Evaluation for Software Testing: Project EXOTEST

Michel Cheimanoff
Statistical Techniques Evaluation for Software Testing: EXOTEST

ESSI Project 24266

EuroSTAR ’99, Barcelona, November 8-12

Michel Cheimanoff
Email: Michel.Cheimanoff@detesis.thomson-csf.com
Thomson-CSF Detexis profile

◆ Airborne systems
◆ Radars
◆ Electronic warfare systems
◆ Information technology and systems
◆ Missile electronics
European System and Software Initiative (ESSI)

- Promote improvements in the software development process in industry
- Through the adoption of established methods and technologies
EXOTEST Objectives

(Experiment Of New Testing Strategies)

◆ To evaluate the potential contribution using statistical techniques to improve the software process and to quantify the reliability achieved

Subcontractor: Mathix (F. Vallée)
Major conclusions

Interesting results about:

- Expected gains when implementing statistical techniques
- Impacts on the organisation and the development process
- Contribution to the process improvement
Business motivation

◆ Better control of the testing process
◆ Reduction of development lead-times and costs
◆ Reinforcement of the reliability image of the company products
The experiment
Evaluation of 3 different techniques
Exotest project vs Pilot project

◆ Pilot project is real project under development:
  – embedded software for avionics
  – C library of real time services
  – strong reliability requirements

◆ Two independent teams in parallel:
  – Pilot team in charge of the application's development
  – Exotest team working in parallel (testing phases only)
**Code quality measurement technique**

**Objective:**
- early detection of high risk modules for a better distribution of the test effort

**Principles:**
- elaboration of a predictive risk model based on the code characteristics using a similar project (reference project) on which the modules quality is known
- application of the obtained model on the Pilot application
- unit testing emphasised under the risk detected modules
Code quality measurement: modelling

- **Reference project:**
  - 88 modules for modelling
  - 97 modules for validating

- **87 potential structural metrics examined**

- **Tool: M-square©**

- **Obtained model:**
  - Reduction to 16 significant metrics
  - Completion rate: 75%
  - Correctness rate: 92%
Code quality measurement: results

- Well-balanced distribution of high risk detected procedures: 54%
- Due to a change in the pilot project testing strategy (using expert knowledge), no gain in productivity
- Good distribution of testing effort whoever performs the unit testing
Objective:
- during functional testing, no systematic elimination of defects, but achievement of the predefined operation reliability level

Principles:
- elaboration of an executable model of the software
- random generation of test cases covering the operational profile
- execution of test until reliability achievement

2 statistical techniques
Statistical testing : modelling

- **Obtained model:**
  - 148 states
  - 343 elementary data
- **Tool : Statemate©**
- **Operational profile:**
  - distribution of the 14 software services into 3 classes of frequency of use
- **Generated test cases:**
  - 12 scenarios
  - 1 scenario = 240 calls
Gain of productivity

- Testing effort gain: 20%
- Good detection of dynamic defects

Defects detection by the Exotest team: 4
Defects detection by both teams: 3
Defects detection by the pilot team: 6

Dynamic defects

Statistical defects
Better control of the testing process (stopping criterion)

Mixed testing strategy could be the most efficient
Software reliability modelling technique

◆ **Objective:**
  - be able to forecast correctly software operational reliability

◆ **Principles:**
  - recording of the initial failures in an environment operational like
  - use of the software reliability modelling
  - decision making knowing:
    • the number of failures to appear during a given operational time
    • the effort and time necessary to obtain the objective reliability
Software reliability modelling

- **Data**: failures obtained during the Exotest functional testing
  - 2,880 calls
  - 7 failures

- **Examined models**:  
  - Musa-Okumoto
  - Goel-Okumoto
  - Crow

- **Tool**: M-elopee©
Software reliability modelling: results

◆ **Use as stopping criterion**

Probability (48% with Musa-okumoto model) of having a new failure, during a new functional testing session of 2 880 calls, to low to continue with respect to the test effort

◆ **Good estimation of future behaviour**

- Predicted number of failures for the next 4 months (around $180 \times 10^6$ calls): 
  $[9,14]$ failures with a 80% confidence interval (Musa-Okumoto model prediction)
- Real number of failures appeared during these 4 months: 10
Lessons learnt (1)

◆ This experimentation is directly correlated to the CMM (level 3 and 4) activities (Measurement database, System test, Acceptance test, Measures and analysis, Statistical control)

