Establishing the Relevancy of the Bookkeeping Libraries to the Functional Testing of Computer Implementations

Stamatis Vassiliadis, *Fellow, IEEE*,
George Triantafyllos, and Walid Kobrosly, *Member, IEEE*

Abstract—In this paper, we address issues related to the definition of "faults," "errors," and "failures" and their separability, and attribution to the different development processes of computing systems. In particular, we deal with historical databases, which presumably contain certain data (i.e., test failure data) and describe the methodology that can be used to analyze the database and obtain the pertinent information. The validation method may be of particular importance, especially when information from the database needs to be extrapolated for a purpose other than the one for which the database was developed. Our methodology was used to evaluate the historical data collected during the development of the IBM 4381 and 9370 family of computers, and to extrapolate the faults found during the function testing.

Index Terms—Functional testing, errors, bugs, faults, software reliability, error data, data accuracy, error prediction models.

1 Introduction

FUNCTIONAL testing, defined here as the process of evaluating the functions of computer systems and software products to assure that they meet prespecified requirements, constitutes an integral and important process in the development of computer systems and software products. Functional testing, as most of the processes involved with the computing systems, has a number of diverse aspects. The major contribution to the diverse aspects of functional testing is what is commonly referred to as a "bug," which may be discovered, corrected, and attributed to different processes of the development of a system. For example, a "bug" observed during the execution of a program can be attributed to software, hardware, logic design, technology, manufacturing, etc., and depending on the process involved different aspects of the testing may be uncovered. Due to the plurality of the processes involved with functional testing, in most cases, researchers consider issues related to some processes and exclude others. For instance, a researcher may consider issues concerning the functional testing of the processes involving logic design and microcode development, and exclude other processes involving technology, power supply, packaging, cooling, software, manufacturing, etc. At no exception, our studies in the past five years have been concentrated in the general topic of "error" prediction models, see for example [1], and their relation to the "bugs" (also referred to commonly as "errors," "defects," "faults," etc.) to be experienced by a development team during the design and implementation of a computer system. In particular, we were concerned with two development teams, namely, the logic design and microcode development teams.

One important issue, often neglected and not discussed in the literature, is the choice and validation of the data used in the research of functional testing. In this paper, we discuss the procedure we used to choose and validate the "error" data in our studies. As it will become obvious from the presentation to follow, the choice, extrapolation, and validation of data (even though necessary) was more involved than originally anticipated providing a partial justification why most researchers accept the data "as is." In any case, we hope that our experience provides some general guidelines for procedures in the maintaining and using the "error" data and precise definitions that allow at least a uniform treatment of "error" data in the future.

Before we proceeded in the investigation of the different aspects of functional testing in the development of computing systems, we considered of extreme importance to provide the answer to the following questions:

- What constitutes an "error"?
- Can an "error" be attributed to the different development processes?
- Can the "errors" be separated to the different development processes?
- Which historical data sources contained information pertinent to the aspects of interest regarding functional testing?
- How can the "errors" be attributed to the functional testing, after the source of historical data is established?
- Is the "error" data extrapolated from a historical source accurate and representative of the aspects of the process under consideration?

The answers to the previous questions for our application will be found in the sections to follow. The organization of the discussion is as follows: First, we describe what we considered to be an error, fault and failure, and discuss the separability of errors. Consequently, we discuss issues regarding the historical data sources and the data accuracy. Finally, we describe a database, denoted as the library subsystem, that we considered as the most appropriate for our research, the extraction of the "error" data using a fuzzy logic question answering system [2], and finally we establish a confidence level for the accuracy of the error data.

2 Errors and Separability of Errors

In this paper, we adopt the predominantly accepted definition for failure, fault, and error proposed by Avizienis and Laprie [3], which indicates that:

> "A system failure occurs when the delivered service deviates from the specified service, where the service specified is an agreed description of the expected service. The failure occurred because the system was erroneous: An error is that part of the system state which is liable to lead to failure. The cause-in its phenomenological sense-of an error is a fault. An error is thus the manifestation of a fault in the system, and a failure is the effect of an error on the service."

