describes the method we have used; it introduces genetic programming, the target application and the experiments we have carried out. The results from the experiments are given in section 4 followed by a discussion in section 5. Finally, we summarize the conclusions that can be drawn from this study and indicate future work.

2. Forced design diversity

In their seminal paper, Littlewood and Miller extended the model of multi-version program development introduced by Eckhardt and Lee, to account for diverse development methodologies [Littlewood89]. In the model, a development methodology is characterized by a mapping, S, giving the probability that a particular program, π, is chosen from the population of all possible programs, $\wp$, i.e.

$$P(\Pi = \pi) = S(\pi)$$

where Π is a random variable representing the random selection of a program. A key average performance measure is $\theta(x)$, giving the probability that a randomly chosen program fails for the input case x (the model can handle varying probabilities of input cases but we do not consider this in the present study; we assume that all input cases are equally probable). For a randomly chosen input X, $\theta(X)$ is a random variable Θ.

To model diversity between development methodologies Littlewood and Miller introduced the notion of a *design decision*. An example of a design decision would be the choice of testing strategy. In general, design decisions can have multiple levels but we focus on binary decision and associate '0' and '1' with the two possible outcomes of such decisions.

A development methodology is defined by the outcomes of a number of design decisions and can be described with a binary vector. For example, in the experiments in this paper there are seven binary design decisions and methodology number nine is defined by the descriptor

(0, 0, 0, 1, 0, 0, 1)

A natural measure of design diversity, i.e. the degree of diversity between development methodologies, when using these binary vectors is Hamming distance. We thus define

> the *developmental diversity*[1] between two methodologies M_1 and M_2 , denoted
> $\delta(M_1 , M_2)$, as the number of positions in which the descriptors of the methodologies differ, i.e. their Hamming distance.

The developmental diversity can be used to state an important theorem on forced diversity (theorem 3 in [Littlewood89]) in natural language as: "The greater the developmental diversity between two methodologies, the lesser chance for coincident failures of two versions developed with the methodologies". As Littlewood and Miller point out the theorem holds on average, when we consider all the possible programs that might be developed with a methodology and check the behavior on all possible input cases; for particular input or programs it might not hold.

The above theorem is important since it implies a rule for how to develop multi-version software. If we choose methodologies that have maximum developmental diversity, the probability of coincident failures is decreased and the reliability of the system should increase.

The proof of the theorem relies on the Cauchy-Schwarz inequality and the following three assumptions (for a pair of binary design decisions):

- A1, *Decision choice independence*: The choice taken in the different design decision should be independent of each other, so that
  $$P(\Pi \in (1,1)) = P(\Pi \in (1,*)) \cap P(\Pi \in (*,1))$$
  where * is used to mark all possible outcomes of the decision.
- A2, *Decision failure independence*: There is no interaction between the design decisions in their effect on the failure behavior, so that
  $$P(\pi \in (1,1) \mid \pi \text{ fails on } x) = P(\pi \in (1,*) \mid \pi \text{ fails on } x) \cap P(\pi \in (*,1) \mid \pi \text{ fails on } x)$$
  for all input cases x.

[1] We use this term to distinguish it from the more general notion assigned to 'design diversity'.

- A3, *Methodology indifference*: We are indifferent between methodological choices related by permutations of labels. Two specific instances are used in the proof:
  - A3a, Indifference for methodologies with developmental diversity 0:
    $E(\Theta_{11}, \Theta_{11}) = E(\Theta_{01}, \Theta_{01}) = E(\Theta_{10}, \Theta_{10}) = E(\Theta_{00}, \Theta_{00})$
    where $E(\Theta_{M1}, \Theta_{M2}) = P(\Pi_{M1}$ and Π_{M2} both fail on X) = probability that randomly chosen programs from two methodologies both fail on a randomly chosen input.
  - A3b, Indifference for methodologies with developmental diversity 1:
    $E(\Theta_{11}, \Theta_{01}) = E(\Theta_{11}, \Theta_{10}) = E(\Theta_{00}, \Theta_{10}) = E(\Theta_{00}, \Theta_{01})$

