A software evolution experiment

Miguel Goulão, António Silva Monteiro, José Furtado Martins,
Fernando Brito e Abreu, Alberto Bigotte de Almeida, Pedro Sousa

Abstract

This paper describes an ongoing experiment where the evolution of a legacy system developed by the Portuguese Navy and its maintenance process are analysed. A framework used in the registering and analysis of software evolution requests, called RARE, is presented here. The problems related to data collection based on semiformal requests and to the "cultural adaptation" underway to gather more significant data, without a significant overhead, are addressed. A system's evolution is analysed with the usage of a set of statistically significant product metrics obtained by means of data reduction techniques. Process and product control and their role in the definition of predictive models based on software metrics are also discussed. Reliability analysis and statistical process control techniques were performed and used as basis for future process improvements.

1. Introduction

The necessity to build increasingly sophisticated applications is pressuring the organisations to develop efforts towards a more controlled production process. One of the great challenges faced by today's software industry is the software process improvement, as a means to achieve a higher quality in the developed products at a lower cost and within schedule [10]. Although there is no silver bullet to solve this problem, it is generally considered that the introduction of procedures to control the software process and assess the software quality is a step in the right direction. These procedures should be optimised to gather as much relevant data as possible in a non-intrusive way for the development teams. The purpose of collecting this data is the detection of eventual process flaws, pitfalls or improvement opportunities and the monitoring of the product's quality. An investment in the achievement of quality may lead to an effective cost reduction due to the decrease of costs associated with lack of quality [4].

DAMAG, the organisation responsible for analysing and developing management support methodologies within the Portuguese Navy, has decided to face this challenge and undertake a software process improvement (SPI) program, in co-operation with INESC, a Portuguese R&D non-profit organisation in the ITT field. A pilot-project, the Personnel Information Integrated System (SIIP), developed internally using COBOL (ANS and OS/VS V1R2M4) and SQL/DS in a proprietary system was chosen for starting that SPI initiative.

SIIP has been in operation since 1989 and presently is composed of around 650 modules. During this period, the system has been subject to both corrections and enhancements to accommodate the evolving needs of final users. As shown on Figure 1, the first two years of operation have witnessed a considerable decrease in the number of modules. This apparently odd behaviour derived from a lack of user involvement in the design phases, prior to the release of the first version. Instead of incremental delivery, the development team produced a system with a too wide range of functionalities that indeed were not used by the final users and were progressively excluded from operation, during that period.

In fact, the stabilisation of the system seemed to occur on 1992 and it has evolved relatively smoothly since then. Successive new specifications have been added to the system. On 1996, one of the system's branches was removed from SIIP and evolved as an independent subsystem.

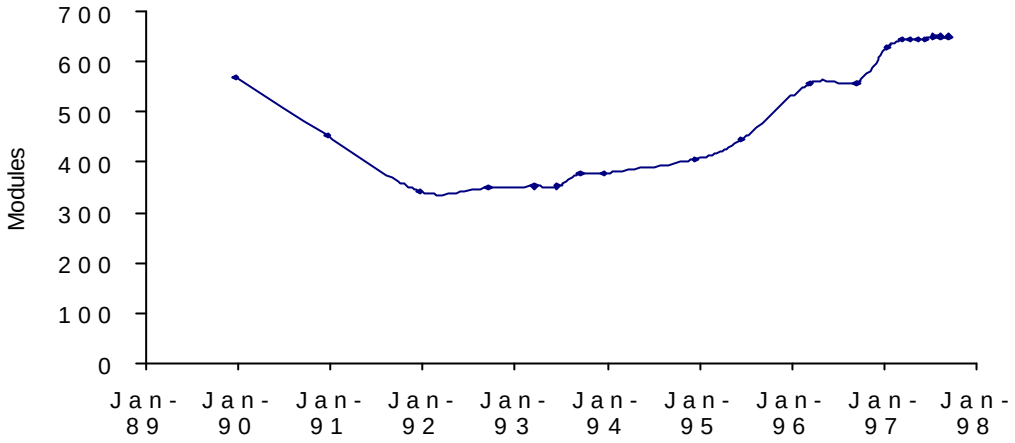

Figure 1 - Evolution of the Number of Modules

One of the aims of this initiative is to offer an orientation for the resources assignment in an eventual strategy of reengineering of the system [12], by identifying the most fault prone modules. An analysis on the SIIP modules based on software product metrics, including Halstead's Software Science metrics [3] and McCabe's control graph's metrics [7], amongst other ones, was carried out to fulfil this requirement.

