The Performance of the N-Fold Requirement Inspection Method

Kantorowitz, Eliezer
Guttman, Arie
Arzi, Lior

Computer Science Dept.
Technion - Israel Institute of Technology
32000 Haifa, Israel

email: kantor@cs.technion.ac.il

ABSTRACT

Faults in system requirements can be very harmful. It is therefore often required that the inspection achieves a high *fault detection rate* (FDR). To achieve this, a large number of inspectors is required. Large teams are known to be inefficient. Therefore, the N-fold requirements inspection method divides the inspectors into *N* small efficient teams. All teams inspect the same requirements document. Experiments with both information and real-time systems demonstrate that the different teams detect different faults such that they achieve together a higher FDR value. The analysis suggests that the FDR is primarily a function of the *level of expertise* of the inspectors and of the number of teams. A quite simple probabilistic model that matches the experimental results enables the prediction of the FDR as a function of these two parameters. A diagram based on the model enables a fast estimation of the FDR and of the most effective number of inspections teams money-wise. The model may also be employed for measuring the efficiency of requirement inspection methods.

KEYWORDS: requirements, inspection, n fold inspection, fault detection rate, probabilistic model

INTRODUCTION

The requirement analysis phase of the software development process results in a user requirements document (URD). It is well know that faults in the URD can result in high development costs, delays in product delivery and loss of reputation. This applies specially to mission critical software for which a high level of correctness is required. Fairley[1] presents cases in which it was 5 times more costly to correct a fault at the design stage than during initial requirements analysis, 10 times more costly to correct it during coding, 20 to 50 times more costly to correct it at acceptance testing, and 100 to 200 times more costly to correct problems that surfaced during program operation. It is therefore worthwhile to invest in detecting and removing faults from the URD before taking further steps in the program development process. A common method for such a URD validation is the formal inspection. An overview and bibliography of inspection methods is found in [2]. It has been observed that many errors are not detected by inspection [3]. Johnny Martin and W. T. Tsai therefore proposed the N-fold inspection method [4], in which the formal inspection is repeated by *N* different teams of inspectors. It is expected that different eyes see different things, and that the *N* different teams will, therefore, together detect more faults than a single large team.

In the first phase of the fault detection process, which is called the *preparation* phase [10], each inspector inspects a part of the URD separately. In the second phase, which is called the *collection* phase, all the inspectors of the team meet and collect the faults found by the individual inspectors. This collection process involves a critical review of the faults found in the preparatory phase. Some of the faults suggested by the individual inspectors may prove not to be faults, and are therefore removed. New faults may be detected in the collection meeting through group interaction. The efficiency of the team in the collection phase in terms of the number of person-hours per detected inspector mistake and per detected new fault is a function of the size of the team. The efficiency of teams as a function of the number of their members has been thoroughly studied by psychologists [8] [9]. The conclusion is that teams of 2-3 members are most efficient. Large teams are quite inefficient. In a meeting with many members, only few are active while the rest are quite passive. The efficiency of the N-fold inspection method is thus based on the use of small efficient teams and on the expectation that the different teams will detect different faults. These expectations were met by the experiment [4], which is denoted Experiment 1 in this paper. This experiment was made with a URD for a large-scale real-time centralized railroad traffic control system. The 22 page long URD was prepared with the help of a railroad software expert. 99 faults, belonging to 6 different categories, were entered in this URD.

The method of entering faults is quite common in testing. The assumption is that if the inspectors detect x percent of the entered faults, then they have also detected x percent of the total number of faults. The validity of this assumption depends on the degree to which the entered faults are representative of the real faults. The faults entered in experiment [4] were designed to meet this requirement. The participants in the experiment were computer engineering students from the University of Minnesota in their 4th year of study. The students were divided into 9 teams, each having three students. Each team had 6 hours to review the faulty URD. The result was, as expected, that a larger number of inspection teams detected more faults than a smaller number of teams.

The experiment was repeated later [5] and similar results were obtained. This experiment will appear in the paper to be denoted as Experiment 2. Results of Experiments 1 and 2 suggest that the number of detected faults is a monotonically increasing function of the number of inspection teams. It may be useful for a manager of a software project to know this function before starting the inspection. If the costs of an inspection team and the costs of undetected faults are known, the manager may employ this function to determine the optimal *N* value. This paper studies this function experimentally. In order to be able to compare the results of different experiments we introduce a normalization of the function by considering the Fault Detection Ratio (*FDR*). i.e. the number of detected faults divided by the total number of faults, as a function of *N*. This normalized function *FDR(N)* will be called the performance of the N-fold requirement inspection method for a given URD and for a given type of inspection team. Figure 1 shows the performance observed in Experiment 2.

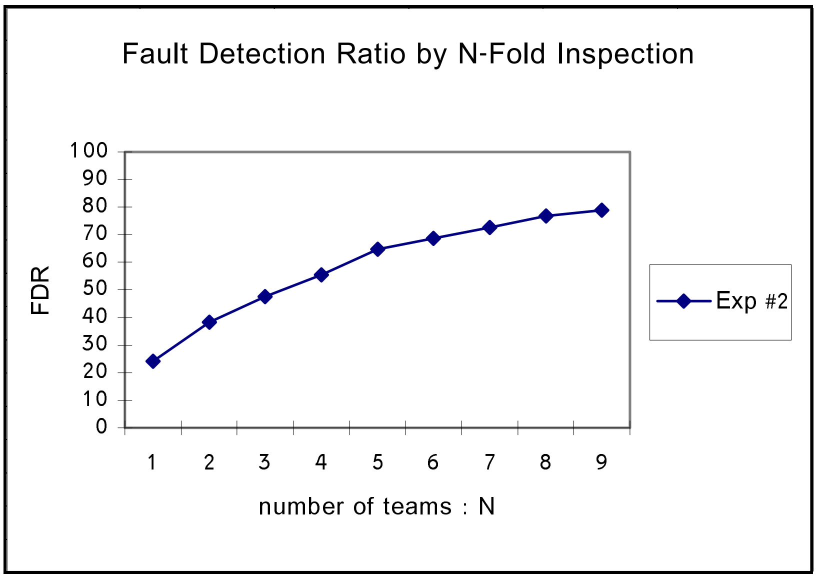


Figure 1 - Performance observed in Experiment 2.

A number of questions may be suggested by the performance curve shown in Figure 1.

1. Are the performance curves essentially identical? Are the differences between the two curves within the measurement error of these curves?

2. Is the shape of the performance curve dependent on the type of the application? Experiments 1 and 2 were done with a real-time railroad control system. Are the results different when we move to another application area, say an information system?

3. Experiments 1 and 2 were done with quite a large system. Are the results different when we move to a smaller or to a larger system? In other words, is there a scaling effect?

4. How does the level of expertise of the inspectors influence the shape of the performance curve?

In an attempt to answer these questions, a number of experiments were carried out.

THE EXPERIMENTS

The two subjects of our experiments were the URD of a small information system and the URD of a small real-time system. These systems differ from the system employed in Experiments 1 and 2 in size, and cover different application domains.

The information system URD was the subject of Experiments 3 through 5 and 8. The purpose of this system, whose URD is shown in Appendix 2, is to manage the information of a costume shop. It should control the inventory and customer databases. It should, furthermore, process orders and produce invoices. The two page long URD was written by a system analyst and an expert in costume marketing. Eighteen faults were thereafter entered into the URD.

The real-time system's URD was the subject of Experiments 6 through 7. The purpose of this system, whose URD is shown in Appendix 3, is to control an automatic missile launcher. The system responsibilities included the radar, the launcher, and the launching algorithms. This URD is three pages long. Seventeen faults were entered into this URD.

The errors entered into the two URDs were selected to meet two criteria:
1. They represent common URD faults.
2. They can produce serious malfunctions if not detected.

Faults of three different types were entered into the two URDs. These types are slightly simplified versions of those employed in [5]). This distribution of faults is derived from the analysis of [4] and is considered to be typical:

1. Missing or insufficient information, e.g. an object used in the system is not defined in the URD.

2. Missing vital functionality, e.g. an inventory control system without an inventory update operation.

3. Contradictions and ambiguous information. Different parts of the URD contain contradictory information and specifications that can be interpreted in more than one way, e.g. in one place of the document, the serial numbers of parts are specified to be 8 digits long, while an example employs the number 12345.