In their paper, Littlewood and Miller point out that the assumptions they make in proving their theorems might not be valid in practice. In this study we assess the assumptions empirically. Furthermore, we want to investigate how the validity of the theorem is affected when there are deviations from the assumptions above. The questions we want to answer in this study can thus be summarized as (for an application X):

1. Is the assumption of failure independence between factors (A2 above) valid for the application X?
2. Are the assumptions of indifference (A3a and A3b) valid for the application X?
3. If the assumptions (A2 or A3) are not fully valid, how does this affect the validity of the theorem? Is it still valid but in a restricted form? Which of the assumptions affect the validity of the theorem the most?

We do not investigate assumption A1 since we can make sure that it is fulfilled by the choice of design decisions.

3. Method

We have developed 435 versions of a software controller employed in an aircraft braking system. The versions were developed with a genetic programming system running on a SUN Enterprise 10000 with the Sun Solaris OS 2.5. The experiment environment, consisting of the genetic programming system GPSys, a simulator of the application and custom-developed software to interface between them was developed in Java and compiled with the java-to-C compiler Toba for increased performance [Quereshi98] [Proebsting96] [Feldt98a]. After development all software versions were subjected to the same 10000 test cases. The failure behavior on these test cases was analyzed to answer the research questions of this study.

Below we give a brief introduction to genetic programming and describe the application and the experiment we have conducted. Further details on the application can be found in [Feldt98a] and [Christmansson98]. Figure 1 shows an overview of the experiment environment