During the development of a computer system, when a deviation from the expected service (a failure) is detected, the state of the system (the error) causing the failure is determined and an attempt is made to correct the cause/causes (faults) leading to the unwanted system state. Given that there are a number of processes associated with the development of a computer system, determining which process is responsible for a fault constitutes a difficulty that needs to be overcome in order to resolve the failure (e.g., an architectural deviation in a computer system implementation may be the result of a fault introduced in manufacturing, logic design, circuit design, etc., and the resolution of failure requires establishing the "responsible" development process). To be able to attribute faults to the different development processes, it is required to be able to categorize the different kinds of faults, and to separate and attribute the faults to the different processes. Avi-

- *S. Vassiliadis is with the Department of Electrical Engineering, Delft University of Technology (T.U. Delft), 2600 EA Delft, The Netherlands. E-mail: stamatis@duteca.et.tudelft.nl.*
- *G. Triantafyllos is with Technology Integration, Products Development Division, Intrasoft S.A., Athens, Greece.*
- *W. Kobrosly is with the AWS Division, IBM Corporation, Austin, TX 78758.*

Manuscript received 7 Dec. 1994.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 104396.

zienis and Laprie [3] categorized the logic circuitry faults, the subject of our general interest, into:

1) Physical faults due to external influences (e.g., power supply fluctuations, electromagnetic interference, radiation) and common weaknesses in the manufacturing process.
2) Design faults introduced by human mistakes and faulty design tools, as well as by ambiguities and errors in the initial specifications.

Clearly, the categorization of faults alone is not enough to serve the purpose of collecting data for the development of a model. Separation and attribution of faults is important as clarified by the following scenario: Assume that we are interested in the logic design process, thus we should be able to separate which faults are associated with physical faults as they are not part of the design faults. Furthermore, we should be able to distinguish logic design faults from faults associated with the tools for the same reasons. In other words, we should be able to distinguish which faults have been introduced by the logic design team, and which were not.

The separation of faults can be achieved by considering the following:

- The group of people participating in a development effort of a system is subdivided into teams.
- Each team is responsible for a particular process.
- Teams communicate with predefined specifications.
- The underlined assumption among teams is that predefined specifications will be met.
- When a failure (resulting in the detection of an erroneous state, i.e., an error) has been discovered, the team responsible for the process that presented the failure will correct the fault/faults.

The essence of the previous statements is that an error/fault can be associated with a particular team as it requires the intervention of the team for its resolution. The suggestion here is that the responsibilities of a team is a key contributor in separating and attributing errors/faults, and that the intervention of a particular team to resolve an error/fault can be used to separate and count the errors/faults. Clearly, errors that require the intervention of multiple teams could (and in our opinion should) be considered the manifestation of multiple faults.

2.1 The Dilemma of the Error Data

In order to develop an "error" prediction model for computer implementations, it is often required to conduct research based on historical data (needed to derive and validate the model). Two sources of databases containing historical data, denoted here as "error tracking" libraries and "bookkeeping" libraries, have been used in the past by researchers for the development of "error" prediction models in software and microcode processes, see for example [4], [5], [6]. The first source, denoted here as "error tracking" libraries, are usually established during the integration of a system to report defects and track their resolution. The second source, which we will refer to as the "bookkeeping" libraries are usually created in the beginning of the development cycle to manage the revision of the code and to maintain information regarding the development processes. The dilemma regarding which library to use is not so much which library is the most representative of the development process, both are considered representative by a number of researchers, but rather which of the two libraries contains the most accurate data. Such a dilemma is seldom discussed in the literature as either researchers have available just one of the libraries, or it is assumed that there are no accuracy problems.

The "error" tracking libraries are usually established during the integration of a system to report defects and track their resolution, and library entries are established as follows: When a deviation of a prespecified behavior has been established, an entry is created. Such an entry usually corresponds to the report of an observed deviation (i.e., the entry represents the description of a failure), or the description (in a number of cases the partial description) of the machine state that leads to a failure (i.e., the description of an error). The bookkeeping libraries are usually created to maintain information regarding the development processes and they are initiated in the beginning of the development cycle. Library entries are established, in addition to reasons unrelated to functional testing, for the correction of faults in the form of changes (e.g., adding newly developed code, updating comments after code is developed, updating code containing the code itself, etc.). Consequently, faults can be accounted by examining the number of changes.

Clearly, the two databases are different and they may face different types of accuracy and representation problems. In discussing the accuracy of the data, we begin by considering a common accuracy problem: An error is the part of the system state which is liable to lead to a failure, suggesting that experiencing no failures does not imply the absence of errors, and thus the absence of faults. This introduces the first inaccuracy in developing models based on historical data, as those data reflect only the discovered errors, an approximation of the total number of errors and faults existing in a product. Second, the libraries available today may not take into account the severity of the errors/faults, an important parameter in scheduling the development, because they attribute the same weight to all library entries.