Figure 2 shows the distribution of the types of faults.

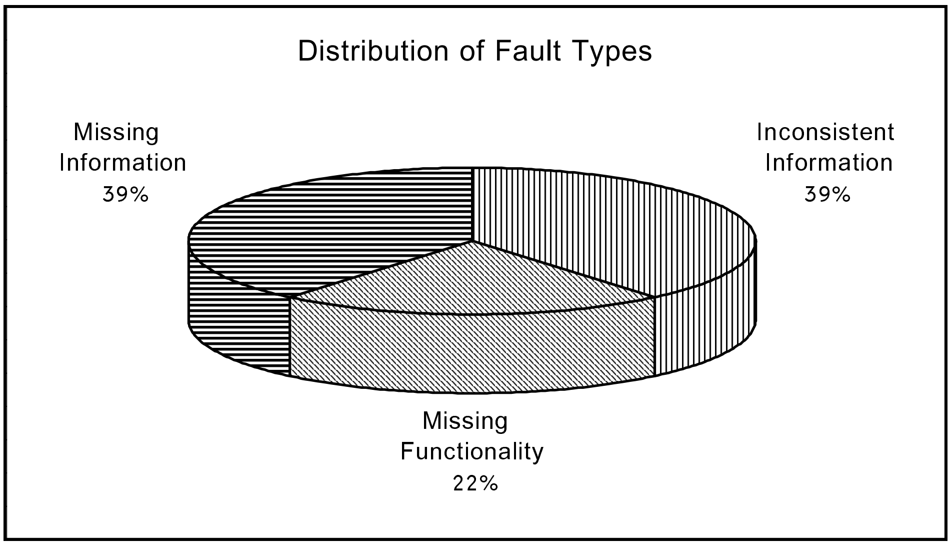


Figure 2 - Distribution of fault types in Experiments 3 - 5.

The participants of Experiments 3 through 6 were students of software engineering courses at the Technion, Israel Institute of Technology. In each experiment the participants were divided into three-participant teams. This division yielded from seven through nine teams in each experiment. In order to start at an even a starting point as possible, all the students received a lecture on URD inspection and on the types of faults that were inserted into our URD. The students were mostly in their fourth (senior) year of study. Each team received a copy of the URD and was given time to review it and to document the detected faults.

The moderator of the experiment entered the detected faults onto a spread sheet and processed them in the break between the lectures. In the lecture, following the break the results were presented graphically to the class and discussed. Only faults that belonged to the set of faults entered deliberately into the URD were analyzed.

Experiment Table (performed by senior students):

	Reference	System	Duration	Teams	Team Size	Participants
1	[4]	Railroad	6 hours	10	3	senior
2	[5]	Railroad	6 hours	9	3	senior
3		Costumes	20 minutes	7	3	senior
4		Costumes	40 minutes	7	3	senior
5		Costumes	30 minutes	9	3	senior
6		Missile launcher	30 minutes	8	3	senior
7		Missile launcher	35 minutes	7	3	freshmen
8		Costumes	40 minutes	8	3	freshmen

Table 1 : Experiment list

In the experiments we assumed that the inspectors had enough time to detect all the faults that they were able to detect. To check this, the students in Experiment 3 were given 20 minutes and in Experiment 4 40 minutes. The two experiments produced roughly the same *FDR* curve, i.e., it can be concluded that 20 minutes were sufficient and that if more time had been given, this would not have changed the results.

RESULTS

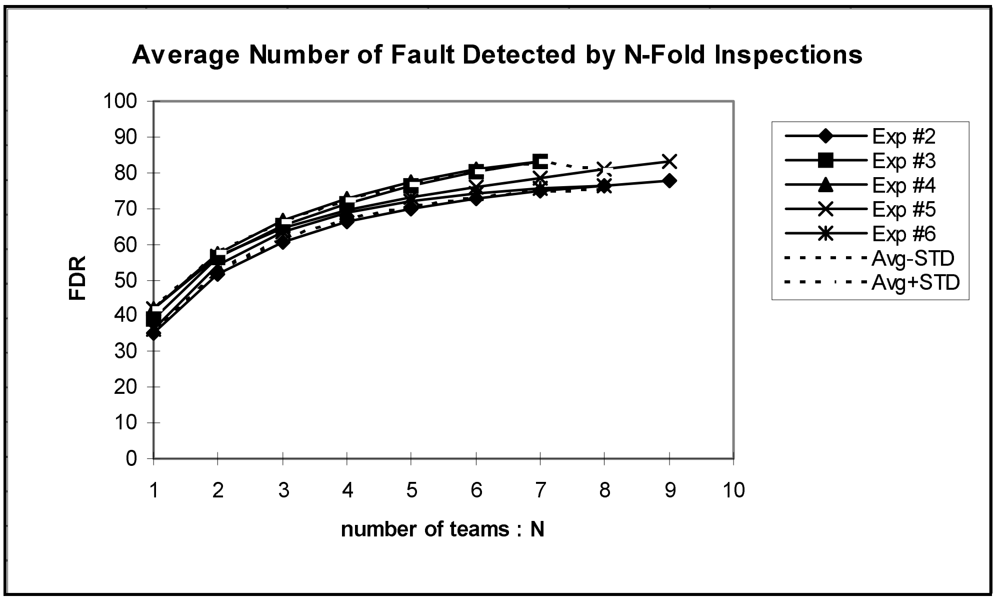


Figure 3 - The Fault Detection Ratio measured in Experiments 2 through 6.

Figure 3 shows the performance of the N-fold inspection method in Experiments 2 through 6. The graph doesn't include Experiment 1 because its data wasn't available in [4], but the results were reported to be identical to those of Experiment 2 [5]. The surprising result demonstrated by Figure 3 is that the curves are close to each other. For the railroad system of Experiment 2, the detection rate goes from 35.1% for *N*=1 to 77.8% for *N*=9. For the costume store system the FDR, the average of results of Experiments 3 through 5, goes from 38.9% for *N*=1 to 83.3% for *N*=7, and for the Missile launching system of Experiment 6, the detection rate goes from 36% for *N*=1 to 76.4% for *N*=8. These Figures support the claim of [5] that by increasing the number of inspection teams, a fault detection rate of 80% may be achieved. The basic assumption of the N-fold technique is that there is little overlap between the faults found by the different teams. Employing a larger number of the teams should therefore result in the detection of more errors. Figure 4 demonstrates that this assumption is met to a considerable extent.

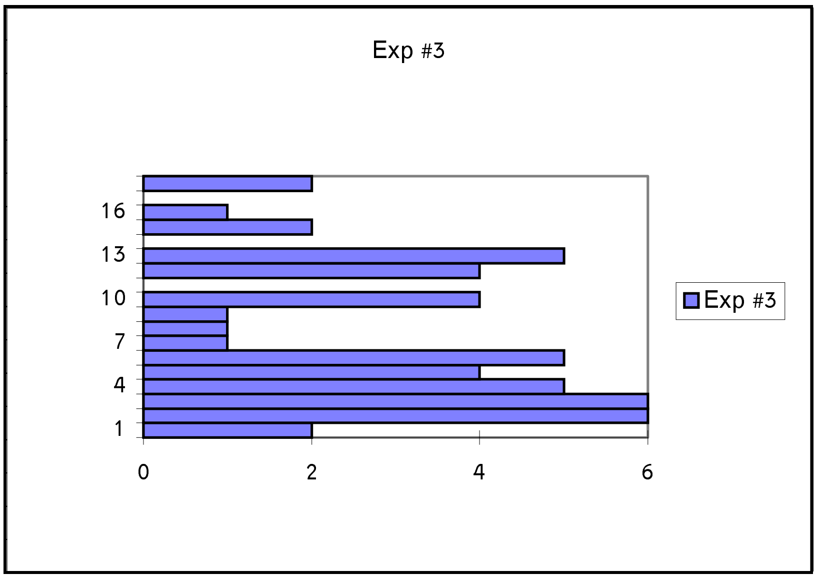


Figure 4 -
The number of teams that detected each of the 18 different faults in Experiment 3

It is observed that none of the faults was found by all the teams. Most of the faults were found by less than half of the teams.