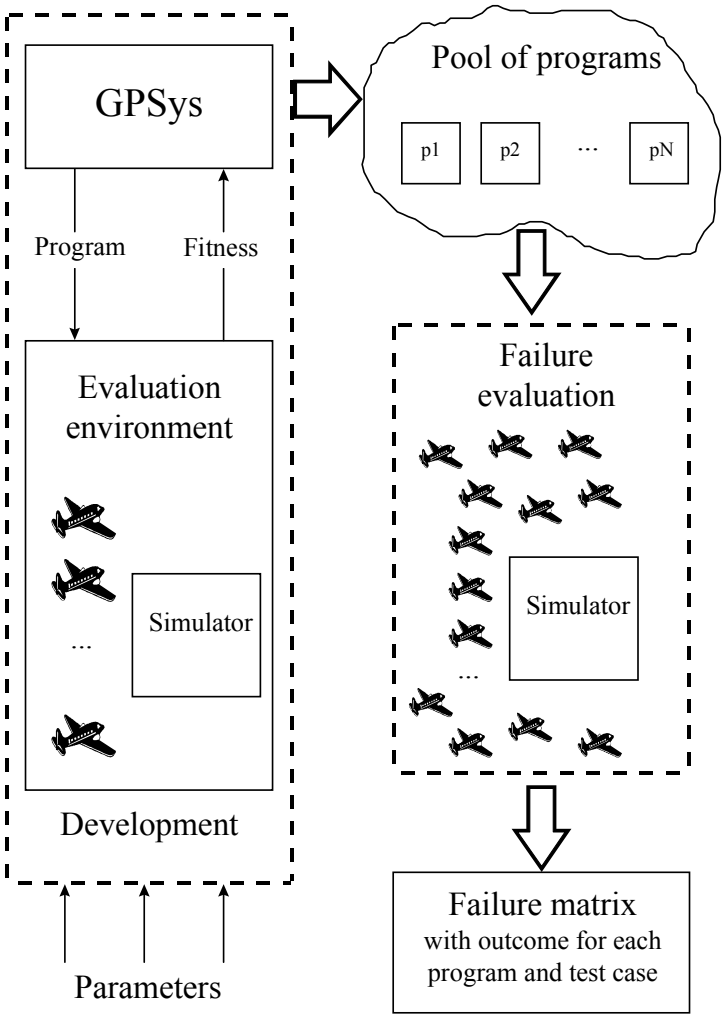

Figure 1. The experiment environment

3.1 Genetic programming

Genetic programming is an algorithm for searching for computer programs with desirable properties. The 'genetic' part of its name is used because the search process shows some resemblance with evolutionary processes in biological systems. The technique was introduced in 1992, is studied under the heading of Evolutionary Computation and have been successfully applied on a number of different problems [Koza92] [Banzhaf, *et al.*98] [Bäck97]. Using biologically inspired ideas in the research and design of fault-tolerant computers has been previously proposed in [Avizienis95a].

The search space searched by a genetic programming (GP) algorithm can be defined by

the user. Parameters govern which basic building blocks, such as variables, constants and functions can be used in the programs. Other parameters govern how the search should proceed in the search space and how the programs should be tested during development. The tests carried out during development are called fitness evaluations; they assess how 'fit' each program is, i.e. how well it adheres to the specification.

Genetic programming is a parallel search procedure having multiple programs that are used as starting points for further explorations of the search space. In each iteration of the algorithm the fitness of all the programs is evaluated. Programs are selected based on how fit they are and combined into new programs by swapping parts between them. Old programs with low fitness are deleted. When this scenario is repeated there is a tendency that, after a large number of iterations, programs with increasingly higher fitness are found. The search procedure is stochastic by nature and running it repeatedly can give different results.

In a previous study we have proposed that diverse software versions could be developed by systematically varying parameters to a GP system [Feldt98a]. In this proposal, the choice of the values of a parameter is analogous to a design choice in Littlewood and Millers model. In an empirical investigation we showed that diverse failure behavior was obtained even though it was limited for the parameter settings giving the lowest failure rate. For more information on these results and on GP in general see [Feldt98a] [Banzhaf, *et al.*98].

3.2 Application

The target application is designed to arrest aircrafts on a runway. Incoming aircraft attach to a cable and the system applies pressure on two drums of tape attached to the cable. A computer that determines the brake pressure to be applied controls the system. By dynamically adapting the pressure to the energy of the incoming aircraft the program should make the aircraft come to a smooth stop. The requirements on a system like this can be found in [US Air Force86]. The system has been used in previous research at our department and a simulator simulating aircraft with different mass and velocity is available. The system is more fully described in [Christmansson98].

The main function of the system is to brake aircraft smoothly without exceeding the limits of the braking system, the structural integrity of the aircraft or the pilot in the aircraft. The system should cope with an aircraft having maximum energy of $8.81*10^7$ J and mass and velocity in the range 4000 to 25000 kg and 30 to 100 m/s, respectively. More formally the program should[2] (name of corresponding failure class in parentheses)

- stop aircraft at or as close as possible to a target distance (275 m)
- stop the aircraft before the critical length of the tape (335 m) in the system (OVERRUN)

[2] Our system adopts the requirements of [US Air Force86] with the addition of the allowed ranges for mass and velocity and a critical length of 335 m (950 feet in [US Air Force86]).

- not impose a force in the cable or tape of more than 360 kN (CABLE)
- not impose a retarding force on the pilot corresponding to more than 2.8g (RETARDATION)
- not impose a retarding force exceeding the structural limit of the aircraft, given for a number of different masses and velocities in [US Air Force86] (HOOKFORCE)

The programs are allowed to use floating point numbers in their calculations. They are invoked for each 10 meters of cable and calculate the brake pressure, for the following 10 meters, given the current amount of rolled out cable and angular velocity of the tape drum.

An existing simulator of the system has been ported from C to Java. It implements a simple mechanical model of the airplane and braking system and calculates the position, retardation, forces and velocities in the system. It does not model the inertia in the hydraulic system or oscillatory movement of the aircraft due to elasticity in the tape. The simulator has been set to simulate braking with a time step of 62.5 milliseconds.