Another important contribution for the organisation is the evaluation of the maintenance process and the corresponding report of recommendations towards its evolution. About 800 evolution requests collected since 1989 were re-coded into a suitable format for a statistical analysis. They allowed the identification of seasonal trends in the evolution of SIIP. Moreover, this analysis was an effective tool on the process evaluation issue.

This paper is organised as followed: Section 2 describes the methodological issues related to the data collection. A discussion will be made on the problems encountered during that process and on the followed approach to bypass them. Section 3 presents the collected process metrics as input towards the evaluation of the software process itself. In section 4, the adopted software product metrics will be discussed. Since the used tool allowed collecting a relatively large number of product metrics, that showed to contain redundant information, we will discuss the problems involved with data reduction without a significant information loss. A software complexity model based on the non-redundant data set will be presented and validated. In section 5, a summary of the most important lessons learned and conclusions derived from this experiment will be presented. The focus will be upon their usage in order to change the *status quo* in the organisation.

2. Data collection

The first step in this experiment was the recoverage of the information about SIIP evolution gathered in the realm of a semiformal evolution request reporting system in place since 1989. Several large paper folders containing eight years of software evolution requests and follow-up information were available. However, their content was highly unstructured. Thus, a normalisation and classification scheme of the evolution requests was developed with the support of the evolution team. It includes a list of items filled by the request author and by the evolution team. A tool named RARE (Register and Analysis of Requests of Evolution) was developed on Microsoft Access for registering the evolution requests classified according to this framework. A set of different measures about the software process evolution was registered on RARE. The RARE framework is presented in Appendix A.

The Logiscope [13] tool was used to collect a set of 42 product metrics, most of them on textual [3] and structural complexity [7]. Twenty versions of SIIP source code were

submitted to Logiscope and generated a sample of over 36000 available observations. Each observation corresponds to a routine.

3. Process evaluation

The data collected with the RARE tool was statistically analysed with SPSS [11] to provide input to the detection of potential targets of improvement.

3.1. Software evolution requests

The determination of the most frequent problems in a system is a valuable information to the organisation. For instance, a high percentage of problems related to incorrect database queries, could lead to the decision of conducting a training action of the support team in that specific matter Figure 2.a) shows the distribution of the requests, by failure type considering only the requests related to source code changes (the type codes may be consulted in Appendix A. Although some of the RARE problem classes were never used, this should not to be thought of as an indication of their unusefulness. The RARE aims to be a generic classification, so, it is natural that some of the classifications do not apply to all projects.

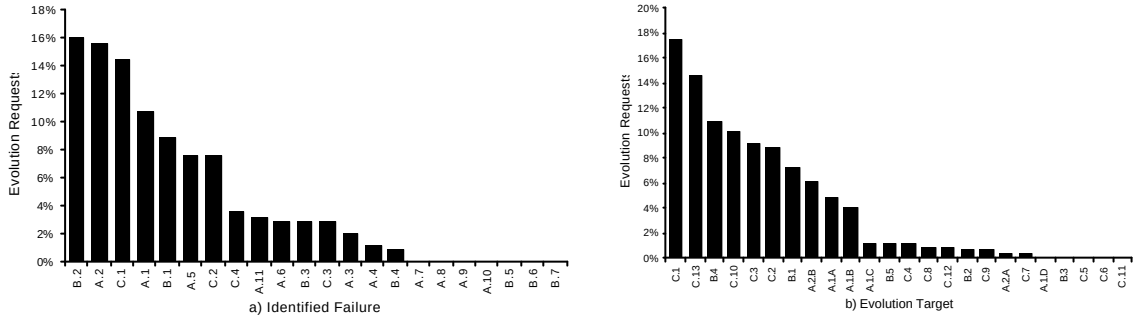

Figure 2 – Software evolution requests by a) Identified Failure and b) Evolution Target.

3.2. Evolution actions

The evolution actions are derived both from the knowledge of the maintained system and the evolution's specification. With the RARE framework, this specification is expressed selecting evolution types from classes of action's categories. Figure 2.b) shows their distribution. As for the problem description, the fact that some of the categories were never used within this sample again does not imply that they should be removed from the set, for generality sake as stated previously.