In order to estimate the accuracy of the *FDR* values measured in the experiments, the corresponding standard deviations (*STD*) were computed. Let $FDR_{i,j}$ denote the average *FDR* value achieved with *i* teams in experement j, let $FDR_{i,avg}$ denote the average *FDR* value achieved with *i* teams in experament 2 to 6, and let *n* denote the total number of experiments, i.e. *5* (2 to 6).

$$STD = \sqrt{\frac{\sum_{j=1}^{n_i} (FDR_{i,avg} - FDR_{i,j})^2}{n-1}}$$

Figure 3 shows the two curves $FDR_{avg} - STD$ and $FDR_{avg} - STD$.

THE ROLE OF THE INSPECTOR'S LEVEL OF EXPERTISE

Experiments 1 to 6 produced nearly identical performance curves in spite of the fact that they involved systems of a different kind and size. Common to all these experiments was the fact that they were performed by computer science students in their last year of study. We decided, therefore, to conduct Experiment 7 and 8 with freshmen students. Those results will be compared to the results of Experiments 2 through 6 which were conducted with last-year students.

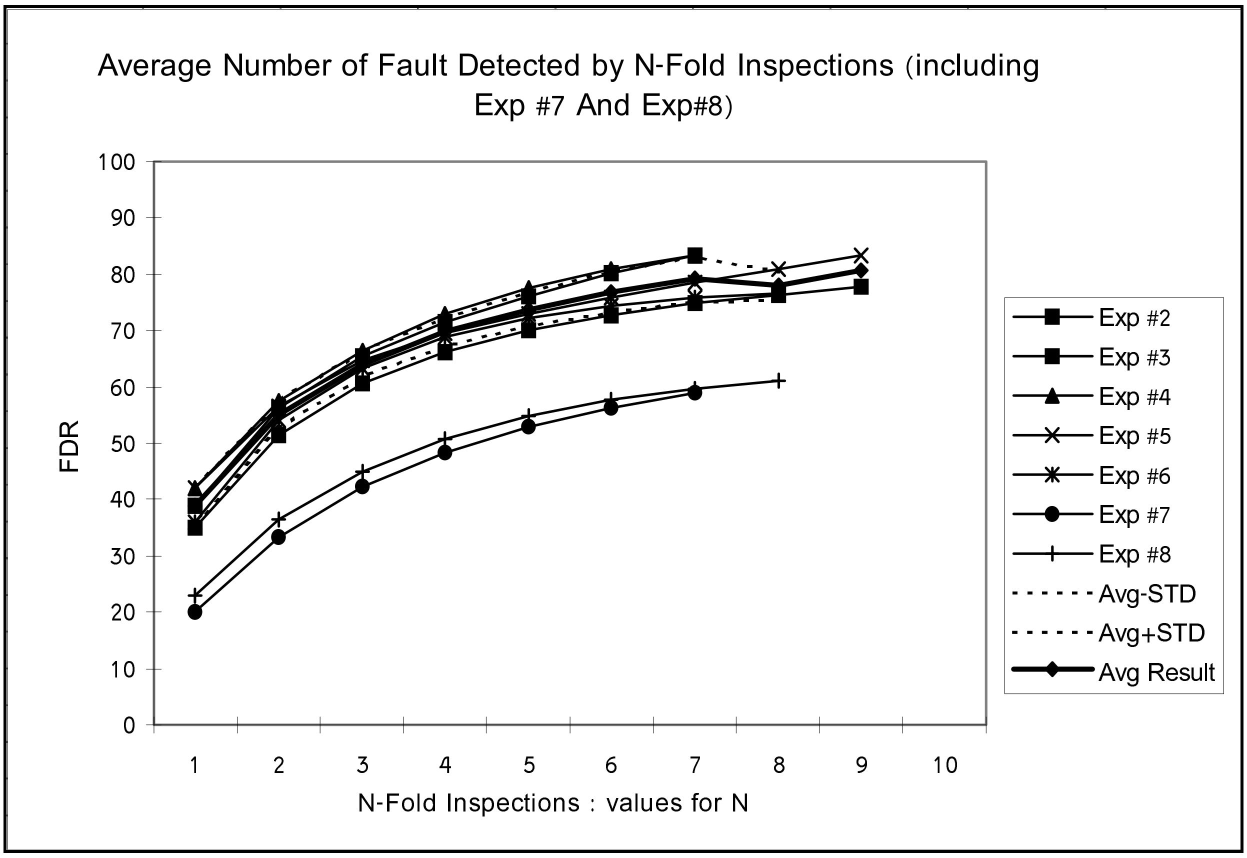


Figure 5 - FDR detected by students after only one year of study (Experiment 7 and 8) are considerably lower that those found by senior students (Experiments 2 through 6)

The students in Experiment 7 and 8 had only two basic courses in programming. The performance results of these students, as shown in Figure 5, was lower than those obtained in Experiments 2 through 6. The *FDR* in Experiment 7 goes from 20.1% for *N*=1 to 58.8% for *N*=7 and the *FDR* in Experiment 8 goes from 22.9% for *N*=1 to 62.1% for *N*=8. It is important to note that in spite of the differences in the performance, the shape of the curve is similar. The experiments suggest that the level of expertise of the inspectors is a dominant factor. In order to better understand this phenomenon, we analyzed the faults detected in the different experiments and constructed a probabilistic model.

THE PROBABILISTIC MODEL

For each one of the experiments we ordered the faults according to the number of teams that detected them. The resulting pattern was similar for all experiments. Figure 6 is an example for such a distribution based on Experiment 3. The data in this figure are the same as those shown in Figure 4.

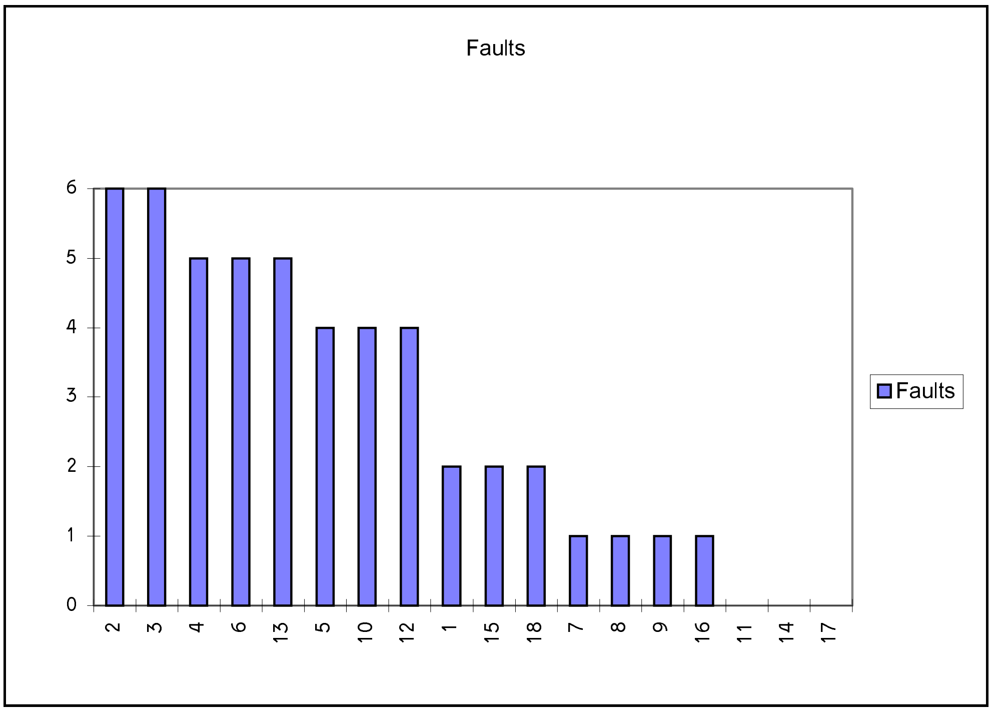


Figure 6 - Number of teams that found each fault during inspection ordered by number of teams.