During development of the programs GPSys invokes the simulator and tests the programs on a number of test cases, i.e. aircraft with different masses and velocities. Critical values from each braking are used to evaluate if a program has violated any of the requirements above. A penalty is assigned if there is a violation for any of the failure classes above.

3.3 Experiments

To answer the research questions stated in section 2 we need to choose two design decisions and develop programs with the resulting four methodologies. The outcome of such an experiment could be highly dependent on the particular choice of decisions. To lessen this sensitivity to design choices we have identified a group of seven design choices and studied all C(7,2) = 21 combinations of two choices from them. The seven design choices were chosen based on our previous experience with the experiment environment; we excluded choices with a negative effect on the failure rate of the resulting programs. This is similar to what we would do if we were to develop multiple versions using human software developers; we would not take design decisions that are known to give high failure rate just to increase the diversity of the software versions.

The design choices and their levels, i.e. the outcomes of the decisions, are shown in table 1. Decision number one concerns the testing performed by the GP system during evolution of the programs. When on level zero the test cases are equidistantly spread on the allowed range of mass and velocity of the incoming airplane. On level one the values are randomly assigned from a uniform distribution.

Design decision	Level	Description
1	0	36 uniformly spread test cases are used to evaluate fitness.
	1	25 randomly sampled test cases are used to evaluate fitness.

2	0	A guiding penalty of 2% of maximum penalty is used.
	1	A guiding penalty of 50% of maximum penalty is used.
3	0	Maximum penalty on the HALTDISTANCE failure criteria is 1000.0.
	1	Maximum penalty on the HATLDISTANCE failure criteria is 3000.0.
4	0	Programs can use the pressure at the previous checkpoint and the indices to the current and previous checkpoints.
	1	Programs can use the current time since start of the braking, the angular velocity and time at the previous checkpoint
5	0	Programs cannot use any subroutines.
	1	Programs can use a subroutine.
6	0	No effect.
	1	Programs can use the statement IF, and operators LE, AND and NOT.
7	0	Programs can use the function SIN and the terminal PI (3.1415).
	1	Programs can use the function EXP.

Table 1. The design decisions and their effect

Decisions two and three determine the fitness function used to rank the programs during development. For this particular application, the fitness function is calculated as a penalty assigned as to how much a certain requirement is violated. The fitness score is a sum of penalties on four criteria, with a penalty of 1000 units assigned when the requirement is violated, i.e. the program fail on the criteria. Decision 2 sets the level of the guiding penalty assigned to grade how far from fulfilling the requirement the programs are. This 'guides' the development in the direction of requirement fulfillment. Decision 3 defines the maximum penalty on the HALTDISTANCE criteria. When on its high level, it indicates that we consider the HALTDISTANCE criteria to be the most important criteria. The programs have something to gain from trying to fulfill this criterion with high priority.

Decision 4, 6 and 7 govern which variables (4) and functions (6 and 7) the programs can use. Decision 5 governs the structure of the programs. When at its high level the programs can use a subroutine.

Twenty-nine different methodologies were constructed from these seven design decisions. They are all the twenty-one methodologies with all pairs of two decisions chosen from the seven at their high level, the seven methodologies with each single decision at it's high level and the methodology with all decisions at its low level. Decisions that were not involved in deciding the methodology were held at their '0' level. This choice of methodologies allows us to test the validity of the theorem for the 21 different pairs of design decisions. Each methodology was used to develop 15 versions resulting in a total of 435 versions and each version were tested on the same 10000 test cases. The test cases were equidistantly spread on the allowed ranges for mass and velocity.

4. Results

In this section we describe the results from the analysis of the failure behavior of the 435 programs. We analyze, in turn, assumption A2, A3 and the validity of the theorem. This corresponds to research questions 1, 2 and 3 in section 2.

