I.J. Information Technology and Computer Science, **2013, 08, 15-23**
Published Online July 2013 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijitcs.2013.08.02

Fault Persistency and Fault Prediction in Optimization of Software Release

Maurizio Pighin
Department of Mathematics and Computer Science, University of Udine, Italy
E-mail: maurizio.pighin@uniud.it

Anna Marzona
LiberaMente Srl, Udine, Italy
E-mail: amarzona@mentelibera.it

***Abstract*—** This article serves two purposes: firstly, it presents an innovative methodology that increases the accuracy of fault prediction measurements. This method is based on the novel concept of "fault persistency", which enables to correct prediction metrics with a weighted value related to the module's history. Secondly, it aims to develop operational processes from the aforesaid prediction metrics that may contribute to software construction and validation. It presents an example of an allocation methodology for resources used for testing purposes. The theoretical part is followed by an extensive experimental phase.

***Index Terms*—** Fault Prediction, Fault Persistency, Software Release, Software Metrics, Software Testing

I. Introduction

Our interest has been long focused on the use of software metrics as a methodology evaluating the risk of a module, where risk is related to the number of faults still contained in the module. Activity in this sector is stimulated both by theoretical interest and by the intention of finding methodologies that may validate with factual data the subjective evaluations adopted in software engineering. Our attention is devoted to methodological and operational aspects concerning applicability of the measurement and prediction processes in real working environments.

The scientific background of our work is related to reliability and risk measures of software modules. The size and complexity of software has grown dramatically during the last decades and especially during the last few years. When the requirements for and dependencies of computers increase, chances of crises from failures also increase. The impact of these failures ranges from inconvenience to economic damages to loss of lives - therefore it is clear that software reliability is becoming a major concern for software engineers and computer scientists. Software development is a complex process in which software faults are inserted into the code during the development process or during maintenance. The literature on this subject shows that the pattern of faults insertion phenomena is related to measurable attributes of the software objects. During the last twenty years, hundreds of metrics have been proposed for software assessment.

Currently one of the most commonly used method for evaluating module fault proneness is to adopt a range of complexity-based metrics. Principal measurements are mainly capable of predicting the total risk of a module in terms of number of faults that the module might produce in its life-cycle. However, often the interest is focused on the degree of risk at a given project release phase, e.g., in order to assess risk when the product is released on the market.

In this article we present a new approach to fault proneness measurement where the degree of risk of a module is calculated in relationship to a specific release. Then we present an example of use of the new measurement in a real environment.

Chapter 2 describes the state of art of research about fault prediction measurements, classification model construction and accuracy evaluation of a model. Chapter 3 presents the characteristics of fault measures not operating on the whole software life cycle, but working on each single software release and the rationale of an application of this methodology on software testing planning. Chapter 4 describes the experimental validation of the whole model previously defined.

II. Fault Prediction Measurement Systems

In an attempt to optimize product development and test for the construction of high-quality software, project managers and programmers try to identify the elements that are most likely to experience problems when in use, in order to focus the releasing activity on them. Attention is usually concentrated on new modules, on pre-existing functions heavily modified by the current release, on the work delivered by the less skilled technicians. These elements are supposed to contain a high number of defects and therefore are intuitively considered potentially highly faulty. But other modules can be critical, even if they do not belong to this first set. In this paper we propose a methodology to find them.

Prediction systems are composed of metrics and a classification model: metrics provide factual measurements of a single element, while classification model partitions these elements into classes or assigns them a certain degree of risk. In literature there are several fault prediction models. Some approaches (for example [1]) evaluate the characteristics of the project documents, identifying any critical element already during the specifications phase. Other research activities (such as [2]) are focused on the analysis of some characteristics, such as the number of people who were involved in module development, the testing time, and the faults detected in the previous phases; on the basis of these measures it is possible to obtain information on the degree of risk of a single module.

We will focus on prediction systems based on metrics of the software manufactured, as they are adopted much more frequently.