Regarding the "error tracking" libraries, multiple failures may correspond to a single fault but logged multiple times in the libraries (the opposite also holds true). Also, the error keeping begins usually long after the initial design and entry of components, implying inapplicability to the entire development process. Additionally, in instances it becomes a means for communication among groups rather than a means of future studies to understand the development process. As a formal process, there may be a reluctance of different groups to report errors and rather rely on private communications. The implications here is that the libraries partially reflect the development of a system and in instances they may contain misleading data. Furthermore, on occasion, some individuals, primarily due to misunderstanding, may report non-existing errors. An example of this is the case in which individuals misinterpret the output of test cases and confuse tool errors for design errors. Additionally, reporting an error in a unit that appears to belong to the responsibilities of a certain group may not be true and thus a concatenation of error entries may occur until the faulty unit is established resulting in multiple entries of an error. This may not always be trackable, depending on the set up of the library, resulting in multiple counting of the same error.

The previously discussed problems are not encountered in the bookkeeping libraries. The advantages and problems with this type of libraries rely on the following:

- Bookkeeping libraries are developed and contain information that is necessary for the development (e.g., provide the security and structure to the process of building the hardware design and microcode by controlling the access to the data files, and maintaining the most current copies of data files).
- Entries always correspond to the faults as the correction of faults require changes and changes are reported as entries and counted as faults (this clearly implies one change one fault not a necessarily true conclusion) and as most other libraries it does not account for the severity of faults.
- The bookkeeping libraries begin early in the development process, thus they can be considered as representative of the entire development cycle.
- It is more difficult, if not entirely impossible, to confuse design with other errors as the observed changes are part of what is used as the design of the system.

- There is no multiple logging of failures or faults.
- Given that a failure may be of the consequence of multiple faults, it is more representative of the "bugs" encountered during the development of a product.

From what has been discussed so far, we concluded that bookkeeping libraries offer a better approximation of history regarding the testing of computer systems. There is, however, a major drawback: Bookkeeping libraries are not designed to keep track of corrections of faults in a system. Thus, it may be difficult to extract the entries of libraries reporting changes in the design. The major challenge with this type of library, and at no exception, the libraries at our disposition, is to be able to answer the following question: Can the number of changes be extrapolated with an acceptable approximation? Before proceeding to answer this question, we discuss the library subsystems, as the existence of this question and its answer is entirely dependent on the library set up.

3 Collection of the Microcode and Hardware Library Data

Three databases are considered in this study created during the microcode and hardware development of the IBM 4381 and 9370 computer systems:

1) The microcode development bookkeeping library for the IBM 4381 computer systems
2) The hardware development bookkeeping library for the IBM 4381 computer systems
3) The microcode development bookkeeping library for the IBM 9370 computer systems

The microcode libraries of the IBM 4381 and 9370 computer systems, which are similar in structure, are accessed by a set of commands that allow a user to add a data file to the library (PUT command), to retrieve an existing file, (GET command), and to perform other type of maintenance to the library such as compile and release the code. A sequence of GET / PUT commands is usually used to retrieve, modify, and store a file back in the library. The libraries described here allow other commands as well and maintain additional information not pertinent to our discussion. All transactions to the library are automatically recorded by the system. A comment field of 40 characters accompanying a GET or PUT command was used to allow the user to describe the reason for issuing the command. The comments are filled by the developers to document the reasons for accessing a data file from the library and placing it back. This field is considered to be a key factor in assessing the relevancy of a library access to functional changes, since there was no other means of explicitly stating whether the entry pertains to functional changes. In addition to the above information, a library record contains a STATE field indicating the state of readiness of a data file with respect to the four test phases, namely,

- component test,
- unit test,
- subsystem test,
- and system test.

During the analysis of the microcode bookkeeping libraries, each record was classified into one of three categories, namely Irrelevant, Definite-Change and Possible-Change. The records in the Irrelevant category include all "routine" entries, such as "GET," "Promotes," and "History." The records in the Definite-Change category include the PUT entries which were promoted from the component test to the system test phase in three days, or less. This assumption is based on the notion that the new code was submitted to the library in an expedient manner as a mean to fix or patch an outstanding problem that was found during the system test phase. This category also includes the PUTs which are identified as patches (i.e., modifications to object or load files). The records in the Possible-Change category include all PUT entries that are not included in the Definite-Change and Irrelevant categories. This is based on the assumption that a PUT possibly signifies a change in the design because it is applied to resubmit an existing data file back into the library after it was retrieved by a GET. Table 1 shows the breakdown of the IBM 4381 microcode and hardware, and the IBM 9370 microcode PUTs into possible and definite changes.