◆ Culture changes needed into the testing approach:
  – progressive shift from a uniform and exhaustive test strategy to a reliability oriented strategy
Lessons learnt (2)

◆ Costs of implementation:
  – high costs
  – necessity to have an expert group

◆ Difficulty of synchronisation when working on a real project
Conclusions

- most objectives attained
- improvement of practices within the company and into the European industry
- help for the demonstration of a better process control
STATISTICAL TECHNIQUES EVALUATION FOR SOFTWARE TESTING:

EXOTEST (EXperiment Of new TEsting Strategies) - ESSI Project 24266

Michel Cheimanoff
Thomson-CSF Detexis
5, av. Jean d’Alembert
78190 Elancourt Trappes – FRANCE
Tel. +33 1 34 81 73 34 – Fax +33 1 34 81 65 37
Email : Michel.Cheimanoff@detexis.thomson-csf.com

Abstract:

This paper presents an ongoing ESSI Process Improvement Experiment, carried out within Thomson-CSF Detexis, which aims to evaluate the potential contribution brought by statistical techniques to first, improve the testing and maintenance processes and then, to quantify the reliability level achieved.

Confronted with the challenge of maintaining or even increasing the reliability level of the ever more complex systems it produces, Thomson-CSF Detexis has to adapt its testing and validation strategy through innovative and yet proven methods. Combining the following techniques lead us to an experimentation covering each and every successive aspect of the software development cycle, from the unit tests to the acceptance test:

- code quality measurement: the objective is to detect those modules that are the most likely to demonstrate low reliability. This detection is based upon a source quality model using the OSR (Optimal Set Reduction) algorithm based on both statistical and machine learning principles. Once the most sensitive modules have been detected, they are submitted to intensive testing or review to enhance their quality level. In order to maintain identical global effort, the test plan alleviates test cases concerning the others modules.

- statistical testing: in the experiment we have decided to use functional statistical testing before acceptance. The objective is to cover the software by following an operational profile and thus, to insure a predefined level of reliability in operation. The test set has been generated by modelling the functional behaviour of the system, execute the model and control the data entry until the operational profile is obtained.

- software reliability modelling: as the ultimate objective of the experiment is to reduce testing and maintenance costs and to keep at least the same reliability level in operation, it is important to be able to quantify reliability. Therefore, software reliability modelling will be used both during development, to monitor the process, and during operation to verify that reliability objectives are effectively obtained.

The better control of the testing process we expect to draw from the adoption of these methods shall, in the future, optimise our development effort by reducing our development lead-times and costs and in a more general way help us enhance the image of our products in our customers' sight.

The objective quantification of the software reliability is likely to improve our control over the risk management and so, eventually reduce our maintenance costs.
1. Executive Summary

1.1. The EXOTEST project

The Exotest project is an ESSI project, carried out within Thomson-CSF Detexis (result of the merging of Dassault Electronique, Thomson-CSF RCM and Thomson-CSF TME - on January 1, 1999), a French company specialised in the design, development and manufacture of sophisticated systems, Software and Information systems in the fields of Aeronautics, Telecommunications, Space and Information Technology.

In relationship with a wider internal plan for improvement of software practices against the SEI CMM model, Thomson-CSF Detexis, within the scope of this project, has experimented new testing techniques based on the use of statistical tools and statistical functional testing practices to mutually enhance their benefits through Software Process Improvement.

The project's aim consisted in estimating, using a project under development (called pilot project), the potential contribution brought by statistical techniques to, first improve the testing processes (unit test phase and functional tests) and then, quantify the reliability level achieved. The pilot project used as a support to the experimentation is carried out at Thomson-CSF Detexis and concerns the development of an embedded basic software.

Thomson-CSF Detexis has chosen as subcontractor, Mathix company to integrate statistical methods in order to improve software testing and evaluation processes with regard to business goals (such as development time, cost effectiveness) and quality metrics (reliability, level of confidence in software produced). Mathix is a French independent consulting firm having a strong expertise in the field of mathematically-based technologies for reliability and safety engineering and which provides assistance, training and statistical tools to bring its expertise in these domains. Mathix also provided us with its M-Square and M-Elopée® tools.

1.2. Major conclusions

This project, coming to an end, has allowed to experiment new innovative testing techniques within real conditions.

The obtained results are satisfactory with respect to the project objectives. They allow to get interesting conclusions, especially about:

- the expected gains when implementing those techniques (productivity, reliability),
- the impacts on the organisation and the software development process,
- the contribution to process improvement in the frame of SEI/CMM maturity model.