3.3. Expected effort for an evolution action

The RARE software evolution classification framework contains both a section on the evolution action and another on the corresponding effort. It is expected that the effort depends on the evolution classification.

As a null hypothesis, we state that the effort does not depend on the evolution action classification. The sample was segmented by evolution action and for each of the segments, the effort's mean and standard deviation were computed. We then made an ANOVA test to our sample, whose results were:

$$F\ Statistic\ (95, 609) = 39{,}91;\ Significance = 0{,}00;\ Eta^2 = 0{,}86$$

This shows that there is a strong probability that the effort does depend on the evolution action. The F Statistic value shows that the estimated variance between segments is considerably larger than the one within each segment. There is a very low probability that the

differences between the average effort for different actions are due to chance. The evolution action is responsible for 86% of the observed effort.

3.4. Medium Time to Repair

A key factor in user satisfaction is the Medium Time to Repair (MTTR). Whenever requesting a software evolution, a user expects it to be fulfilled within the smallest time gap possible. The MTTR measures the average time necessary to fulfil a request. Its value is expected to depend on several aspects, such as the maintenance team workload and the software evolution difficulty. While there is no available data on the team's workload to confirm the first hypothesis, it is possible to test the second one. As a null hypothesis, we state that the MTTR does not depend on the difficulty of the request.

Table 1- ANOVA Medium Time to Repair vs. Difficulty

Difficulty	Small	Moderate	Considerable	High
MTTR	21,15 [days]	49,9 [days]	55,25 [days]	78,2 [days]
(# of cases)	(670)	(40)	(20)	(20)

F Statistic (3, 749) = 6,84; Significance = 0,000

As expected, we have to reject the null hypothesis, thus accepting that the MTTR does depend on the evolution request difficulty, growing with it.

It is worth noticing that even though a request with a small difficulty is defined as one that may be solved in less than an hour, its MTTR is of about 21,15 days. This delay is mostly due to the bureaucracy that the registered changes have to go through, in order to be authorised.

3.5. Process Management

Managing a process depends highly on its stability. An unstable process varies in unpredictable ways, making the task of accurately forecasting its evolution impossible. One has to search for assignable causes for erratic behaviours. Assignable causes arise from the occurrence of events that are not part of the process, but that have an impact on the process performance being responsible for significant changes in its patterns of variation. These should be minimised or removed if their effect is hazardous to the process, or made them part of the process itself, if they bring benefits to it. When all these assignable causes are eliminated, the process becomes controllable.

Since the 1920's, control charts have been used with success in several industries to control their production processes. More recently these techniques are starting to be applied to Software Engineering [2]. In this experiment, we will use U and Z charts to control the average number of requests within versions weighted by their difficulty. The U chart rely on the fact that the analysed data follow a Poisson distribution (a skewed distribution appropriate and useful for phenomena that have a small probability of occurring on any particular trial, but for which an extremely large number of trials are available [11]). Instead of a simple request count, we considered the average monthly request count, due to the non-periodicity of the backups (versions). As the area of opportunity (Number of modules) has changed during the years of operation of this system, we will compute the request's density per K modules. We define as upper and lower control limits the values $+3\sigma$ (UCL) and -3σ (LCL). The process is said to be out of control whenever one of the following criteria is violated [14]:

a) A single point falls out of the 3σ control limits.
b) At least 2 out of 3 values fall on the same side of, and more than 2 σ units away from the central line.
c) At least 4 out of 5 successive values fall on the same side of, and more than one sigma

away from the central line.

d) At least 8 successive values fall of the same side of the centre line.

The values on Z charts are scaled in terms of σ (standard deviation) units.