4.1 The assumption of decision failure independence (A2)

For each pair of decisions we can test assumption A2 four times: one for each combination of decisions 00, 01, 10 and 11. For each of these 21*4 = 84 tests we collected the failure data of the 4*15 = 60 versions relevant to the test. For each test case where at least one variant failed we calculated the dependency ratio

$$r_d = P(\Pi \in (1,1)) \,/\, (P(\Pi \in (1,*)) \lceil P(\Pi \in (*,1)))$$

which according to assumption A2 should be exactly one (we assigned the value one when both numerator and denominator were zero).

Table 3 below gives some summary statistics for the 84 tests. Note that the maximum percentage of ratios equal to 1 were 79.4%.

	Test cases with failure(s)	$r_d < 1$	$r_d = 1$	$r_d > 1$	Average r_d	Stdev of r_d
Max	6087	23.9%	79.4%	23.9%	1.0936	0.3889
Average	3955	12.9%	74.2%	12.9%	0.9993	0.2606
Min	2459	4.2%	69.6%	4.2%	0.9192	0.1764

Table 3. Summary statistics for the 84 tests of assumption A2

Only one pair of decisions had a balanced distribution of dependency ratios; the rest were either skewed to the right or to the left. However, when all ratios were considered together the distribution was balanced with equal number of ratios below and above independence at one.

4.2 The assumption of methodology indifference (A3)

To check assumption A3 we calculated the $\theta(x)$-vectors for all 29 methodologies. For our experimental set-up assumption A3a implies that all the expectations

$$E(\Theta_M, \Theta_M)$$

where M is a methodology, should be equal. In our experiment, they are not. The average, minimum, maximum and standard deviation of these expectations are shown in table 4.

Max	0.0636
Average	0.0509
Min	0.0371

Standard deviation	0.0057

Table 4. Summary statistics for 29 expectations between methods with developmental diversity 0

Assumption A3b implies that all expectations

$$E(\Theta_{M1}, \Theta_{M2}) \text{ where } \delta(M_1, M_2) = 1$$

should be equal. In our experiment, they are not in a single case. There are 49 expectations of this kind and their summary statistics are shown in table 5.

Max	0.0573
Average	0.0482
Min	0.0379
Standard deviation	0.0042

Table 5. Summary statistics for 49 expectations between methods with developmental diversity 1

4.3 Validity of theorem

The theorem of Littlewood and Miller gives an ordering between groups of expectations having equal developmental diversity. For each pair of decisions there are three groups corresponding to the developmental diversities of 2, 1 and 0. The first group contains two elements and the latter groups four elements each. For each pair of decisions, we tested if the ordering of the theorem was valid. We also compared the averages of and the minimum expectation in the different groups. The results from these comparisons are shown in table 6.

	$\delta = 2$ vs. $\delta = 1$	$\delta = 1$ vs. $\delta = 0$
Theorem: Max < Min	0 (0.00%)	0 (0.0%)
Average < Average	17 (80.95%)	21 (100.00%)
Min < Min	5 (23.81%)	12 (57.14%)

Table 6. Number of cases where comparisons between groups of equal development diversity are valid

When we grouped the different expectations in the respective groups together and performed an analysis of variance there were statistically significant differences between the groups ($p < 0.01$). The average expectations for the three groups were 0.0476, 0.0482 and 0.0509 respectively. Thus, the ordering prescribed by the theorem is valid "on average". The ANOVA table is shown below.

Source	Sum of squares	Degrees of freedom	Mean square	Ratio	Significance

Groups with equal dev diversity	$1.98*10^{-4}$	2	$9.91*10^{-5}$	5.88	0.0037
Residuals	$1.97*10^{-3}$	117	$1.68*10^{-5}$		
Total	$2.17*10^{-3}$	119			

Table 7. Analysis of variance table

To assess the individual effects of the different assumptions we repeated the test on the validity once more, this time only including testcases where assumption A2 was valid. The test cases to include were chosen individually for each pair of design decisions. The ordering prescribed by the theorem was not valid in any of the 42 cases. The full result of these comparisons is shown in table 8.