2.1 Software Complexity Metrics

Most of the research activity carried out on complexity concerns structural complexity, i.e., the measurement of the characteristics of the final product in the software development process, which can give an idea of the module difficulty. As programming languages are very formal, the measurement of module structural characteristics is widely recognized as "objective", easy to repeat and to adopt in different environments. Over the years various metrics systems have been proposed to define the structural complexity of software. Additionally, several experiments have been carried out for the purpose of proving in an objective way the relation between high complexity and the faultiness in the code. The most important methodologies are length-based metrics [3,4], instruction flow metrics [5], multivariate systems metrics [6,7,8,9,10,11,12] and entropy and the informational content metrics [13,14].

Here we shortly outline the key elements of the RPSM (Risk Predictive Structural Metric) because we will use this metrics in following sections (a broad definition and validation is given in [12]). RPSM is a multivariate metric proposed by our research group. It is based on a set of parameters that can be broken down into the different classes: flow control instructions, memory allocation instructions, definition and usage of structured variables, preprocessing instructions, function calls, size. We measured these parameters on each software module, and then we combined them with the fault found in the same module; using this methodology we found relations between faults and structure of a module and we descended a mathematical model to evaluate the risk of each parameter and the global risk of each future module in a defined environment. RPSM examines several different aspects of software structure, becoming a powerful predictor of the total number of faults collected by a module all over its life cycle. RPSM, as other structural metrics, does not consider the age of the module: it can be used as a total fault number predictor and it's not directly useful in predicting the expected fault number of a module in a given project release.

2.2 Classification Systems

Classification systems may be based on several methods: for example the use of threshold values identified by means of statistical methods such as the discriminating analysis that divide the set of elements into classes on the grounds of the metrics values [6,11]; the calculation of an index expressing the probability of belonging to a certain class on the basis logistical regression [10], or a classification by means of Bayesian networks [15], linear programming techniques [16] , decision trees [7] or neural networks [17], etc..

The information provided by the measures obtained from metrics is processed by the classification system, which may produce, according to the model on which it is based, a class or a continuous risk value. A metric system can be a more or less effective fault predictor according to the classification model used in the prediction system. For example in [18] we calculated that the degree of accuracy of the McCabe index combined with a threshold classification technique is around 75/80%; the degree of precision of the RPSM multivariate metrics ranges from 85 to 90% if combined with a linear programming classification technique, while it decreases slightly if the classification is carried out on the basis of a threshold value.

We now shortly outline the classification models construction and the concept of model accuracy on which we based our experiment.

2.2.1 Classification Model Construction

Classification models are usually constructed using assisted learning techniques: the user analyses a significant set of data, with a known classification, and divides it into two complementary subsets: the so-called training set, or learning set, used to infer the classification rules, and the testing set, or control set, which is used to measure the effectiveness of the classification system.

Let us consider, for example, the construction of a model that enables to predict whether a given software module is highly fault-prone or not. From historical data we can indicate for each module whether it falls into the "HR" (High Risk) or "LR" (Low Risk) class on the basis of number of faults contained in the module.

A dichotomist classification model can be constructed with the following steps:

1. through a random selection function the set is divided into two subsets: TS (testing) and TR(training);
2. TR is divided into the HR and LR classes on the basis of the classification value known; this method is called subsequent classification (or a posteriori classification), because it is based on a known value;
3. the characteristics of the HR and LR modules are analyzed;
4. from the common characteristics a generalized M classification model is derived;
5. M is applied to TS data, obtaining a classification into two classes, HR' and LR'; this method is called a priori classification, because it is carried out on the basis of a model without having data that support the correctness of the classification;
6. the accuracy of the a priori classification is checked with a subsequent classification dividing TS elements into HR and LR classes and analyzing how many HR and LR elements are really contained respectively in HR' and LR'.

2.2.2 Accuracy of the Classification System

The accuracy of a classification model is determined by several factors: especially in dichotomist systems it is measured as follows: (i) the percentage of elements correctly classified by the model (true positive and true negative elements); (ii) the percentage of false positive elements (the so-called Type I errors): in this case elements without faults classified as high-risk; (iii) the percentage of false negative elements (the so-called Type II errors): in this case faulty elements classified as low-risk. When evaluating the accuracy of a prediction method it is also important to consider the type of errors concerned. For example, if we use fault prediction values to plan testing activities, Type I errors are better than those classified as Type II, as they imply an accurate testing of an element which does not need it; vice versa, in case of Type II errors, highly fault-prone modules are classified as low-risk, and therefore are not tested accurately: it is important to have a good global accuracy and eventually a better accuracy in LR' that in HR'.