Faults 2 and 3 were detected by 6 teams, Fault 4, 6 and 13 were detected by 5 teams, and so on. The three faults that were not detected by any team are to the right side (Faults 11, and 14, and 17 in Figure 6). To facilitate further analysis the numbering of the faults was changed. The left most fault in Figure 6 (fault 2) will be denoted as fault 0. The next fault, i.e. 3, is denoted 1 and so on. We can now, for each one of the faults, divide the number of teams that detected it by the total number of teams. This gives for each fault, a measure of the probability of detecting it. These probabilities may be approximated by the continuous probability distribution curve shown in Figure 7. This curve represents the probabilities of each kind of fault to be detected by a single inspection team. The curve of Figure 7 may be considered as an abstraction of the distribution shown in Figure 6. We did this analysis for all of our experiments. We found that the detection probability distribution could in all the cases be approximated by two straight lines as shown in Figure 7.

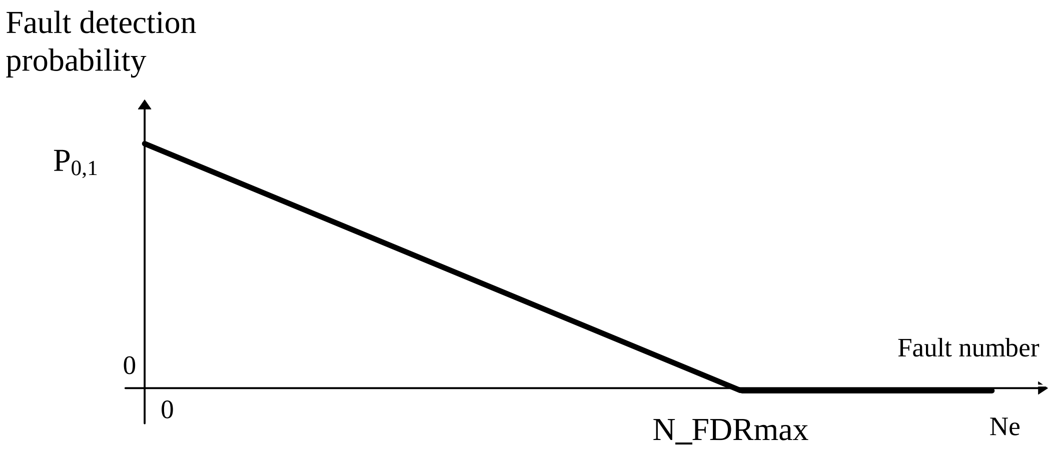


Figure 7 - Typical fault detection probability distribution

The straight line connects two points:

1. $(0,P_{0,1})$ which represents the probability that Fault 0 is detected by one team. Fault 0 is the fault that has the highest probability $P_{0,1}$ to be detected by a single team of inspectors.
2. $(N\_FDR_{max},0)$ which represents the fault numbered *N_FDRmax* that has the probability 0 of being detected. Faults *N_FDRmax* through *Ne* are the faults which were not detected at all. Due to our fault numbering system *N_FDRmax* is the maxim number of detected faults.

We assume that a distribution curve of the shape depicted in Figure 7 exists for any *N*-fold inspection. We employ this distribution to estimate the probability to find each fault by a single team. The following definition and formulas are employed:

Ne = total number of faults.

FDR_{max} = maximal fault detection rate, $FDR_{(\infty)}$. i.e. FDR_j for $j \to \infty$. Where j is the number of inspection team.

$N\_FDR_{max} = Ne * FDR_{max}$, the maximal number of faults that can be found.

$P_{i,j}$ = the probability for fault *i* to be detected by *j* teams.

$P_{0,1}$ = the probability for fault *0* to be detected by 1 team. Fault 0 is the fault with the highest detection probability.

$$(1)\qquad P_{i,1} = \begin{cases} \left(1-\left(\dfrac{i}{N\_FDR\max}\right)\right) * P_{0,1} & i < N\_FDR\max \\ 0 & otherwise \end{cases}$$

Now we can recursively calculate the probability of finding each fault by any number of teams using the recursion formula:

$$(2)\qquad P_{i,j} = P_{i,j-1} + \left(1 - P_{i,j-1}\right) * P_{i,1}$$

This formula describes the probability of finding the ith fault by *j* teams as the probability of finding it by *j-1* groups plus the probability of finding the fault by the additional group.

An equivalent explicit formula for computing $P_{i,j}$ is:

$$(3)\qquad P_{i,j} = \sum_{k=1}^{j} (-1)^{k+1} * \binom{j}{k} * P_{i,1}{}^{k}$$

From the last formula, the *FDR* can be directly computed.

FDR_j = The fault detection rate using *j* teams.

$$(4)\qquad FDR_j = \frac{\sum_{k=1}^{Ne} P_{k,j}}{Ne}$$

The FDR of one team (j=1) is thus:

.

$$(5) \qquad FDR_1 = FDR_{\max} \frac{P_{0,1}}{2}$$

A useful property of the probabilistic model is that its shape is determined by only two parameters, i.e., $P_{0,1}$ and FDR_{max}. If one can determine the values of these two parameters the FDR curve can be plotted.

As shown in Figure 8, the *FDR* computed by this formula fits well with the experimental results. The small differences between the probabilistic model and the experimental results are due to the differences between the actual fault detection probability distribution, as illustrated by Figure 6, and the idealized linear model (Figure 7 and equation (1)). A useful property of the probabilistic model is that its shape is determined by only two variables, i.e., $P_{0,1}$ and FDR_{max}. If one can determine the values of these two variables the *FDR* curve can be drawn.

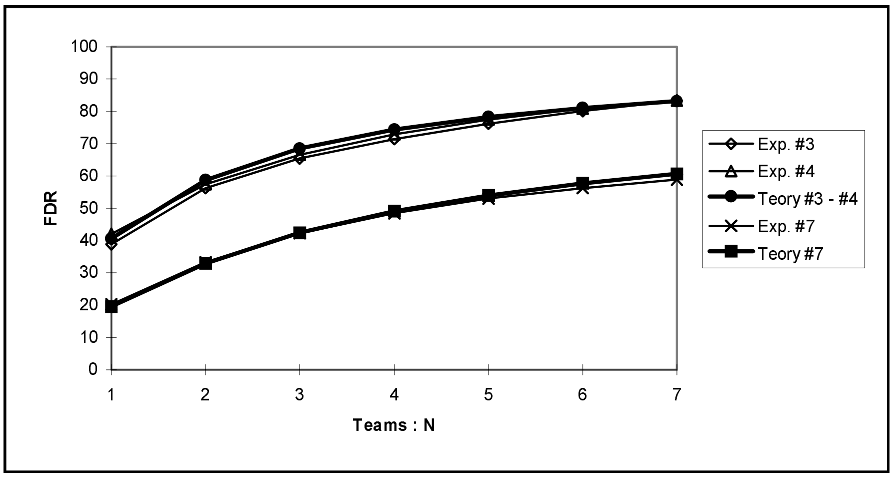


Figure 8 - Comparison of the probabilistic *FDR* model and the *FDR* measured in Experiments 3, 4, and 7.

The probabilistic model enables computing FDR_j, i.e. the *FDR* achieved with j teams, by using Equation 4. We employ $P_{0,1} = 0.9$, a value that was observed consistently in all of our experiments for both senior and sophomore students. *FDRmax* was in the case of senior students in Experiments 1 through 6, about 0.8, and for the sophomores in Experiments 7 and 8 it was about 0.6. This suggests that the shape of the *FDR* curve is essentially determined by *FDRmax*, which represents the level of expertise of the inspectors. In our probabilistic model, we therefore define:

> The *level of expertise* of the inspectors is defined by the *FDR* that may be achieved by an infinitely large number of teams, i.e. *FDRmax*.

Note that this definition is not dependent on the fault detection probability distribution, i.e. it is also valid when the distribution is different from that represented by equation (1). With this definition, the sophomore in our experiment had a level of expertise of 0.6 and the last year students had level 0.8.
Figure 9 shows the *FDR* curves derived from our probabilistic model for levels of expertise ranging from 0.60 to 1.00. The diagram assumes that equation (1) is satisfied and that $P_{0,1} = 0.9$.