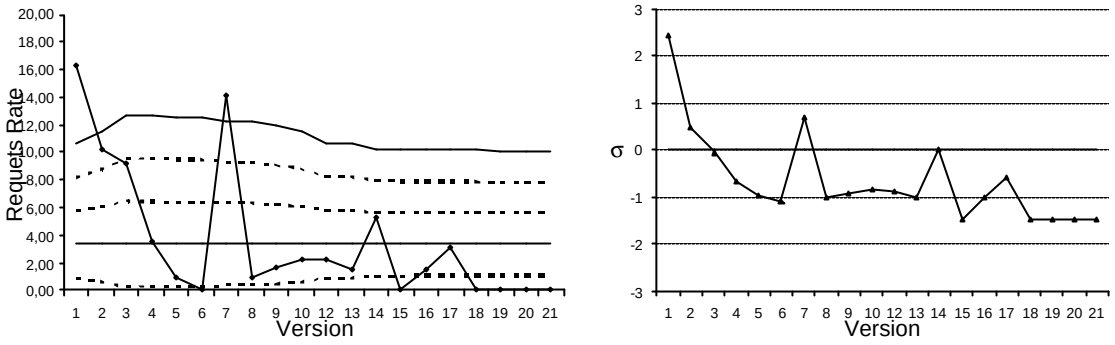

Figure 3 – U and Z charts for the requests density.

On the left side of Figure 3, we have the U chart requests density considering all the versions. Versions 1 and 7 of the system correspond to points of instability of the process, as do the latest 7 versions. The Z control chart, on the right side, reinforces the evidence of instability, stressed on its non-random pattern.

4. Towards a software complexity model

One of this experiment's goals is the definition of a complexity model based on the product metrics. The chosen metrics should reflect the main aspects of the software complexity, while providing a reasonably easy way to interpret the model.

Dealing with all of the collected metrics would be too much of a complex task. It is hard to keep track of a large number of metrics at the same time. When one analyses the available product metrics' definitions, it becomes apparent that in one way or another, most of them reflect the software size [15]. When defining, for instance, a multivariate estimation model where product metrics are used as explanatory variables [1], the validity of such an approach depends not only on the number of observations but also on the uncorrelation of those variables.

Principal Component Analysis is a technique to reduce the redundancy by minimising information loss. It allows multivariate data to be transformed into uncorrelated variables that may then be used to define a more robust linear model [9]. Moreover, this technique allows the analyst to choose only a subset of the new variables, as the remaining ones provide insignificant information on the data set. This technique has been used with success in the software engineering field [6], [8].

As expected, a correlation analysis confirmed the existence of redundancy in the collected metrics. The correlation matrix was then used to find out if there were any correlation patterns, in order to factoring out the underlying dimensions conveyed by all these metrics.

We defined as a threshold for the selected number of factors that they should account for at least 90% of the overall variance. We have then chosen only the first three factors that met this criterion (Table 2).

Table 2 - Complexity variance explained

Factor	% of Variance	Cumulative %
1	60,041	60,041
2	23,248	83,289
3	10,587	93,875

Table 3 - Rotated Component Matrix (using the Varimax rotation method).

Product Metric	Factor 1	Factor 2	Factor 3
Number of Edges	0,948	0,209	0,206
Number of Nodes	0,944	0,225	0,218
Essential Complexity	0,941	0,005	0,139
Number of Structured nodes	0,937	0,183	0,238
Number of Statements	0,925	0,253	0,187
Unconditional Jumps	0,922	-0,217	-0,011
Cyclomatic Complexity	0,913	0,298	0,197
Number of Sequence Nodes	0,909	0,307	0,231
Total of Operands	0,907	0,282	0,217
Total of Operators	0,876	0,311	0,135
Direct Calls	0,029	0,958	0,001
Maximum Levels	0,156	0,941	0,194
Exit Nodes	0,221	0,760	0,592
Entry Nodes	0,537	0,706	-0,099
Pending Nodes	0,363	0,105	0,912

Table 3 shows the Component Matrix, rotated with the Varimax method [5]. For each factor, some metrics will have a significant weight and all the others will have a weight as close to zero as possible. The rationale is to maximise the variation of each factor's weights thus facilitating their interpretation. Each metric will be used in the interpretation of a single factor, with this technique.

The first factor is both dominated by control flow metrics and by vocabulary complexity ones. All these metrics are somewhat related to the complexity within each component, either due to its inherent structure or to the number of symbols it manipulates. We will call this first factor *Intracomponent Complexity.*

The second factor is dominated by metrics somehow related to changes of context in the source code. These changes of context span from the number of nesting levels of the implementation, to the liaison of each procedural component with all the others, in other words, the relations between components. While following a program's execution systematically, these factors will affect its traceability. This second factor will be called *Intercomponent Complexity.*

The third factor is dominated by the number of terminal nodes in the control flow graph, and will therefore be referred here as *Pending Nodes*.