TABLE 1
BREAKDOWN OF THE IBM 4381 AND THE IBM 9370 LIBRARIES

Library	Total Records	Total PUT Records	Possible and Definite Changes	Definite Changes	Possible Changes
4381 Hardware	17,811	7,874	7,874	228	7,646
4381 Microcode	136,016	29,955	27,473	3,632	23,841
9370 Microcode	475,909	85,572	65,160	4,662	60,498
Total Records	The total number of records in a library				
Total-PUTs	All PUT records in a library (for the IBM 4381 hardware library; this number includes only the Development PUTs)				
Possible + Definite Changes	All Possible and Definite Changes				
Definite Changes	The PUT records which were promoted from the component test to the system test phase in three days or less, and the PUTs, which are identified as patches				
Possible Changes	All PUT records not included in the Irrelevant records, and Definite Change Categories				

4 Analysis of the Bookkeeping Libraries by the Question Answering System

Table 1 indicates that the majority of the database records belong to the Possible-Change category. Given that the database is written by humans, in order to determine if the possible changes are indeed functional changes or routine accesses, we developed a database question answering system based on fuzzy logic described in [2]. The question answering system analyzes the comments written in a spoken language and determines whether the comments are related to a particular subject of interest. The system is depicted in Fig. 1.

In Fig. 1, the "Unique Word Generator" generates an alphabetically sorted list of all unique words contained in the comments of the PUTs in the Possible change category of the IBM 4381 microcode and hardware libraries. The "Word Processor" requires the manual processing of the words and the automatic examination of the database. During this manual process, the list of the unique words generated by the "Unique Word Generator" is analyzed in order to extract and place potential "relevant" words to functional testing in a table which is referred to as the **replace table**. We considered relevant words the words that are likely to be used in describing a functional change, such as "error," "fix," "bug," "change." The replace table contains two columns, the first contains the words as they appear in the database, and the second contains the synonyms corresponding to the words that appeared in the first column. All words in a database that match the first column of the replace table are replaced by the words on the second column, and all other words are deleted. As a result of this analysis, new "comment records" are generated and a second database is established. This database, which is referred to as the **modified database**, contains the words that are only present in the replace table. Using the Unique Word Generator and operating on the modified database, the list of all unique words in the modified database is generated, and it is referred to as the **relevant word table**. Subsequently, each word of the relevant word table is assigned a confidence value; a value between zero and one, which represents the degree of likelihood that a relevant word is used to describe an action associated with functional testing. Assigning a confidence value to a word may be achieved in many ways including surveys of participants in the creation of the database which was the case in our investigation.

Several hardware and microcode developers were asked to attribute a confidence value to each word in the word table which reflected his/her perception of the usage of that word, when used in a comment, in describing a functional error, a bug, or a change. Thus, for any given word, a multiplicity of values were collected corresponding to the opinion of people. To construct the membership grades for each word in the relevant word table from the responses of the survey participants, an algorithm was needed to determine the most expected value from the scaled values. This can be achieved by establishing the fuzzy expected value for each word in the relevant word table (FEV Processor) [7]. We considered a number of algorithms, including the FEV [8] and the WFEV [9], and developed a new algorithm, denoted as the Clustering Fuzzy Expected Value (CFEV), shown to have a superior performance for this type of application [10].

Using the clustering algorithm, the elements of a fuzzy set are grouped into separate clusters, and the population sizes and their mean are determined for each individual cluster. Then, the mean of the entire fuzzy set is evaluated and adjusted based on the population sizes and the mean of the formed clusters to form the CFEV value of each word in the relevant word list. A detailed description of the CFEV can be found in [10]. The CFEV algorithm computes the fuzzy expected value by:

$$CFEV = W_A + \sum_{i=1}^{m} \left(\frac{N_i}{N} \right)^2 \left(W_{Ai} - W_A \right) \quad (1)$$