HOW TO EMPLOY THE PROBABILISTIC MODEL

The diagram in Figure 9 may be readily employed when it is judged that the fault detection probability distribution is represented by equation (1), and $P_{0,1} = 0.9$. This was the case in all our experiments. The diagram may, in this case, be employed by a manager of a software project who wishes, for example, to estimate the number of inspection teams required for detecting 60% of all the faults (FDR=0.6). With inspectors having a level of expertise of 0.9 three teams are needed. The diagram shows that these teams will detect 65% of the faults. The manager may then compare the costs of these three inspection teams to the costs incurred if these 65% of the faults are not removed. The manager may repeat this comparison with different numbers of teams and determine the number of inspection teams that is most cost effective. It is difficult to accurately estimate the costs incurred by undetected URD faults. Therefore, it makes no sense in this case to work hard for a precise estimation of the FDR. The manager may therefore employ an easy imprecise method for estimating the level of expertise of his inspectors. The manager may employ our observations that computer science students, after one year of study, have a level of expertise of 0.6 and after 3 years a level of 0.8, while the highest possible level is 1.0. Thus for professionals, a possible guess may be 0.9. The advantage of this approach is that the manager can, in a few minutes, make a reasonable decision as to the number of inspection teams.

More precise predictions require that the level of expertise of the actual inspectors is known. This level may be measured by an N-fold experiment similar to those described in this paper. This involves entering representative faults in the actual URD. The measured FDR values may then be plotted in Figure 9 over the corresponding number of teams. If the shape of the plotted curve fits well with the shapes of the existing curves, one can readily estimate the level of expertise. For example, if the plotted curve is half way between the 0.8 and the 0.9 curves, the level of expertise is probably 0.85. It is, in this case, not necessary to do any further calculations.

If the shape of the plotted FDR curve differs from those of the curves of Figure 9 the explanation may be that $P_{0,1} \neq 0.9$. In this case the diagrams of APPENDIX 1 may be employed. They correspond to $P_{0,1} = 0.6, 0.7, 0.8, 0.9$ *and 1.0.* Another possible reason why the FDR curves may not fit is that equation (1) is not met, that is, the linear distribution shown in Figure 7 is a poor approximation of the actual fault detection probability distribution. It is required to check the actual distribution, the analysis described in the previous section must be made. This involves plotting a diagram similar to Figure 4 showing the number of teams that detected each one of the inserted faults. Then the fault detection probabilities are sorted into a diagram similar to Figure 6. From this diagram the actual values of $P_{0,1}$ and of N_FDR_{max} may be read, and. the level of expertise is computed as N_FDR_{max} / N_e. We can also check if this diagram has a shape similar to Figure 6 (equation (1)). Small deviations between the two shapes seem to have little impact on the results. The actual distribution shown in Figure 6 produced results that were quite close to the model (Figure 8). If the two shapes differs from each other considerably a new approximation may be developed. This new approximation will substitute equation (1), and a new probabilistic model is derived using equations (2) and then (4). These equations are valid for any formulation of equation (1).

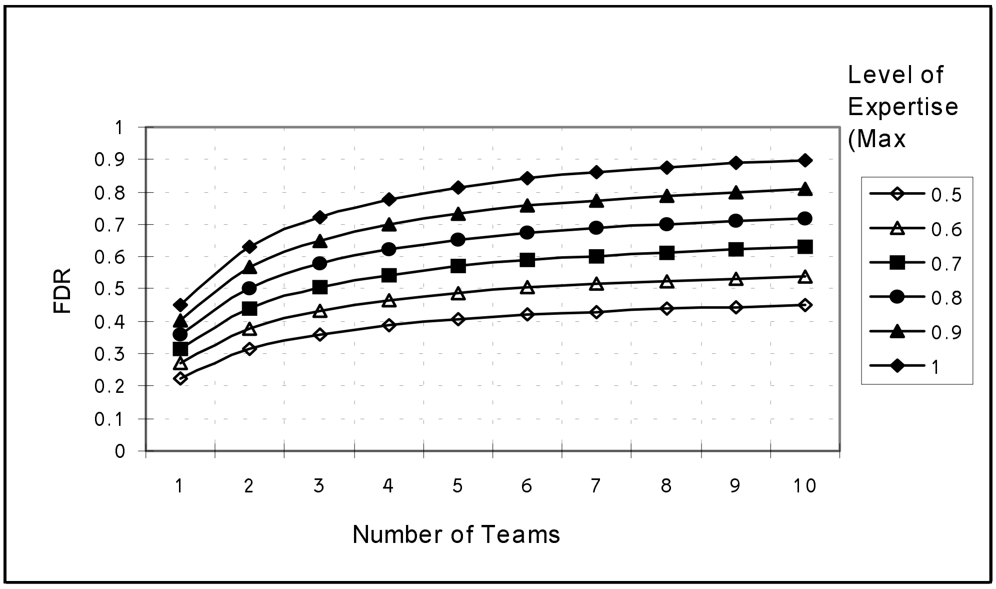


Figure 9 - Probabilistic model FDR as function of the number of inspection teams and of the level of expertise. $P_{0,1}=0.9$, which is the value observed in all the experiments reported in this paper.

CONCLUSIONS

Analysis of the experiments carried out by previous researchers and by the authors of the present study suggest that *FDR* may be estimated quite accurately with the N-fold inspection method. The standard deviation in the *FDR* measured in the experiments was around 0.03. The experiments also suggest that the *FDR* achieved by N-fold inspection seems not to be dependent of the type and size of the system. The dominating factor for the performance of the N-fold inspection method appears to be the level of expertise of the inspectors. This hypothesis is supported by seven experiments. These experiments involved a large number of students. Further support is due to the fact that experiments made by researchers in two different American universities and in one Israeli university.

Another observation is that it is relatively easy to make these kinds of experiments in a class with reasonably accurate results. A condition for succeeding in these experiments is that the students are well instructed before starting the experiments.
In one of the experiments, the instructor did not explain the need to find missing information and functionality. As a result the students found only a few faults of these kinds and the experiment was discarded. These experiments have proven useful in conveying the software engineering approach to students and may also produce new scientific insights.

There is experimental evidence that computer science graduate students provide an adequate model of the professional population [10]. In these experiments students and professionals from the industry were asked to detect the faults in the same set of requirements using a number of different methods. While the professionals detected a larger number of the faults, the outcome of almost all statistical tests was identical. This is similar to our observation that the same kind of probabilistic model can describe the performance of junior and senior students, and the difference is that the performance curve of the senior students is higher than that of the junior ones (Fig 8). The probabilistic model is therefore expected to apply also for professionals, i.e. the difference between junior students, senior students, and professionals in our model is in their different levels of expertise.

The level of expertise is a representation of the number of detected faults. Inspectors employing an efficient inspection method will therefore have a higher level of expertise than equally-able inspectors who employ an inferior inspection method that detects less faults. Differences in the efficiency of different fault inspection methods are reported in [10], where three different fault inspection methods were compared experimentally. The inspectors who employed a method called *scenario* detected a significantly higher number of faults in comparison with those detected by employing the *ad-hoc* and *checkli*st methods. Inspectors using the scenario method have therefore in our model a higher level of expertise than those using one of the two inferior methods. Using efficient methods is one of the characteristics of experts. The efficiency of two different inspection methods may therefore be compared by means of the levels of expertise achieved by employing them. An enterprise that wishes to monitor possible improvements of their inspection method may do that by measuring the level of expertise achieved in the inspections. Measurement of the level of expertise of the inspections could be a standard part of the software process of the enterprise.

One of the practical results of this work is the probabilistic model developed for the N-fold inspection method. The model seems to capture the performance of the N-fold requirements method using only two parameters, i.e. the level of expertise and $P_{0,1}$. This model is represented in the diagram in Figure 9, which may be employed for a fast estimation of *FDR* for determining the optimal number of inspection teams and for solving similar problems. Diagrams that enable such fast decision making are, by the way, common in handbooks of established engineering disciplines. The model seems also applicable to monitoring the efficiency of the inspection process and for more accurate predictions. This involves, however, that the actual levels of expertise be measured by N-fold experiments.