III. The Rationale of the Research

After the short description of state of art, we start now analyzing the core of our research. One of the major limitations to the use of structural complexity metrics is the fact that risk predictions made on the basis of the total number of faults in the training set is referred to the full module's life-cycle; usually, however, the intention is from fault proneness evaluation models to obtain information about the risk for the current software release. The evaluation of fault allocation according to software age has only recently become object of systematical investigation: even though some trends, such as "a module that had a lot of faults in the past is likely to have them also in the future", were outlined in previous studies, only in 2000 did Graves [19] propose a process metrics which considers age a key factor in the assessment of fault proneness: risk is calculated starting from the number of modifications made in time weighted with the module age and introducing a corrective element that halves the weight of the older modifications. The studies by Ostrand and Weyuker, applied to the development teams of AT&T, are more systematic [20]: they have been carried out concurrently with the starting of our research. In both cases initially the approach was based on the analysis of fault allocation per release, investigating the fault trend, the correlation between the structural characteristics of the module and the number of faults in a release, the persistency of faults from one release to the following one [21], the incidence of age of a module on the number of faults in a given project release. The results obtained by the two groups working on completely different environments are basically consistent and point out some general trends briefly outlined at section 3.1. As far as we are concerned, knowing such trends enabled us to create a method for correcting structural metrics-based predictions thanks to the use of a regression function calculated on the average density of faults per release. This method is outlined in the following sections.

3.1 Characteristics of Fault Partitioning per Release

The analysis of fault trends through releases was the purpose of the first step in our research [21]. The characteristics analyzed were the following.

Age: for each release we created two subsets: one with the files introduced in the release concerned (new) and one with the files already included in previous releases (old); the experimental analysis (surprisingly) demonstrates that *the new files do not show a significantly higher rate of faults than the old files*;

Faultiness in the first release: files that in their first release contained a number of faults exceeding a certain level were classified as faulty files; the analysis shows that *faulty files tend to maintain a higher fault density in the following releases than non-faulty files*; we called this phenomenon "*persistency*". This idea confirms studies previously carried out by Ostrand and Weyuker [20]. They summarized their findings with the sentence "*once faulty ever faulty*", which means that when a file has a high fault density in its first release (once faulty) it tends to maintain a high fault density in subsequent releases as well (ever faulty). In [21] we verified experimentally this phenomenon.

Total file faultiness: in literature the 80%-20% principle is well-known (80% of faults are in 20% of modules), which is represented in the Pareto distribution model; this rule holds true also in our environment, and we *have found out that modules with a large number of faults had a much higher concentration of faulty files than the whole project;*

Structural complexity of files: the analysis showed that *faulty files tend to have a slightly more complex structure than non-faulty files.*

3.2 Fault Prediction per File Release

We have also made some research on the fault trend in following releases, which is represented by a decreasing curve that is different for faulty and non-faulty files. Our innovative idea was to combine the two elements (structural complexity measures and persistency) in order to evaluate the fault proneness of a specific module release. This new measurement is useful for various application, for example better allocating integration testing time before the release of a new project version. In our study we aim to validate the following hypotheses:

- a prediction measurement based on single modules complexity can become even more accurate with the introduction of a factor connected to each module's release
- we can use this predictions in real world application: as example we propose a testing time partitioning weighted on the basis of each module fault proneness and we evaluate the benefit which this approach can introduce in resource allocation.

Now the issue is how to measure the degree of risk of a current release and how to allocate testing time on the basis of risk measurement. The new fault prediction measure proposed for a version to be tested is obtained by weighting one of its structural measurements with other factors, including one related to age.