From an original set of 15 product metrics, we are now down to 3 factors of complexity, with a total loss of less than 7% of information. The construction of the complexity model with these 3 factors has to take into account that their typical values are of different magnitude spans. A relatively small variation of one domain would be able to hide a relatively high variation of the others. In order to avoid this effect, all the variables are standardised before being used to compute the domains. The values of the standardised values of the factors for each component were computed by regression.

Figure 4 presents the Pareto charts of the normalised factors, for each module of the system. These charts show the uneven distribution of the complexity throughout the modules that make up the SIIP. The percentage indicated for each of the modules is the ratio between that module's factor value and the respective highest factor value in the whole system. As the complexity increases with the growth of each of these factors, we are able to determine which modules are more error-prone, considering only one aspect of the complexity at a time. For instance, if we are concerned with intra-modular complexity, we will use the first factor as a selection criterion of the candidate modules for restructuring, to lower their harm potential.

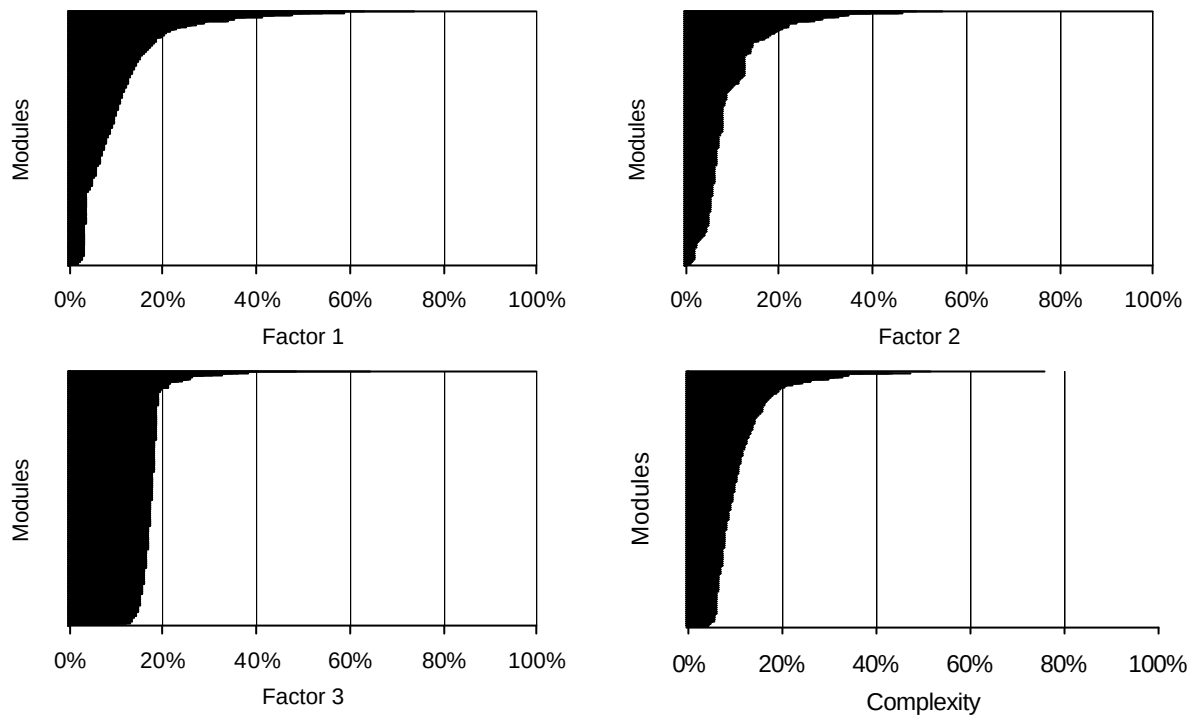

Figure 4 - Pareto charts for Factors 1, 2 and 3 and for overall complexity.

The next problem we have to cope with is how to combine these factors into a single complexity measure.

Each component's percentage of the explained complexity variance was used as a weight in the computation of the overall complexity. This enables us to sort the modules by their estimated complexity (Figure 4, bottom right).

To validate this model, there are at least two possible approaches. The first one would be to validate it using process data, as indirect measures of software complexity. As that was not possible with the available data, our second approach was to ask the most experienced member of the maintenance team to make a list of the most critical modules and of those other complex modules that don't require maintenance actions very often. The modules not belonging to either group were considered to be relatively simple, and constitute a third group. We removed from the sample those files that were either a) discontinued from the system, but were still part of the last analysed backup, or b) belonged to subsystems unknown to the consulted system expert. As a null hypothesis, we state that there is no relationship between the computed complexity and the expert's opinion.