REFERENCES

1. Fairley, R., *Software Engineering Concepts*, McGraw-Hill, New York, 1985.

2. Porter A., Siy H. and Votta L., “A Review of Software Inspections”, in *Advances in Computers,* Vol. 42, Academic Press, NY, 1996, pages 39 - 76.

3. Boem, B., “Industrial Software Metrics Top-10 List”, *IEEE Software,* Vol. 3, No. 9, September, 1987, pages 84-85.

4. Martin, J. and Tsai, W.T. “N-Fold Inspection: A Requirements Analysis Technique”, *Communications of the ACM*, Vol. 33, No. 2, February, 1990, pages 225 - 232.

5. Schneider, G. M. Martin, J. and Tsai, W.T., “An Experimental Study of Fault Detection In User Requirements Documents”, *ACM Transactions on Software Engineering and Methodology*, Vol. 1 No. 2, April, 1992, pages 188-204.

6. Porter, A.A. Votta, L.G. Jr. and Basili, V.R. “Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment”, *IEEE Transactions on Software Engineering*, Vol. 21, No. 6, June, 1995, pages 563-575.

7. Boem, B., “Verifying and validating software requirements and design specifications”, *IEEE Software*, Vol. 1, No. 1, January, 1984, pages 75-88

8. Hare, A. P., “Group Size”, American Behavioral Scientist, Vol. 24, No. 5, May/June 1981, pages 695-708

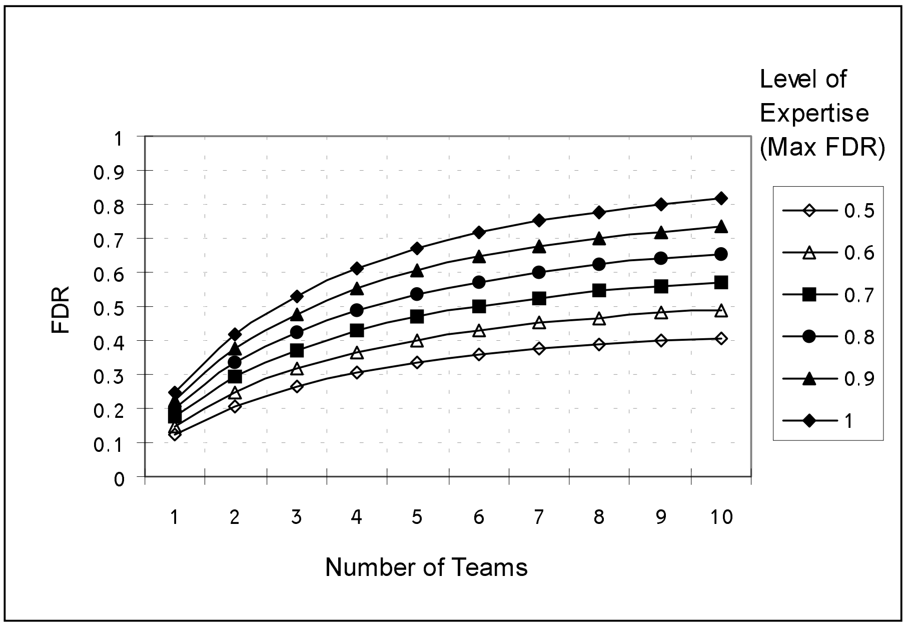
9. Hare, A. P. , Small Group Research: A Handbook, Ablex 1994

10. Porter, A., Votta, L., Comparing Detection Methods for Software Requirements Inspections: A Replication using Professional Subjects, Paper No. 2 in http://www.cs.umd.edu/~aporter/html

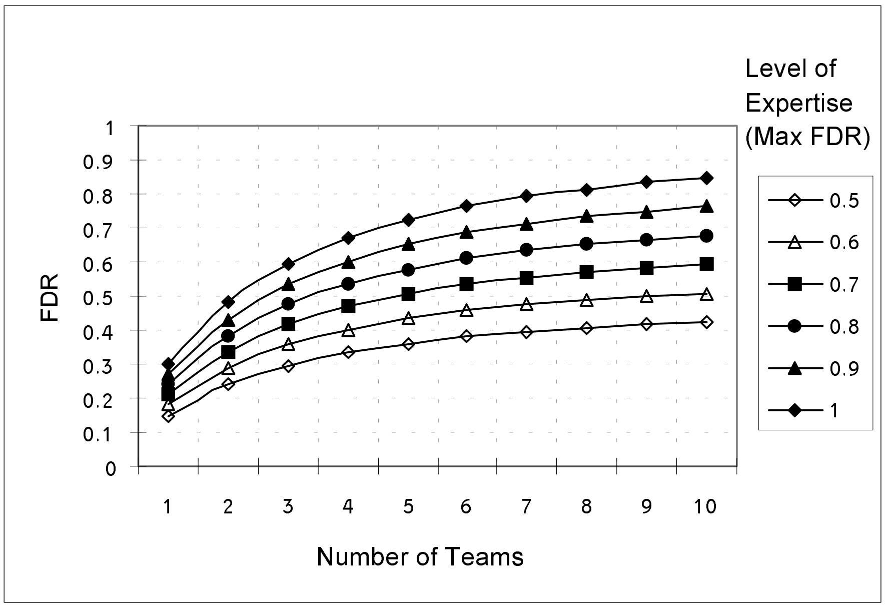
APPENDIX 1

PROBABILISTIC MODEL FOR DIFFERENT $P_{0,1}$ VALUES

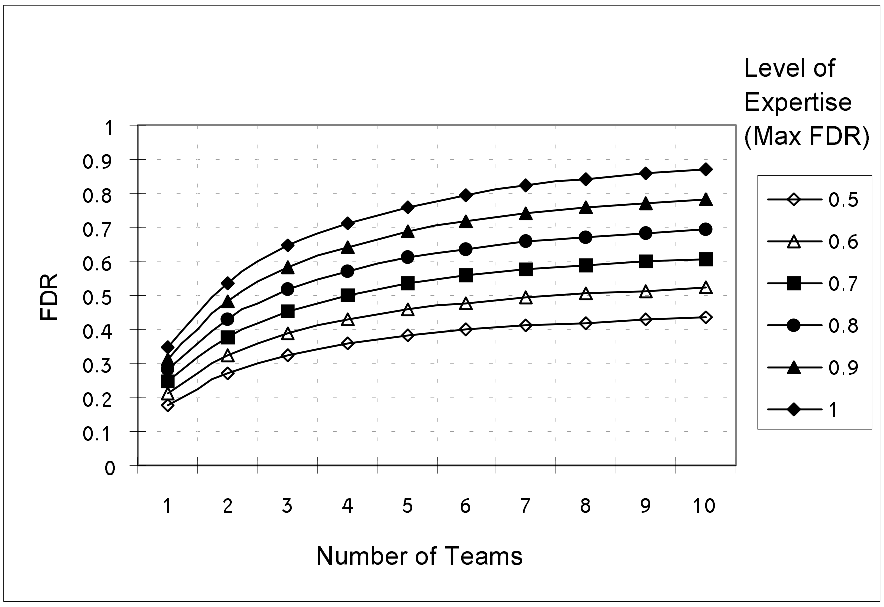
The model is defined by equations (1), (2) and (4).