The set of elements that in our opinion contribute to measuring module risk of a given release can be calculated as

$$R_{j,k} = S_j * G_j * C_j * F_{j,k} \tag{1}$$

where:

- *j* is a project module to be tested;
- *k* is the module release, that is the age of a module in terms of releases (the maximum value for *k* is equal to the current release and the minimum is 1 when it is new);
- S_j is the subjective module risk, measured by the person in charge of the testing activity according to the most unpredictable (and difficult to formalize) conditions, such as the competence of the developer, the functional criticality of the module, the kind of users, the attention dedicated to customer expectations, problems related to software specifications, etc.;
- G_j is the module test gravity factor, i.e., a measurement of the difficulty of the testing activity;
- C_j is the absolute module risk factor, calculated as a complexity measurement of the module *j*: it can be for example the McCabe index or any other structural measurement;
- $F_{j,k}$ is the release correction factor; it is calculated applying the persistency function on the module j in its kth release,
- $R_{j,k}$ is the corrected risk factor calculated for the module j in its kth release, according to the formula: it is the risk measure we used to classify modules by risk.

Our first aim is to demonstrate that introducing the correction factor connected to the module age it is possible to identify the more fault-prone modules in the current release with greater accuracy than with a classification based exclusively on the C_j risk measurement. For this purpose we simplify the function proposed by omitting the subjective factor S_j and the gravity factor G_j, considering both equal to 1, so as to emphasize the contribution of the correction factor $F_{j,k}$.

Therefore the simplified function for fault prediction becomes

$$R_{j,k} = C_j * F_{j,k} \tag{2}$$

In order to demonstrate the effectiveness of the age-based correction factor we create two classification models, M and M'. The former is based on the new measurement, the second one exclusively on the complexity measurement. At the end of the classification we have compared the two models' accuracy. More in detail, we follow these steps:

- We make a random partition of the project modules and test the model creating two different subsets (TR – Training Set, and TS – Testing Set)
- We use TR elements to define the classification model through: *(i)* the construction by means of regression methods of persistency functions F to be used for the calculation of the correction factor related to the file age and to whether the file is faulty or non-faulty; *(ii)* the calculation of C_j and $R_{j,k}$ for each module *j*; *(iii)* the construction of *a priori classification* model M on the basis of $R_{j,k}$ and the number of faults detected in the current release of each module *j*; *(iv)* the construction of *a priori classification* model M' on the basis of the complexity metrics C_j only and the number of faults in each module j current release.

- We use TS elements to validate the classification model through: *(i)* the calculation of $R_{j,k}$ and C_j for each module *j*; *(ii)* the *a priori classification* of each module applying to $R_{j,k}$ the Model M; *(iii)* the *a priori classification* of each module applying to C_j the Model M'; *(iv)* the assessment of the degree of accuracy of M and M' by counting the faults really detected in each TS module *j*, and the comparison of the results.

The experiment's details and results are described at section 4.

3.3 Examples of Practical Use of Risk Measure

The measure described in the previous paragraph can be used for different purposes: operational processes should consider the activity necessary to take this measure ad to use it. The decision to use the risk based measure influence, for example, the following items. *(i)* Configuration management and quality management documents: they must describe processes, documents, parameters and actions related to the acquisition and the use of the measure. *(ii)* Daily development activity: the measure can be taken of software modules under development to early evaluate their potential faultiness. *(iii)* Integration test: the measure can be used to allocate more testing time to the most risky modules (see par. 3.4 and 4.6 for an example)

The process must be supported by a data collection system, to store collected measures and modules attributes. The parameters of the model must be periodically recalculated, in order to tune the model and to reflect the changes of the organizational structure of the company. If there is no historical data, the process can be set up using data collected in the development of systems similar for technology, team, etc, to calculate the initial value of model's parameters. As the process runs, model's parameters can be regenerated by using actual measures collected during the process, until they converge to stable values.

3.4 Risk-Based Testing Time Partitioning of a Given Release

This part of the study must be considered an example of application of our risk evaluation in real world.

Our starting point is the limit that each software house has for testing time: in order to test a given release usually they have a limited amount of time that must be used to achieve an optimal result in terms of overall project reliability.