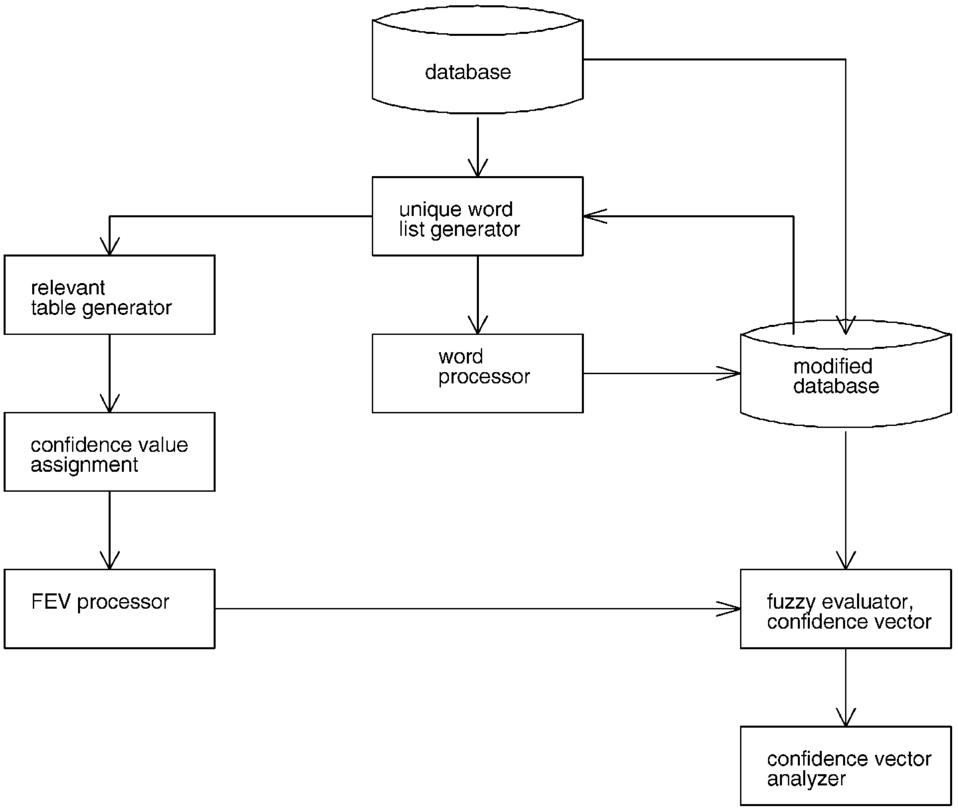
database
unique word list generator
relevant table generator
word processor
modified database
confidence value assignment
FEV processor
fuzzy evaluator, confidence vector
confidence vector analyzer

Fig. 1. The question answering system.

In (1), W_A is the mean of all responses to a particular word, N is the number of responses, m is the number of clusters produced from the data for each word, N_i is the number of responses in a cluster i and W_{Ai} is the mean of cluster i.

Based on the CFEV value of each word in the relevant word lists, the confidence of the comments in the libraries are computed based on [2] as follows: The "Fuzzy Evaluator" operates by analyzing each comment within the "modified database" in conjunction with the list of words in the "relevant word table" and the degree of confidence associated with each word:

1) For a comment with no relevant words, a confidence of zero is assigned to the entire record.
2) For a comment with one relevant word, the CFEV value of this word is assigned to the entire record.
3) For a comment with two relevant words, the confidence value of the entire record was based on the following function

$$\mu_A\left(w_i w_j\right) = \frac{1}{1 + ce^{-k(w_i w_j - l^2)}} \qquad (2)$$

In (2), $\mu_A(w_i, w_j)$ is the membership function attributing a degree of confidence with regards to A (functional testing) for a comment, where, w_i and w_j are the confidence values associated with two individual words i and j appearing in the same comment, k is a constant greater than 0, l is a constant between 0.00 and 1.00 indicating that words r that having confidence value $w_r > l$ are considered to favorably describe the subject of interest, while words having $w_r < l$ are considered to adversely describe the subject of interest, and words with $w_r = l$ are considered to be "neutral". The parameter c is defined as

$$c = \frac{1-m}{m},$$

where

$$m = \frac{w_i + w_j}{2}$$

(i.e., the average confidence of the two words i and j).