Table 4 resumes the analysis of variance between the three groups of modules. Clearly, we reject the null hypothesis, and accept that the complexity value grows along with the expert's opinion classification. This complexity model may be used as a basis for comparing modules.

Table 4- ANOVA Computed Complexity vs. Expert's Opinion

Simpler Modules	Complex Modules	Critically Complex Modules
0,090	0,195	0,274
(501)	(18)	(31)

F Statistic (2, 547) =344,985; Significance = 0,000

5. Conclusions and further work

The process data analysis revealed an uncontrolled software maintenance process. In other words, although we know the system has been evolving during all the analysed period, a significant (however impossible to quantify) proportion of the evolution actions were not registered. This realisation has two sorts of consequences:

Any attempt to identify cause-effect relationships between the product and the process data was jeopardised by the absence of essential process data. The development of reliable effort estimation models is one of the priorities of this project. This leads to the second and perhaps most important consequence of the first phase of this project: the presentation of evidence of the problems raised by the semi-formal evolution request policy was an essential step towards the objective of process improvement.

This experiment motivated the organisation to adopt the RARE as a framework for registering the evolution actions, not only in the analysed system, but also in 3 other systems that are currently being developed. The RARE application is being enhanced to accommodate not only the data used in the computation of process metrics, but also valuable information to the project managers. The rationale is to embed in the same environment the ability to track changes while recording the data for the process metrics. This enhancement has the advantage of motivating the development teams in the usage of the RARE. The next step will be to encourage the evolution requests submission via a RARE version available on the Navy's Intranet. These actions, along with "marketing actions" with the involved personnel are believed to enable a controlled process data collection.

In turn, this will enable us to identify cause-effect relations between product and process data and therefore to develop effort estimation models. We plan to use autoregression models for this, to accommodate the intrinsic seasonality in process data. Autoregressive models are useful in time series, since successive values are correlated with each other and that makes the use of the ordinary least squared regression inadequate.

In the near future, all these activities will result in a better-controlled development environment at DAMAG. This will allow a more effective resource distribution to face the never ending challenge of final user satisfaction with the maintained systems and provide guidance in the definition of reengineering policies.

6. References

[1] Abreu, F.B., Melo, W., "Evaluating the Impact of Object Oriented Design on Software Quality", Proceedings of the 3rd International Software Metrics Symposium (METRICS'96), IEEE, 1996.

[2] Florac, W.A., Park, R.E., Carleton, A.D.,"Pratical Software Measurement: Measuring for Process Management and Improvement", CMU/SEI-97-HB-003, 1997.

[3] Halstead, M., "Elements of Software Science", Elsevier North-Holland, New York 1977.

[4] Houston, D., Keats, J.D., "Cost of Software Quality: A Means of Promoting Software Process Improvement", 1996.

[5] Kaiser, H.F., "The Varimax Criterion for Analytic Rotation in Factor Analysis", Pshycometrica, 1958.

[6] Khoshgoftaar, T.M., Allen, E.B., Kalaichelvan, K.S., Goel, N., "The Impact of Software Evolution and Reuse on Software Quality", Empirical Software Engineering, 1, Kluwer Academic Publishers, Boston, 1996, pp. 31-44.

[7] McCabe, T., "A Complexity Measure", IEEE Transactions on Software Engineering, Vol. 2, Nº4 pp 308-320, 1976.

[8] Pighin, M., Zamolo, R., "A Predictive Metric Based on Discriminant Statistical Analysis", Proceedings of the ICSE 97, 1997, pp. 262-270.

[9] Reis, Elisabeth, "Análise factorial das componentes principais: um método de reduzir sem perder informação", Giesta ISCTE, 2ª Ed., 1993.