If we consider the time available as partitionable into T *units* of tests that can be allocated in a discrete way, the problem can be solved with the following equation system

$$\begin{cases} \mathrm{SUM}_{jk}(T_{j,k}) = T \\ T_{j,k} = \mathit{Fun}\,(NF_{j,k}) \end{cases} \qquad (3)$$

$T_{j,k}$ is the number of units to be partitioned in order to test the j^{th} module in its k^{th} release. $NF_{j,k}$ is the number of fault presents in j^{th} module in its k^{th} release; *Fun* is a function that enables to allocate a greater amount of time to more fault-prone files. When we plan tests, we do not know $NF_{j,k,}$ so we approximate its values with $R_{j,k}$, obtaining the equation $T_{j,k} = \mathit{Fun}\,(R_{j,k})$.

Our target is to demonstrate that testing time partitioning to modules on the basis of $R_{j,k}$ (*weighted* partitioning) enables to reach a more effective partitioning of testing units to faults than a generic flat one to each module: note that we used as reference a "flat" partition due to the fact that in our experimental environment the modules were constructed using rigid software engineering rules, so where all of similar length and so we have not "a priori" motivation to test with different resources different files; in other experimental environments it is possible to use as reference other partition methodologies (usually based on dimension or other similar parameters).

IV. Experimental Validation

The whole experiment has been carried out on a real project, i.e., an integrated company management system developed in 10 years by a software house employing 20 programmers. We have information on the composition of groups working on thematic subset of modules (i.e., on Accounting, on Production, on Logistics, …). We know also that all people worked on the development of the standard edition of the software, on its customization and on software maintenance, but we have no detailed information on each developer's work, nor on his personal characteristics.

The software in its latest release is made up of 1,061 modules in C language (amounting to a total of more than 450,000 lines of code.), it runs on Unix environment and interfaces an Informix database. It has been provided to us in its most recent version with a list of releases and a list of faults found and corrected between one release and the other. The development environment adopted at first was very innovative and did not change during the period mapped by our data collection.

The set of files in the current release has been partitioned between the two TR set, which contain 40% of files, and TS, which contain all the others. TR has been used for the classification model construction and TS for all the remaining evaluations. The experiment has been repeated on four different random partitions of the sets of TR and TS files in order to have more significant results from a statistical point of view. The experiment has been carried out on CC (Cyclomatic

Complexity, the McCabe index) and RPSM. Therefore totally we have analyzed 8 test cases, and the results confirmed our expectations.

4.1 Calculation of Persistency Functions

For each of the four distinct test sets, persistency functions have been evaluated as follows:

1. partition of TR files between Faulty and Non Faulty classes;
2. calculation of average fault density per release on each of the two classes;
3. interpolation of density values per release with a 4th degree polynomial regression function selected to describe faithfully but without ups and downs density trend in time;
4. for first-version modules, whose faultiness class in their first release is not known, the correction factor adopted has been the average result obtained from the interpolation functions in the first release. An example of the function graphs calculated on the two sets is shown in Fig. 1.

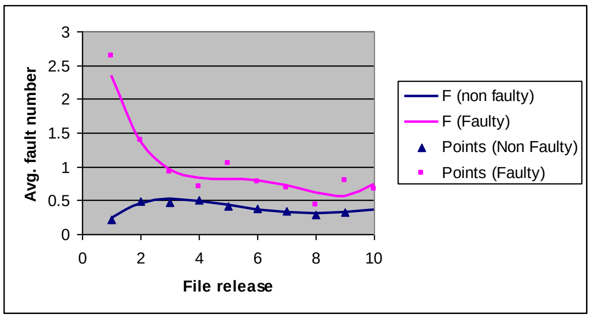

Fig. 1: Graphs of fault density functions per release

4.2 Calculation of Structural Metrics C

The experiment has been carried out correcting the two different structural metrics CC and the RPSM. We used in the present experiment these different metrics because the first is very famous and well known, while the second is a new metric proposed by our Laboratory.

The experiment has been carried out correcting the two different structural metrics CC and the RPSM. We used in the present experiment these different metrics because the first is very famous and well known, while the second is a new metric proposed by our Laboratory. As we could see in the following sections, the results are very similar and other evaluation with other metrics obtained also similar results: we assess that the results depend on the methodology applied, not on the specific complexity metric used. The RPSM calculation model has been constructed on the basis of the elements in TR: from structural measurements and the file faultiness we have calculated the relative weight of each structural parameter.