4) For a comment with more than two relevant words, the confidence value of the entire record may be computed by:

$$\Phi_{i=1}^{n-1} \mu_A\left[\left(w_i, w_{i+1}\right)\right] \qquad (3)$$

The operator Φ, for any given i, applies (2) with possible inclusion. The inputs to (2) are the confidence values attributed to words i and i + 1, and the output value of Φ for all the is between 1 and including n − 1 is a set of confidence values. Consequently, a confidence value is attributed to a comment by applying the following algorithm:

- Step 1: If there exists at least one element in the set produced by Φ that exceeds a given threshold value ρ_0 and the average of all pairs is $<l$, then the confidence value of the comment is assumed to be the MAX confidence value present in the set.
- Step 2: If step 1 does not hold true, and if there exists at least one element in the set produced by Φ, less than a given threshold value, ρ_1 and the average of all pairs is $<l$, then the confidence value of the comment is assumed to be the MIN confidence value present in the set.
- Step 3: If neither step 1 nor step 2 holds true, then the confidence value of a comment is assumed to be equal to the average of the confidences.

The detailed explanation of the above algorithm can be found in [2].

5 Establishing the Accuracy of the Data

The Question Answering System [2] was invoked with the threshold values ρ_0 = 0.66 and ρ_1 = 0.33, the k and l constants k = 7.00 and l = 0.50, and the d and s clustering parameters d = 0.10 and s = 0.20. The comments relevant to functional testing were extracted from each bookkeeping library. The outcome of the analysis of the bookkeeping libraries is shown in Table 2.

TABLE 2
ANALYSIS OF THE IBM 4381 AND THE IBM 9370 LIBRARIES

Library	Percentage of Tool Changes	Percentage of Tool + Definite Changes
IBM 4381 Hardware	38	40
IBM 4381 Microcode	48	55
IBM 9370 Microcode	25	31
Percentage of Tool Changes	The percentage of records assessed as related to functional changes based on the number of Possible changes.	
Percentage of Tool + Definite Changes	The percentage of Tool and Definite changes based on the number of Possible and Definite changes.	

There are three validations to be made regarding the accuracy of the number of changes, considered in this presentation as faults, namely:

1) The accuracy of the tool.
2) The accuracy of the tools in the library entries considered to report possible changes.
3) The overall accuracy of the data.

To answer the questions, it is of interest to operate on representative data and measure the accuracy by measuring the tools' responses with the expert human assessments. While human assessments can be subjective in the absence of better library systems and the potential inability of creating "objective" entries in future libraries (library entries will always reflect the opinion of the person determining the "meaning" of what constitutes an entry to the library before an entry is established) they represent the best approximation.

In attempting to answer the above questions without resorting in manual evaluation of the entire databases, an upper bound of comments to be evaluated manually by a human expert in the field had to be established. With the man-power and time at disposition, we decided to evaluate no more than 10,000 comments. In determining the experimental databases, we proceeded as follows: First, we extracted the profiles, in terms of number of words, of all the databases in the library, before and after the application of the tool. It was immediately observed that after the application of the tool the comments with zero relevant words were as low as 19.45 percent, and as high as 39.99 percent (see Table 3). This discovery provided a complication in the composition of the experimental databases. Given that, the percentages indicate that as much as 40 percent of the entries have been considered to be routine accesses (i.e., they have zero relevant words) by the tool, it was imperative to guarantee that the tool was highly successful in excluding these comments imposing the consideration of more comments containing zero relevant words than initially anticipated.

The previous discussion was incorporated in the three databases (DB1, DB2, and DB3) with their composition described in Table 4. The three databases, DB1, DB2, and DB3, were extrapolated from the IBM 4381 hardware and microcode, and the IBM 9370 microcode Possible Changes, respectively. We selected the entries randomly and made sure the entries appeared only once. The characteristics of the databases are described in Table 5. A number of things should be noted from the characteristics of the databases, namely:

TABLE 3
FREQUENCIES OF RELEVANT WORDS

Number of Relevant Words in a Comment	IBM 4381 Hardware Possible PUTs (7,646 Records) Percentage of Relevant Words	IBM 4381 Microcode Possible PUTs (23,841 Records) Percentage of Relevant Words	IBM 9370 Microcode Possible PUTs (60,498 Records) Percentage of Relevant Words
0	25.53	19.45	39.99
1	48.34	41.34	39.93
2	21.24	27.50	15.07
3	4.34	9.40	4.17
4	0.54	2.09	0.66
5	0.01	0.18	0.15
6	0.00	0.04	0.03

TABLE 4
NUMBER OF RELEVANT WORDS IN THE EXPERIMENTAL DATABASES

Database	Zero Relevant Words	One Relevant Word	Two Relevant Words	> Two Relevant Words	Total	Possible Changes
DB1	1,400	1,250	635	165	3,450	7,646
DB2	2,230	1,650	930	440	5,250	23,841
DB3	505	560	180	55	1,300	60,498