	$\delta = 2$ vs. $\delta = 1$	$\delta = 1$ vs. $\delta = 0$
Theorem: Max < Min	0 (0.00%)	0 (0.00%)
Average < Average	18 (85.71%)	21 (100.00%)
Min < Min	6 (28.57%)	12 (57.14%)

Table 8. Result of comparisons when expectations are calculated on test cases where assumption A2 holds

5. Discussion

We have separated the discussion of the results in two parts. In the first we consider the results without acknowledging the fact that the programs have been developed with genetic programming. In section 5.2 we discuss how this fact might affect the validity of our results.

5.1 Implications of the results for forced design diversity

The results tell us that, for this particular application and with this particular choice of design decisions, the theorem on the effectiveness of forced diversity is not valid. However, this should not be expected since its assumptions are not fulfilled.

Assumption A2 was on average valid in about 74% of the test cases where at least one version failed. Furthermore, a majority of methodologies had skewed distributions with dependency ratios as high as 1.09 and as low as 0.91. This supports the intuitive notion pointed out by Littlewood and Miller that assumption A2 can not be expected to be valid in practice since there will be interactive effects between the design decisions. An example scenario of this would be if one design decision would be between programming languages where only one of them supports automatic garbage collection and memory handling (such as between C and Java) and a decision on testing governing if the tool `purify` is used to analyze the memory behavior during execution. An interactive effect be-

tween these design decisions seems plausible.

In our experiment, assumption A2 does not seem to be a major cause of theorem invalidity. When we tested the validity of the theorem and forced assumption A2 to be fulfilled this did not increase the level of theorem validity. This indicates that it might not be crucial for this assumption to be fulfilled. However, more experiments are needed to settle this.

If A2 is not the major cause of the invalidity of the theorem then this must be due to the unfulfillment of A3, the assumption of indifference. For all of the 21 pairs of design decisions and all their three groups of methodologies with equal developmental diversity there is some spread of the expected failure probabilities. Thus, the assumption of indifference is not valid in a single case. The expectations in each group are spread out so that there is some overlap between the groups. This causes the invalidity of the ordering that would be seen if the theorem would be valid.

It is interesting that the theorem is still valid in a majority of cases if we look at the averages of each group of developmental diversity. Analytical investigations of this might reveal extensions of the theorem that need not assume strict indifference. One possible way to go would be to express the theorem in terms of the amount of variation between expectations in the same groups. Such investigations would be important since it seems unlikely that we, in practice, will be strictly indifferent between methodologies. Interactive effects will likely invalidate the indifference assumption.

These results complicate the picture for practical development of multi-version systems. If indifference is typically not the case, larger developmental diversity will not guarantee smaller chance of coincident failure; variations among the combinations of methodologies with equal developmental diversity can alter the ordering. Further empirical and analytical investigations of these questions are needed to clarify the picture.

We would like to stress an important point stated by Littlewood and Miller: in the case of ignorance of the effect of different decisions the assumption of indifference will be plausible and, for this "state of knowledge", the theorem will be valid. However, over several projects developers might build up experience on the effects of different design decisions that can be used to take more informed design decisions. Extending the theorems of forced design decisions to guide these decisions seems worthy of future studies. One result in this direction were conjectured by Littlewood and Miller and a natural extension of this study would be to assess this conjecture and how it can be used when we have knowledge on the effect of different design decisions. Another important task for research on forced diversity will be to investigate the effect of different design decisions on diversity.