4.3 Construction of Classification Models M and M'

We have decided to use as dichotomist classification model a threshold value identified by means of a logistical regression function, a value that enables us to partition the files into the HR and LR classes. The regression function used as a basis for the calculation is a linear regression on average complexity values of the elements with the same number of faults in their current release: we have partitioned TR into four classes on the basis of the number of faults (0, 1, 2, >2). The partitioning into four classes has been determined on the grounds of the fault distribution per release.

Table 1: Number of Files per Fault Class

Number of faults	Average % of files in TR
0	72%
1	18%
2	6%
>2	4%

Then we have calculated the average complexity (Cm_0, Cm_1, Cm_2, Cm_3) of the elements contained in each of the four classes and then we have calculated the regression curve coefficients at (Cm_0, 0), (Cm_1, 1), (Cm_2, 2), (Cm_3, 3). The threshold for the partitioning of files into the HR and LR classes has been calculated searching the zero of the third derivative of the logistical regression function applied to the regression line: this value represent the point where the distribution of fault change drastically its behavior and so can classify correctly the two classes; similar analysis where used by Denaro and Pezzè [10]. For each of the four TR sets and for each of the two complexity measures we have calculated the threshold values with this method

- T_r: the discriminating factor of the Model M, based on the structural complexity measurement corrected with the persistency factor ($R_{j,k}$);
- T_c : the discriminating factor of the model M', based on the non-corrected structural complexity measurement (C_j).

4.4 A Priori Classification of TS Elements

For each TS element we have calculated

1. the structural complexity measurement C_j (McCabe and RPSM);
2. the persistency correction factor $F_{j,k}$, by means of persistency functions;
3. the corrected complexity measurement $R_{j,k}$ (McCabe and RPSM).

With the two measurements we have divided the elements into the two HR'_r and LR'_r classes on the basis of the model M comparing the value of $R_{j,k}$ with the threshold value T_r.; then we have created a new partitioning into the HR'_c and LR'_c classes on the basis of the model M', comparing for each model the value of C_j with the threshold value T_c.

4.5 Evaluation of Classification Accuracy

Model accuracy has been evaluated on the basis of the overall classification accuracy (number of files correctly classified) which is better detailed with an analysis of the classification accuracy in the highly faulty class. The analysis of data enabled us not only to see the difference between the two models in terms of classification accuracy, but also the variation in the number of elements in the highly faulty class.

Table 2: Classification accuracy comparison

	M' (CC)	**M (CC Corrected)**	**Variation %**	**Improv. %**
A	87.1%	92.2%	+5.9%	39.5%
B	35.6%	56.0%	+57.3%	31.7%
C	10.7%	8.8%	-17.8%	
	M' (RPSM)	**M (RPSM Corrected)**	**Variation %**	**Improv. %**
A'	85.0%	91.0%	+7.1%	40.0%
B'	47.2%	64.7%	+37.1%	33.1%
C'	14.1%	11.4%	-19.1%	

The results are detailed in Table 2: it presents detailed results of HR', which is the worst classified class. The table shows in separate columns the degree of accuracy measured with the classification methods M' and M, indicating the difference between accuracy measurements in percentage both in terms of variation (M – M')/M', and improvement (M-M')/(1-M').

4.6 Effectiveness Assessment of Testing Time Weighted Partitioning

In this section we present, as an example of the application of fault prediction, a partitioning of testing modules on the basis of the complexity measure corrected with persistency. We divided the range of $R_{j,k}$ in n steps $V_1..V_n$. We allocated the total testing time T on the basis of the following equation system

$$\begin{cases} T_{j,k}= A_1*Class_1(R(_{j,k}))+A_2*Class_2(R(_{j,k})) +... \\ \quad ..+ A_n*Class_n(R(_{j,k})) \\ SUM_{j,k}(T_{j,k}) = T \end{cases} \quad (4)$$

where a module *j* at its release *k* belongs to $Class_i(R_{j,k})$, and consequently $Class_i(R_{j,k})=1$, if $R_{j,k}$ belongs to V_i and 0 otherwise; $A_1, ..., A_n$ are calculated in order to make the testing time per fault homogeneous among the various classes.