TABLE 5
PERCENTAGES OF RELEVANT WORDS IN THE EXPERIMENTAL DATABASES

Database	Percentage of Zero Relevant Words	Percentage of One Relevant Word	Percentage of Two Relevant Words	Percentage of > Two Relevant Words	Percentage of Possible Changes
DB1	40.58	36.23	18.41	4.78	45.12
DB2	42.47	31.43	17.72	8.38	22.02
DB3	38.85	43.08	13.84	4.23	2.15

TABLE 6
FREQUENCIES OF RELEVANT WORDS BASED ON ALL THE POSSIBLE CHANGES

Database	Percentage of Zero Relevant Words	Percentage of One Relevant Word	Percentage of Two Relevant Words	Percentage of > Two Relevant Words	Percentage of Possible Changes
DB1	71.72	33.82	39.10	44.12	45.12
DB2	48.09	16.74	14.18	15.76	22.02
DB3	2.09	2.32	1.97	1.81	2.15

- We considered almost half the comments (45.12 percent) of the IBM 4381 hardware.
- We considered more than one fifth comments (22.02 percent) of the IBM 4381 microcode.
- We considered only 2.15 percent of the IBM 9370 microcode.
- Clearly, the experimental database extrapolated from the IBM 9370 microcode as indicated earlier can be argued to be nonrepresentative.
- Even though the composition of the databases did not reflect the actual characteristics, we have evaluated almost half the comments in the IBM 4381 hardware and more than one over five of the IBM 4381 microcode suggesting that if the tool was close to the human evaluation, then the confidence associated with the success of the tool should be high.
- The composition of DB1 and DB2 (especially DB1), reported in Table 6, suggests that we have evaluated more than satisfactory comments with zero relevant words (we considered 71.72 percent for DB1 and 48.09 percent for DB2 of the comments with no relevant words in the entire IBM 4381 hardware and microcode databases). Furthermore, for at least DB1, we have operated on substantial overall percentages of all the comments (Table 6). For DB1, we considered for examination 33.82 percent of the overall comments left in the entire IBM 4381 hardware database after the examination containing one relevant word, 39.10 percent containing two relevant words, and 44.12 percent containing three or more relevant words.

The previous discussion suggests that high degree of accuracies in DB1 of the tool suggest a high degree of confidence in the final outcome for at least in the IBM 4381 hardware database and potentially by extension a high degree of confidence to the other libraries shown similar accuracy of the tool. In conducting the discussion of the experimental databases, it should be noted that we were interested in the overall accuracy regarding the closeness of the evaluation to the human evaluation. In other words, we wanted to evaluate the percentage of records that correspond to

TABLE 7
TOOL VERSUS MANUAL EVALUATION (IN PERCENTAGES)

	DB1	DB2	DB3	DB1.0	DB2.0	DB3.0
Tool Accuracy	96.09	95.70	95.86	97.79	99.37	99.41
Tool Changes	31.51	35.45	23.30	0.00	0.00	0.00
Manual Changes	30.55	35.51	21.82	2.21	0.63	0.59
|Tool - Manual|	0.96	0.06	1.48	2.21	0.63	0.59

functional changes versus the percentage of records that did not, and to evaluate the closeness of the percentages between the tool and the manual evaluation. To evaluate the tool accuracy, it is necessary to first evaluate the tool error and consequently use such an error to represent the disagreement between the manual evaluation and the tool and produce the agreement between the two evaluations which constitutes the tool accuracy. The tool accuracy can be found in the first row of Table 7, which was compiled to include the following as tool errors:

- The tool is in error when it includes comments that are not considered as functional changes by the manual evaluation.
- The tool is in error when it excludes comments that are considered by the manual evaluation as functional changes.