[10] Roberts, M.A., “Experiences in Analizing Software Inspection Data”, Proceedings of the Software Engineering Process Group Conference, 1996.
[11] SPSS, “User’s Guide”, Technical and Promotional Documents Relating to the SPSS V7.5 Tool, 1997.
[12] Schwald, A., “Metrics, People and Their Roles in a Software Project”, Metrics News, 1997.
[13] Verilog, “Logiscope Viewer – Basic Concepts”, Technical Documents Relating to the Logiscope Tool. Verilog SA., 150 rue Nicolas Vauquelin, BP 1310, 31106, TOULOUSE cedex, France, 1993.
[14] Western Electric Co., Inc., “Statistical Quality Control Handbook”, Indianapolis, Indiana: AT&T Technologies, 1958.
[15] Wiper, M., Göbbels, M., Brunenberg, L., “Relations between software metrics”, Proceedings of Eurometrics ’92, Brussells, 1991, pp. 91-99.

Appendix A

A REQUEST FRAMEWORK FOR SOFWARE EVOLUTION

USER ENVIRONMENT

SUBMIT DATE: *When was the Problem reported?* ____ - ____ - _____

SUBMITTER: *Who reported the Problem?* _____

PRIORITY: *How critical is the Problem?*

Low (1) ☐
Medium (2) ☐
Serious (4) ☐
Critical (8) ☐

FAULT TYPE: *What is the nature of the Problem?*

A. FORMS

1. Non-existent information ☐
2. Presentation of the information ☐
3. Incorrect presented values ☐
4. Redundancy or incoherence of presented values ☐
5. Constraints on data values are too restrictive ☐
6. Constraints on data values are too loose ☐
7. Insufficient Help context ☐
8. Non conformity with interface standards ☐
9. Interface errors in a terminal or specific emulator ☐
10. Excessive response delay ☐
11. Transaction error ☐

B. OUTPUT'S

1. Non-existent information ☐
2. Presentation of the information ☐
3. Incorrect presented values ☐
4. Redundancy or incoherence of presented values ☐
5. Non conformity with interface standards ☐
6. Interface errors in a specific printer ☐
7. Excessive time of answer ☐

C. NAVIGATION AND SECURITY

1. Definition of user and group profiles ☐
2. Definition of auditing facilities ☐
3. Definition of backup and recovery facilities ☐
4. Organisation of navigation ☐

D. DATA MAINTENANCE

1. Data change ☐
2. Data access ☐

PROBABLE PROBLEM CAUSER: *Who is the possible Problem cause?*

Development Team ☐
Users ☐
Others ☐

DEVELOPMENT ENVIRONMENT

DIFICULTY: *How difficult is to fulfil the request?*

Low (1) ☐
Moderate (2) ☐
Considerable (4) ☐
High (8) ☐

EVOLUTION ACTION: *What actions were performed to fulfil the request?*

A. DATABASE
1. Relational design DDL (Logical Design)
   a. Definition of new structures ☐
   b. Change of existent structures ☐
   c. Incorrect or non defined uniqueness integrity constraints ☐
   d. Incorrect or non defined referential integrity constraints ☐

2. Data manipulation (DML)
   a. Incorrect data type conversion ☐
   b. Incorrect processing ☐

3. Data
   a. Automatic data backup ☐
   b. Manual data backup ☐
   c. Manual data entry ☐
   d. Data access ☐

B. FORMS
1. Change of fields specification ☐
2. Change of initialisation fields values ☐
3. Incorrect type conversion ☐
4. Change of data entrance validation ☐
5. Change in accordance with form design specifications ☐

C. CODE
1. Validation on data entry ☐
2. New data structure ☐
3. Change existent data structure ☐
4. Incorrect environment variables manipulation ☐
5. Change of initialisation data structures values ☐
6. Incorrect passage of parameters ☐
7. Incorrect treatment of error and exceptions ☐
8. Incorrect manipulation of operating system files ☐
9. Incorrect flux control ☐
10. Change of algorithm or logic processing ☐
11. Change in accordance with coding rules ☐
12. Code documentation ☐
13. Code restructuring ☐

FOLLOW-UP STAGE: *What was the final out come?*

Request Satisfied ☐

Request Not Satisfied ☐

1. False alarm ☐
2. Request already satisfied ☐
3. Duplicated request ☐
4. Available resources were not sufficient for fulfil the request ☐
5. Resolution differed *sine-dia* ☐
6. User did not supply additional information asked for the fulfilment of the request ☐

CLOSING DATE: *When was the Problem resolved?* ____ - ____ - _____

EFFORT: *How costly was it to fulfil the request?* _____ Person-Hour

EVOLUTION ACTION RESPONSABITITY: *Who's the responsible for the actions?* _____