The method has been validated by calculating subsequently for each risk class the average number of testing units per fault obtained with this time partitioning method; the result has been compared with the average number of units per fault obtained with a flat partitioning.

The general problem is very complex, and we analyzed different number of partitions. At the end we obtained good results adopting a simplified version with a partition into three equidistant classes: augmenting the number of classes the enhancement of results is not very significant in the context. Partitioning into classes has been carried out normalizing the risk measurement and reducing it to an interval between 0 and 10. Then we have partitioned the modules into classes identified by means of equidistant thresholds.

Table 3: Files partitioning into risk classes

Class (Risk)	**Num. of files**	**Total Num. of faults**
High	17	30
Medium	64	59
Low	552	136

The time partitioning test was carried out on the overall data risk calculated by means of the CC index corrected with the persistency factor. We have considered a real case in which the testing time available was 160 hours partitioned into 10-minutes testing units, for a maximum number of 960 units. We have simulated the distribution of testing time on the basis of the time allocated to each class. We have experimented three different criteria of class weight and compared them (a flat distribution and two different weighted distribution).

From the results shown in Table 4 emerge the partitioning of average time values per fault into the various classes. Weight indicates the weight allotted to the risk class, while Average is the average number of units per fault, calculated with a subsequent evaluation. We can see that Weighted Distribution 2 have an almost equally average unit per faults and so is the optimal solution

Table 4: Testing time partitioning into risk classes

Weighted distribution 1			
Class (Risk)	**Weight**	**Testing Units**	**Average units/faults**
High	3	66	2.20
Medium	2	165	2.80
Low	1	713	5.24
Weighted distribution 2			
	Weight	**Testing Units**	**Average units/faults**
High	8	136	4.53
Medium	4	256	4.34
Low	1	552	4.06
Flat distribution			
Class (Risk)	**Weight**	**Testing Units**	**Average units/faults**
High	1	25	0.85
Medium	1	95	1.62
Low	1	823	6.05

V. Conclusion

We presented in this paper a large research work done in our Laboratory. The principal and innovative idea is the introduction of the new parameter (fault persistency), which is a function of the age of a file. The most important results we reached are the following.

The identification and evaluation of a new factor which can help to better understand the fault behavior with the age of a module: the persistency factor, widely discussed in section 3.

The new approach of evaluating fault prediction, considering the age of modules as well as their structural characteristics, leads to a more accurate classification andfault prediction than the traditional methodologies (for instance Table 2) .

As an example of real world application of fault prediction, we proposed a testing time partitioning methodology: if we apply a weighted partitioning of the modules on the basis of their membership to a given risk class, we can allocate testing time in an effective way, priorizing testing and so giving more testing time to high faulty modules and less to low faulty modules; the total testing time is available is usually fixed, but time is more correctly allocated and we obtained a more homogeneous sharing of testing resources.

The following steps of the research study concern the analysis of the full testing time distribution equation described at section 3.2 and not only a reduced version. In particular, we are now analyzing the influence of people related factors.

Further investigations concern the assessment of the model on new data sets coming from other application contexts, to find common characteristics and peculiarities of each application area; the analysis of fault proneness variation through a regression analysis over releases, to better describe fault density functions; the evaluation of efficiency of the method proposed for testing time allocation, by analyzing data collected from organizational processes that adopt this time allocation method

References

[1] N. Ohlsson, M. Helander, and C. Wohlin, Quality Improvement by Identification of Fault-Prone Modules using Software Design Metrics, in *Proceedings Sixth International Conference on Software Quality*, ICSQ 1996

[2] [K. Koga, Software reliability design method in Hitachi, in *Proceedings of the Third European Conference on Software Quality*, CSQ 1992

[3] M.H. Halstead, Elements of Software Science (Operating, and Programming Systems Series) Volume 7. New York, NY, Elsevier, 1977

[4] N.E. Fenton, S.L. Pfleeger, *Software Metrics*, PWS Publishing Company, Boston 1997