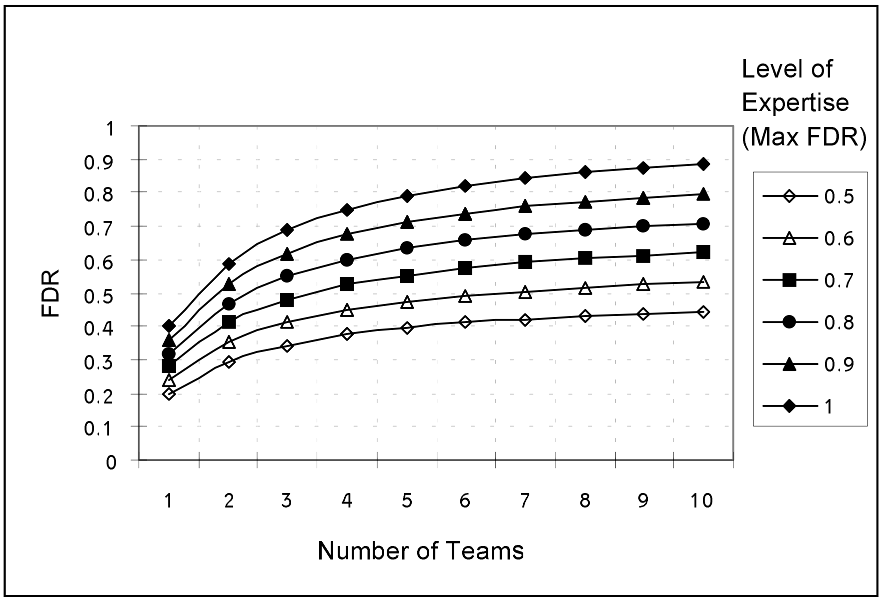
$P_{0,1}$=0.5



$P_{0,1}$=0.6

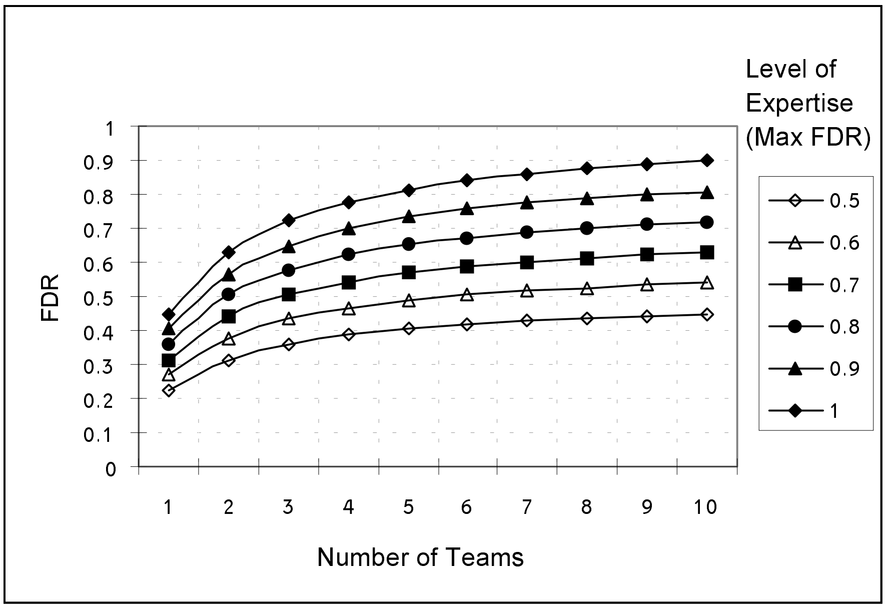

Level of Expertise (Max FDR)
FDR
Number of Teams
0.5
0.6
0.7
0.8
0.9
1

$P_{0,1}$=0.7

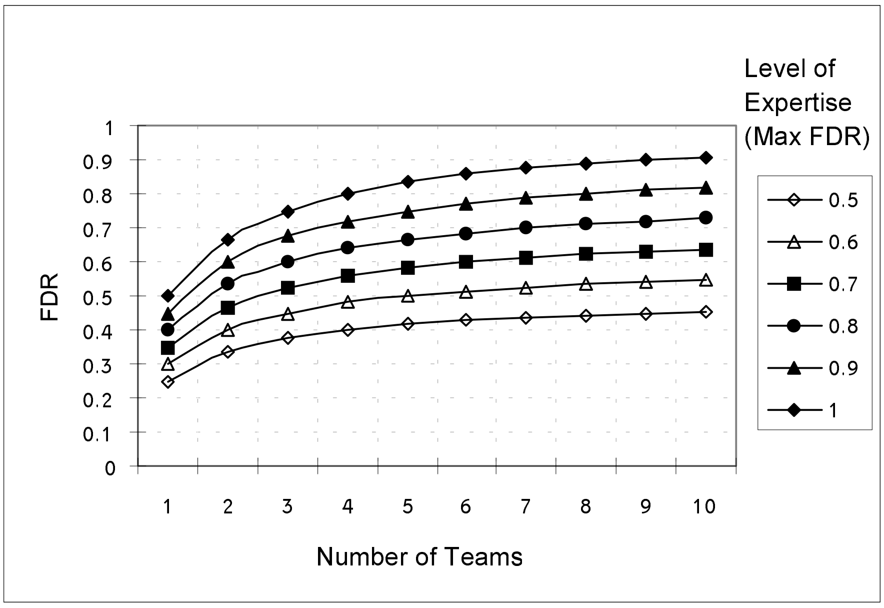

Level of Expertise (Max FDR)
FDR
Number of Teams
0.5
0.6
0.7
0.8
0.9
1

$P_{0,1}$=0.8

\


Level of Expertise (Max FDR)
0.5
0.6
0.7
0.8
0.9
1
FDR
Number of Teams

$P_{0,1}$=0.9


Level of Expertise (Max FDR)
0.5
0.6
0.7
0.8
0.9
1
FDR
Number of Teams

$P_{0,1}$=1.0

APPENDIX 2

The URD employed in Experiments 3 through 5:

User Requirements Document

PURPOSE

1. Manage a costume shop, which rents and sells costumes.
2. Control the inventory and customer databases. Manage orders and invoices.

   CUSTOMER DATABASE - SYSTEM ACTIVITIES

3. Enter new customers.
4. Automatic updates of the customer's database.
5. List of customers active over the last three years.
6. List of customers ordered by the age of the children.
7. List of customers ordered by their purchase and rental transactions.
8. Report of customers ordered by the number of costumes sold and by their total price paid.
9. List of troublemaker customers.
   Remark: The system will be able to send mail to the customers listed in any one of the previous categories. Includes the printing of address labels.
10. The system will keep in the database only the name and address of the customer. This information will be automatically deleted after two years.

    INVENTORY DATABASE - SYSTEM ACTIVITIES:

11. List inventory of costumes at start of season.
12. List inventory of costumes at end of season.
13. List current inventory of costumes.
14. List inventory at a specified date based on sales prediction.
15. List inventory of costumes at a specified past date.
16. List inventory of costumes ordered by their year of manufacture.
17. List inventory of costumes ordered by size and sex.
18. Alarm when inventory of certain items reaches a specified minimum level.
19. List inventory of costumes ordered by the season to which they are suited.
20. Price list of all the costumes, including price lists of past years.
21. Support of costumes active and non active (non active are those items that are not manufactured any more).
22. Report of the costumes that were missing last year (costumes that were missing are costumes that were out of stock in the last week and a half before the Purim festival).
23. Unisex costumes (suitable for both sexes).
24. Possibility of adding items to the inventory.
25. Report of the value of the inventory.

    Orders and invoices:

26. Entries and updates of the customer database may only take place at the time of invoice production.

27. Support of both costume orders and costume reservations.
28. Support for order and invoice cancellation.
29. Support for order updates.
30. Sales report for any specified (past and current) date.
31. List open orders.
32. Not possible to cancel guarantied orders.
33. Each costume in stock is for sale.

List of Errors inserted in the URD

Inconsistent Information:

1. Lists of customers entered by different techniques that contradict each other (lines 3 and 26).
2. Cancellation of an order that was reserved is illegal (lines 28 and 32).
3. The systems do not keep customer data for more than three years (lines 5 and 10).
4. There is not enough information about the customers in the system (lines 6 and 10).
5. An article that was reserved cannot be sold. (lines 27 and 33).
6. It is impossible to order the costumes by sex (line 17 and 23).
7. Not enough information about clients is kept (lines 10 and 9).

Missing functionality:

8. There is no automatic update of the stock when an invoice is issued.
9. No program was defined to register the initial stock.
10. No stock update was defined.
11. No online software assistance was defined.

Missing information:

12. “Costume” was not defined.
13. Order properties were not defined.
14. Invoice properties were not defined.
15. No service programs were defined, type of backup and files rescue, and the quantity or frequency needed.
16. No hardware and users (single / multi-user) were defined.
17. The stock’s database does not distinguish between items that are related to a certain season and those that are not season related.
18. No details of critical dates were mentioned.