The findings indicate that the agreement between the tool and the manual evaluation is as low as 95.70 percent and as high as 96.09 percent (for the three experimental databases, DB1, DB2, and DB3). This indicates that the tool is very accurate in its decisions to separate the database entries in relevant and irrelevant to functional testing regions. Finally, it is of interest to identify if a database is pertinent to the functional testing by establishing the percentage of pertinent comments to the functional testing. The second and third row of Table 7 report such percentages for the three experimental databases (DB1, DB2, and DB3) for functional changes computed for the tool (second row) and the manual evaluation (third row). The fourth row reports the absolute value of the deviation between the tool and the manual evaluation. The findings indicate that the tool and the manual evaluation are very close among each others indicating that the tool can be used to establish the relevance of a database to the functional testing. Given that the excluded comments were a concern, in Table 7 (columns DB1.0, DB2.0, and DB3.0) we also report the result of the evaluation for the comments containing zero relevant words. Given that the lowest percentage is 97.79 percent, it can be suggested that the tool when it considers entries as routine accesses to the database, then there is a high degree of certainty that they are indeed routine accesses. In conclusion, the experimentation strongly suggests that the tool will operate more than satisfactory. We note here that we performed additional experimentations regarding the actual database composition and overall validation. The results are in accordance with what is reported here. Interested readers are referred to [11].

6 CONCLUSION

In this paper, we first identified a number of issues related to the functional testing. In particular, we addressed the issues related to the definition of “faults,” “errors,” and “failures” and their separability to the various development processes, the dilemma of the research data and the choice of the database that provides the most confidence in the reflection of the entire development cycle. Consequently, we discussed the assessment of the IBM 4381 microcode and hardware and the IBM 9370 history libraries, two databases containing more than half a million records, and established their relevancy to the study of functional changes by applying a fuzzy reasoning database question answering system [2]. As a result of this assessment, it was concluded that the libraries are pertinent to functional testing based on the percentages of the relevant records in the IBM 4381 microcode and hardware and the 9370 microcode bookkeeping libraries and the error data used for example in [1] were extracted. While the confidence associated with the tool and the overall data could be considered to be high, we caution the reader to consider the final results of our analysis as an approximation of the error data. As a final note, we indicate that in the absence of better databases and a precise common methodology, approximations is what can be done today in the arena of error data. We hope that the investigation reported in this paper will help in the discussion for future improvements, related to the maintaining of databases, and provide grounds for discussion to achieve a common methodology in maintaining commonly acceptable good error data so that extensive and complicated extrapolations and validations of approximately good error data is avoided.

REFERENCES

[1] G. Triantafyllos, S. Vassiliadis, and W. Kobrosly, “On the Prediction of Computer Implementation Faults via Static Error Prediction Models,” *J. Systems and Software*, pp. 129–142, Feb. 1995.

[2] S. Vassiliadis, G. Triantafyllos, and W. Kobrosly, “A Fuzzy Reasoning Database Question Answering System,” *IEEE Trans. Knowledge and Data Eng.*, pp. 868–882, Dec. 1994.

[3] A. Avizienis and J. Laprie, “Dependable Computing: From Concepts to Design Diversity,” *Proc. IEEE*, vol. 74, no. 5, pp. 629–638, May 1986.

[4] V.R. Basili and B.T. Perricone, “Software Errors and Complexity: An Empirical Investigation,” *Comm. ACM*, pp. 42–52, Jan. 1984.

[5] T.J. Yu, V.Y. Shen, and H.E. Dunsmore, “An Analysis of Several Software Defect Models,” *IEEE Trans. Software Eng.*, vol. 14, no. 9, pp. 1,261–1,269, Sept. 1988.

[6] D. Potier, J.L. Albin, R. Ferreol, and A. Bilodeau, “Experiments with Computer Software Complexity and Reliability,” *Proc. Sixth Int’l Conf. Software Eng.*, pp. 94–103, 1982.

[7] L. Hall, S. Szabo, and A. Kandel, “On the Derivation of Memberships for Fuzzy Sets in Expert Systems,” *Information Sciences*, vol. 40, pp. 39–52, 1986.

[8] A. Kandel, *Fuzzy Math. Techniques with Applications*. Addison-Wesley, pp. 72–101, 1986.

[9] M. Friedman, M. Schneider, and A. Kandel, “The Use of Weighted Fuzzy Expected Value (wfev) in Fuzzy Expert Systems,” *Fuzzy Sets and Systems*, vol. 31, pp. 37–45, May 1989.

[10] S. Vassiliadis, G. Triantafyllos, and G.G. Pechanek, “A Method for Computing the Most Typical Fuzzy Expected Value,” *Proc. Third IEEE Conf. Fuzzy Systems*, pp. 2,040–2,045, June 1994.

[11] S. Vassiliadis, G. Triantafyllos, and W. Kobrosly, “Establishing the Relevancy of the Bookkeeping Libraries to the Functional Testing of Computer Implementations,” IBM Technical Report TR 01.C728, Endicott, N.Y., p. *37*, Apr. 1993.