[5] McCabe T.J., A complexity measure, IEEE Transactions on Software Engineering, 4/1976

[6] J.C. Munson, T.M. Khoshgoftaar, Predicting software development error using software

complexity metrics, in *IEEE Trans. on Software Engineering*, 2/1990

[7] T.M. Khoshgoftaar, Tree based software quality estimation models for fault prediction, in *Proceedings of the 8th IEEEE Symposium on Software Metrics*, METRICS 2002

[8] L. Briand, S. Morasca, V. Basili, Property based software engineering measurement, in *IEEE Transactions on Software Engineering*, 1/1996

[9] Morasca S., Briand L.C., Towards A Theoretical Framework For Measuring Software Attributes, in *Proceedings of 4th International Software Metrics Symposium*, METRICS 1997

[10] G. Denaro, M. Pezzè, An empirical evaluation of fault proneness models, in *Proceeding of International Conference of Software Engineering*, ICSE 2002

[11] M. Pighin, R. Zamolo, A predictive metric based on discriminant statistical analysis, in *Proceedings of the 19th International Conference on Software Engineering*, ICSE 1997

[12] M. Pighin, P. Kokol, RPSM: A Risk-Predictive Structural Experimental Metric, in *Proceedings of the European Software Measurement Conference*, FESMA 1999

[13] B.A. Kitchenham, The certainty of uncertainty, in *Proceedings of the European Software Measurement Conference*, FESMA 1998

[14] P. Kokol, Long-Range correlations in computer programs, in *Cybernetics & Systems*, 1/1997, Taylor & Francis Publisher, 1997

[15] N.E. Fenton, P. Krause, M. Neil, Software Measurement: Uncertainty and Causal Modeling", in *IEEE Transactions on software engineering*, 4/2002

[16] M. Pighin, V. Podgorelec, P. Kokol, Program risk definition via Linear Programming Techniques, in *Proceedings of the 8th IEEEE Symposium on Software Metrics*, METRICS 2002

[17] A. Mahaweerawat, P. Sophatsahit, C. Lursinsap, P. Musilek, Fault Prediction in Object-Oriented Software Using Neural Network Techniques, in *Proceedings of the InTech Conference*, InTech 2004

[18] M. Pighin, V. Podgorelec, P. Kokol, The Operative Constraints of Software Reliability Prediction, in *Proceedings of Sistemics, Cybernetics and Informatics Conference*, SCI 2001

[19] Graves T.L., Karr A.F., and others, Predicting fault incidence using change history, in *IEEE Transactions on software engineering*, 7/2000

[20] T.J. Ostrand, E.J. Weyuker, R.M. Bell, Where the bugs are, in *Proceedings of International Symposium on Software Testing and Analysis*, ISSTA 2004

[21] M. Pighin, A. Marzona, Influence Of Structural Complexity On Fault Persistence, in *Proceedings of the International Association of Science and Technology for Development*, IASTED 2004

Authors' Profiles

Maurizio Pighin: Professor at the Department of Mathematics and Computer Science of the University of Udine; currently teaches advanced courses of Software Engineering and Information Systems. His major research interests are in the area of Software Engineering and ERP and Data Warehouse Systems. He is the author of more than 70 scientific publications in international journals, books and refereed conference proceedings. He worked at several national and international research and development projects. He is a referee of various international journals. He has been involved in the organization of some important events in the fields of Software Engineering and Information Systems.

Anna Marzona: Contract Professor at the Department of Mathematics and Computer Science of the University of Udine; currently teaches courses of Information Systems and Data Warehouse at the University of Udine. Her major research interests are in the area of Software Engineering and ERP Systems. She is a member of the Academic Spinoff LiberaMente Srl, a company that deals with design and implementation of computerized processes and data analysis systems for SMEs and the public sector.

How to cite this paper: Maurizio Pighin, Anna Marzona,"Fault Persistency and Fault Prediction in Optimization of Software Release", International Journal of Information Technology and Computer Science(IJITCS), vol.5, no.8, pp.15-23, 2013. DOI: 10.5815/ijitcs.2013.08.02