APPENDIX 3

AERMS - Automatic Enemy Ranging Missile System

1. Introduction

1.1. Document purpose:
The purpose of this document is to give a preliminary overview of the fire control software of the AERMS system, as was discussed in the meeting between the customer (Colombian Mafia Ltd.) and the development organization (Arrow Edge Ltd.). This document will be used as a Software Requirement Specification and should be given to the software division for construction.

1.2 Product range:
The software defined in this document includes only the radar information processing and the fire control. All other software components in the AERMS system are defined in separate documents.

1.3 Special definitions and abbreviations: None.

1.4 Resources: Our imagination.

1.5 Document overview:
The document contains the following sections:
Chapter 2 - general overview of the software and its requirements.
Chapter 3 - detailed description of:
Software interface to other system components, the main functions and the database.
No appendices included.

2. General overview

2.1. General overview of the product:

2.1.1 AERMS is an automatic missile launcher installed on vehicles moving in hostile territory. The system should defend the vehicle from enemy aircraft without any human interference.

2.1.2. AERMS interfacing the radar and the launching components.

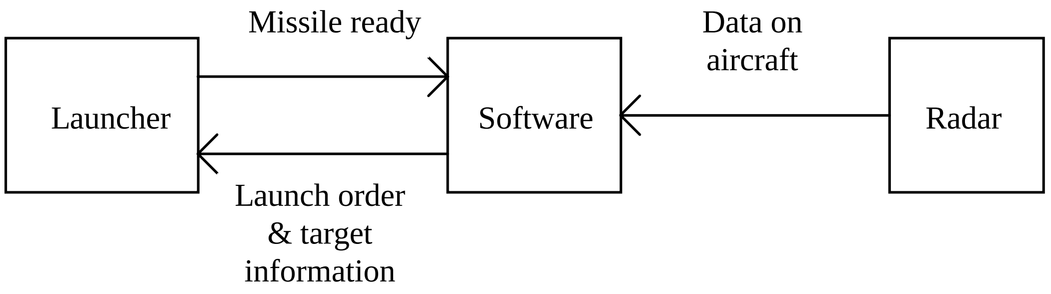


Figure 1: Software relationship with other components.

2.2. System functionality:

2.2.1. The system will conduct a database to monitor the aircraft spotted on the radar. The database will be updated constantly with the data generated from the radar. It will monitor the existing aircraft, delete those which exit the range, and insert those entering it.

2.2.2. The system will evaluate the amount of danger facing each aircraft in the database according to roles supplied by the customer.

2.2.3. The system will determine independently when and on which target to open fire, according to algorithms provided by the customer.

2.3. System user specification:
No user.

2.4. General requirements:

2.4.1. In no case will the system issue a missile launch order against a friendly aircraft.

2.4.2. The system will fire sparingly , in order to preserve its missile stock.

2.4.3. The system operation will be strictly automatic. Therefore, error handling must be done by software (exit with error message or return control to the user are not allowed).

<u>3. Specific requirements</u>

3.1. Software interface:

3.3.1. Radar interface:

The radar unit rotates continuously at 360 degrees and sends the following data for each rotation, on each identified aircraft :
1. Azimuth of the aircraft (orientation from the radar) [degree].
2. Range [kilometer].
3. Height [low, medium, high].
4. Velocity [slow, medium, fast].
5. Direction [closing in, moving away].
6. If the radar receives a special signal from the aircraft the parameter ,“friendly aircraft”, will be sent.
   Otherwise the aircraft will be defined as an enemy aircraft.

On any reception of data from the radar, the system will automatically update its database.

3.1.2. Launcher interface:

The launcher will inform the system when a rocket is ready to be launched. The system checks the amount of risk coming from each aircraft while a missile is ready in the launcher. According to this information the system will determine if a missile should be launched and against which target. At launch time, the system will transfer the target information to the launcher (1-5 in section 3.1.1.).

3.2. Tracking activities:

3.2.1. Updating existing aircraft position:

If the information coming from the radar matches the information of an aircraft which already exists in the database according to the following criteria:
1) new azimuth = old azimuth ± 5 degrees.
2) new range = old range ± 2 degrees.
3) if there is more than one aircraft that matches criteria: 1, 2, the closest one according to the other data should be chosen (i.e., the one with the most suitable information about altitude, velocity and direction).

Then:

4) The following data of the aircraft will be updated:
Azimuth, range, altitude, direction, speed and latest update time (= actual time).
5) The hazard level of the aircraft will be updated (the means of updating will be detailed later on).

3.2.2 Insert a new aircraft
If the data received from the radar doesn't match any DB aircraft according to the criteria defined in 3.2.1, a new record will be inserted with the radar data (described in 3.1.1 1-6) and with a random serial number.

3.2.3 Delete an aircraft from the DB
To prevent a downed aircraft or an aircraft that is out of radar range from remaining in the system, the system will search the whole DB each time that an update or insert operation is engaged. This search will dump any aircraft from the system whose latest update occurred more than two minutes ago.

3.3 Hazard level classification
The hazard level is a number in the range 0 to 9. The number 0 means not dangerous and 9 means most dangerous. The hazard level will be updated each time the current aircraft is updated according to the following criteria:
3.3.1 The hazard level for a friendly aircraft will always be zero.
3.3.2 The hazard level for an unfriendly aircraft in a range closer than 15km, will be increased by one.
3.3.3 The hazard level of an aircraft that flies away will be decreased by one.
3.3.4 The hazard level of an aircraft that flies at high speed and approaches or that is in a range closer than 5km, will be increased by two.

3.3.5 The hazard level of an aircraft that approaches or that is in a range closer than 10km and with an altitude lower that 2km, will be updated to nine.

3.4 Launch decision

3.4.1 First of all, the DB will be searched and will make a list of all aircraft with a hazard level higher than 7, and without a launched missile (= estimated hit time has already passed).
3.4.2 If there is an empty list, no missile will be launched. Otherwise, a launch missile command will be executed (with data as described on 3.1.2) against the aircraft with the highest hazard level. If there is more than one candidate, one of them will be chosen randomly.
3.4.3 If a launch missile command was executed, the target aircraft data will be updated with the estimated hit time (according to a computation that will be available later on from the launch system).

3.5 Database (DB)

3.5.1 The DB will be able to store at each moment all the aircraft that the radar can discover on one scan (rotation).
3.5.2 Aircraft record description:

1) System number (a two digits number).
2) Azimuth .
3) Range.
4) Altitude.
5) Speed.
6) Direction.
7) Predicted hit time, (recorded as the internal system clock format).
8) Friendly aircraft (Boolean).
9) Hazard level (0..9).
10) Last update time (same format like (7)).

3.6 Others

For every additional day after the deadline, one programmer will be decapitated.

URD Errors table

Index Number	Sub-index (if any)	Error description
3.1.1	6	It is not clear enough whether in the case of an enemy aircraft the radar will supply this information or will not supply information about friendliness at all.
3.1.2		There is no way of knowing that the missile launcher isn't ready for launch (Does a cancel message exist ?).
3.1.2		What will be the system review frequency for launching.
3.2.1	2	Km instead of degrees.
3.2.1	3	What happens if there is more than one match.
3.2.1	3	There is also a need for a match "friendly" field.
3.2.1	4	The "friendly" field isn't updated.
3.2.2		There are no startup (initial) values for the hazard level and the last update time fields.
3.2.2		There is no defined value for the "last launch time"
3.2.2		Random is not necessary unique in contradiction with 3.5.2
3.2.3		Should cancel be carried out before or after an update/insert?
3.3		There could be underflows under 0 or overflows over 9.
3.3.3-5		Does it mean a friendly aircraft or any aircraft (in which case it is in contradiction with 3.3.1).
3.3.4		And/or conditions are not clear.
3.3.5		There is no way of knowing if the altitude is below 2 Km (only lower/intermediate/higher).
3.4.1		Including or not including 7.
3.4.2		"The aircraft with the highest hazard level" on the list or anywhere.
3.4.3		TBD - Must define execution time.
3.5.1		"The number of aircraft that the radar can discover on one (turn) scan" must be defined.
3.5.2	1	Two digits = Not more than 100 aircraft. In contradiction to 3.5.1 .
3.5.2	7	Internal system clock format not defined.