The model of multi-version development used by Littlewood and Miller uses the set of all possible programs that could be developed with a methodology and the set of all possible input cases to the programs. In our experiment we have sampled from these sets and all our results includes uncertainty as to these samples. We are confident that the sample from the input cases does not affect the results. In previous experiments with this application the qualitative results were not affected when the number of test cases was increased; only minor alterations in the quantitative results occurred. This could be due to the fact

that our input space is simply a two-dimensional rectangle of the plane and we can cover it sufficiently good with a relatively small number of test cases. Furthermore, our application is a fairly simple controller with "continuos" characteristics. However, for applications with more complicated input spaces this would be an important issue.

To assess the effect of the sampling of the programs we repeated all the calculations and analyses reported above for different numbers of programs from each methodology in the range from 4 up to 15. This showed a convergence to the results reported above at the number of 7. This indicates that the results have converged and will not be seriously affected by increasing the size of the sample any more. However, a full analysis taking statistical inference into consideration at each step would be valuable.

5.2 Using genetic programming to investigate design diversity

Genetic programming is different from ordinary software development. It can be questioned if the results and analyses reported here are valid for software developed by humans. Our stance is that statistical theorems do not differentiate between programs because they have been developed using different techniques; the theorems should be valid in general.

There are a number of advantages of using an automated technique, such as genetic programming. A large number of versions can be developed at a relatively low development cost and parameters can be systematically varied to emulate different design decisions. These characteristics are important to get statistically significant results.

Of course, there are also a number of disadvantages of using genetic programming. The failure probabilities are typically higher than would be the case if we developed the software by hand. It is possible that this affects the results so that other behavior will be observed for lower failure probabilities. Furthermore, genetic programming has mostly been applied to smaller problems and the applicability of our approach is directly tied to the applicability of GP. If GP can not be used on larger and more complex problems then neither can our approach. These issues are discussed in more detail in [Feldt98a]. To settle them studies that compare the use of genetic programming with ordinary software development are needed. We are in the process of approaching researchers in the fault-tolerance area that have conducted studies with replicated-run replicated-variant experiments to try to make such studies possible [Knight86] [Kelly83].

In summary, we think the technique of developing multiple software versions using genetic programming can be used to investigate the theoretical limits of diversity and multi-version software. It can be used as a research tool to explore new possibilities and increase the knowledge on software and design diversity. This could help minimize the cost and increase the effectiveness of conducting research on multi-version software using human software developers.

6. Conclusions

We have developed 435 program versions of a software controller braking aircraft coming in to land on a runway. The versions were developed using an automated program

searching technique called genetic programming. By varying parameters to the genetic programming algorithm we obtained 29 different development methodologies and developed 15 versions with each of these methodologies. By analyzing the failure behavior of the programs we tested the validity of a theorem indicating how to force diversity, stating that larger diversity between development methodologies, what we call developmental diversity, gives a smaller probability of coincident failures [Littlewood89].

We tested two assumptions that need to be fulfilled for the theorem to be valid: the assumption of independent failure behavior between design decisions (A2) and the assumption of indifference (A3). Assumption A2 was fulfilled in average on about 74% of the test cases while assumption A3 was never fulfilled. In accordance with this the theorem was only valid in part of one case (2.35% of the possible orderings). The major cause of theorem invalidity seems to be that assumption A3 is not valid. Thus, if we are not indifferent between design decisions, it is not guaranteed that increased diversity of design decisions will decrease the probability of coincident failures. An example of this situation would be if we suspect that two design decisions have an interactive effect; in this situation it is unclear if maximally forced diversity will give increased reliability.

When we considered averages of the groups of failure probabilities for methodologies with equal developmental diversity, the orderings predicted by the theorem was valid in a majority of cases. However, our results show that if we are not indifferent between combinations of design decisions there can be no theorem stating that the smallest probability of coincident failures will be in the group with highest developmental diversity. There can be variations within the groups that invalidate orderings of them. Analytical and / or empirical investigations into the possibility of extending the theorem based on the amount of variation in the groups would be valuable.