1.3. Business motivation

Quantitative measures of the improvement brought by these methods with respect to those commonly used within the group are likely to allow us to concretely appraise their adequacy to our scope of activities.
In a more general way, the better control of the testing process we expect to draw from the adoption of these new methods shall, in the future, help us enhance the image of our products in our customers’ sight.

Optimising our development effort will also help us reduce our development lead-times and costs and therefore reinforce our status against our competitors.

The objective quantification of our software reliability is likely to improve our control over the risk management and so, eventually reduce our maintenance costs and the operating costs of our customers.

Besides, this quantification will certainly play a great part in the motivation of the technical teams, by offering them measurable targets and the direct feedback of their work results.

2. The experiment

2.1. Practical implementation

The project has consisted in setting up an independent team (called Exotest team) who carried out, with the help of new techniques based on a statistical approach, a test campaign using a pilot application under development and without interfering with the development process retained for the selected pilot application.

The results obtained by the Exotest team and the pilot team (the one in charge of the pilot application development) are compared in terms of reliability and quality.

Due to its nature, this study has imposed the selection of a pilot project meeting certain conditions: testing process just starting or in the progress of being started and having a development process compatible with Exotest project: same test stages retained (unit testing, functional testing, …), information build-up and duration of the project.

The pilot project that has been selected is a subset of an embedded software (written in C language and provided as a library of services).

Combining several techniques has lead us to an experimentation covering each and every successive aspect of the development cycle, from the unit tests to the acceptance test. These techniques were the following:

- code quality measurement,
- statistical testing,
- software reliability modelling.

These techniques and the results that they have provided are described in the next paragraphs.

2.2. Code quality measurement

Experiment

Code quality measurement aims to determine, during the unit test sequence, the test workload for each procedure included in the pilot application as a function of the risk identified and
corresponding to the procedure. It uses basically another project, called reference project, for which the source code risks and characteristics are already known (explanatory metrics) in order to statistically build a predictive model of such risks based on the characteristics and to apply this model subsequently to similar projects.

This approach has been developed and validated by Mathix with the NASA Software Engineering Laboratory (SEL) [MSQ]. It is based on the following steps:

- definition of a test strategy (type of unit tests to be carried out as a function of the risk identified by the model),
- identification of a reference project, having the same characteristics as the pilot project (programming language, application type, project size, etc.) and having experiment feedback information in sufficient quantity to initialise the quality measurement model of the source code (analysis of the software defect sheets),
- drawing up of a list of potentially explanatory metrics of the quality criteria,
- construction of a predictive model of the risks using M-Square tool from Mathix company. This tool, starting from the OSR algorithm (Optimal Set Reduction), is based on statistical principles and learning techniques (statistical detection of patterns identifying the risk and built with explanatory metrics). The data from the reference project have been separated into a reference sample used for building the model and a test sample used to evaluate the model quality using the data not used to build it. Several modelling processes were carried out in order to optimise the model in terms of number and pertinence of the metrics used and in terms of size of the modelling tree,
- extraction of the metrics selected (and retained in the final model) on the pilot application,
- application of the model to the pilot application source code and classification of the procedures as a function of the identified risk for each procedure (determination of the test workload to be carried out as a function of the predicted risk),
- execution of the unit test campaign using these procedures and applying the test strategy previously defined,
- recording of the results obtained from the two test campaigns carried out by both teams for analysis.

Main results

In the Exotest project, the chosen reference project size was 690 files and 2 146 procedures. Among them, 154 procedures have been detected of poor quality, i.e. they were certainly at the origin of one or more failures during the software operation.

A list of potentially explanatory metrics of the quality criteria was first established: 32 design-level metrics, i.e. metrics assessed at file level, 55 coding-level metrics, i.e. metrics assessed directly at procedure level. This list was a trade-off between the metrics that were initially selected and the metrics that could effectively be measured.

The characterisation of the risk on the reference project was based on feedback information from the reference software defect sheets. In many cases, this information did not offer a sufficient granularity level (it was known that a file contained a defect, but the faulty procedure was unknown). However, it has been possible to build a modelling subset (used to
build the predictive model of the risks) consisting of 88 procedures and a test subset consisting of 97 procedures. Each one of these subsets contained procedures that had undoubtedly contained one (or more) defect(s), and procedures that had not contained any defects, in equal numbers.