It may not be the case that these results affect the actual strategies that should be used when developing multi-version systems. As was pointed out by Littlewood and Miller the assumptions they make in proving their theorem are plausible given that we do not *know* the effect of different design decisions. Our results do not invalidate this and in a state of ignorance about the effects of different decisions the assumption of indifference will be plausible and the theorems valid.

Our results have been obtained for one application. We need to investigate more applications to evaluate the generality of our results. It would be especially interesting to make comparative studies with previous experiments on multi-version software. In addition, this would make it possible to assess how software developed using genetic programming differs from software developed by humans. There are large dissimilarities between the two that could question the validity of our results. However, genetic programming should be a valuable tool in investigating the theoretical limits and theorems of diversity; versions are not treated differently depending on how they have been developed.

Genetic programming is a computational technique inspired by biological processes. It is the author's opinion that many new and interesting ideas for building and conducting research on fault-tolerant computer systems can be found by studying nature and biological systems as was provoked in [Avizienis95a].

Acknowledgement

The author wishes to acknowledge Jörgen Christmansson, Marcus Rimén, Martin Hiller, Jan Torin and Håkan Edler whose comments increased the quality of this paper.

References

Avizienis, A. "Building Dependable Systems: How to Keep Up With Complexity." Proceedings of the Fault-Tolerant Computing Symposium 25th Silver Jubilee (FTCS-25): 1995a. 4-14.

---. "The Methodology of N-Version Programming." Software Fault Tolerance. editor M. Lyu. Chichester, England: John Wiley & Sons, 1995b. 23-46.

Bäck, T., U. Hammel, and H-P. Schwefel. "Evolutionary Computation: Comments on the History and Current State." IEEE Transactions on Evolutionary Computation 1.1 (1997): 3-17.

Banzhaf, W., et al. Genetic Programming - an Introduction. San Fransisco, California: Morgan Kaufmann, 1998.

Bishop, P. "Software Fault Tolerance by Design Diversity." Software Fault Tolerance. (ed.) M. Lyu. Chichester, England: John Wiley & Sons, 1995. 211-30.

Christmansson, J. "An Exploration of Models for Software Faults and Errors.". Chalmers University of Technology, 1998.

Feldt, Robert. Using Genetic Programming to Systematically Force Software Diversity. Gothenburg, Sweden: Chalmers University of Technology, 1998a. 296L Thesis for the degree of Licentiate of Engineering.

---. "Generating Multiple Diverse Software Versions Using Genetic Programming - an Experimental Study." IEE Proceedings - Software (1998b).

Kelly, J. P. J., and A. Avizienis. "A Specification-Oriented Multi-Version Software Experiment." Proceedings of the 13th Fault-Tolerant Computing Symp. (FTCS-13): 1983. 120-26.

Knight, J. C., and N. Leveson. "An Experimental Evaluation of the Assumption of Independence in Multiversion Programming." IEEE Transactions on Software Engineering 12.1 (1986): 96-109.

Den klassiska empiriska stuiden med 27 varianter av Missile Launch Interceptor utvecklade av student-team på 2 personer. Mycket bra artikel med väl diskuterade forskningsresultat.

Koza, J. Genetic Programming - on the Programming of Computers by Means of Natural Selection. Cambridge, Massachuchetts: MIT Press, 1992.

Littlewood, Bev, and Douglas R. Miller. "Conceptual Modeling of Coincident Failures in Multiversion Software." IEEE Transactions on Software Engineering 15.12 (1989): 1596-614.

Proebsting, T., et al. "Toba: Java for Applications - A Way Ahead of Time (WAT) Compiler". Technical report, University of Arizona, 1996.

Quereshi, A. GPSys 1.1 Homepage. Web Page. URL: http://www.cs.ucl.ac.uk/staff/A.Qureshi/gpsys.html. 20 November 1998.

US Air Force. Military Specification: Aircraft Arresting System BAK-12A/E32A; Portable, Rotary Friction. 1986. MIL-A-38202C, Notice 1.