Correlation analyses and various modelling processes were carried out successively (independent modelling processes using the design metrics only, using coding metrics only, and then combined modelling). This work has made it possible to highlight the most interesting metrics and to build an optimised model in terms of number and pertinence of the metrics used (reduction from 87 to 16 of the number of these explanatory metrics) and of the size of the modelling tree.

The metrics that were assessed at file level proved to be the metrics that most carried the desired reliability information. For example: number of array variables, total number of variables, number of imported functions and total number of function in the file.

The modelling that was performed on the reference sample gave satisfactory results. The resulting model makes it possible, on the reference sample, to detect 80% of risky procedures (i.e.: procedures that contain at least one defect). The initial objective that consisted in reducing the number of metrics was largely fulfilled.

The test strategy that was defined at the beginning of the experimentation provided for the type of unit tests to be performed on each procedure as a function of the (low, medium, high) risk level on the procedure: code review, black box tests, glass box tests. This implied a homogeneous distribution of the procedures per risk level. The application of the model on the pilot application gave satisfactory results, which allowed the predefined test strategy to be effectively applied: detection of 32% of low-risk procedures, 14% of medium-risk procedures, and 54% of high-risk procedures.

On the contrary of what was expected the experimentation did not demonstrate any gain in productivity. The number of defects detected by the pilot team and by the Exotest team were quite similar. In that sense the experimentation did not reveal a difference of quality efficiency between the two techniques used. Moreover both teams did consume the same effort.

It must be noticed here that the pilot team did not apply a uniform testing strategy as usual but adapted its effort in function of its expertise on each component complexity. If they had applied the usual techniques, they certainly would have spent much more time to get a similar level of detection.

However using models to detect the most complex code parts and adapt the testing effort accordingly will generally help to formalise an approach which so far was part of the non-formalised expertise of the tester. This also contributes to the information building effort prescribed by the different maturity models. Also, the possibility of re-using an OSR model under other projects of similar type will favour the reproducibility of the approach.

2.3. Statistical testing

Experiment

Statistical testing as experimented in this project is geared towards a test objective that is not based on a systematic elimination of defects, but on the achievement of a predefined operating reliability level. The test strategy must consist in performing the tests in a context...
as close as possible to the operational use environment, in order to have a perfect knowledge of the software reliability behaviour in such environment and to be capable of controlling it.

To implement statistical testing, Mathix's approach [STU] consists in modelling the software in order to specify and generate (random generation) scenarios of pertinent and realistic test sets (i.e. matching the operational use profile). The technique requires to have a pilot project specification/design documentation of sufficient value to make software modelling possible by a team who had not participated in the drawing up of said documentation.

This has implied in the Exotest experiment to execute a first phase with the following steps:

- modelling of the pilot project software: the behaviour of each function was modelled from the software specifications with the Statemate tool from I-Logix Inc. [STA]. The model thus obtained, comprising a hierarchical set of diagrams, can be executed in a factual and temporal way. The model was checked by simulation and manual, exhaustive validation of the behaviour of the different services and resulting effects,

- modelling of the software environment, i.e. definition and modelling of a typical application that is representative of the use of the software in terms of invoked services and of the frequencies of invocation of such services,

- execution of these models (software and environment) and random generation of test scenarios that converge as much as possible towards the expected profile. Based on a random simulation of software function invocations, the software prompting environment was reproduced, taking into account the logic of each function. The test scenario generation process was carried out through successive refinements with the aim of getting as close as possible to the expected profile,

- effective test specification through successive refinements (using the trace of execution information provided with the Statemate Prototype function).

The second phase consisted in creating and setting up the functional tests previously specified:

- development of a test application capable of implementing the test scenario sets specified during the previous step,

- execution of the test run (the same test environment is used by both the Pilot and the Exotest teams).

- comparison with the results obtained by the pilot team.

**Main results**

It has been possible to model the behaviour of each one of the software services. Since the modelling was not performed by the team that had specified the software, the specification/design stage formalising level proved to be insufficient to build the model, especially in terms of the data structures. Additional information from the pilot team was required.
The expected operational statistical profile of the frequencies of invocation of the various software services was used as the input for the model. This operating profile was defined according to statistical aspects (software service distribution profiles) which are generally hardly formalised in the field of onboard software (focus is more on the interfaces and the performance characteristics of the individual functions).

As the aim of the study was to specify test scenarios, rather than making a statistical validation of the information flows, the model was only made up of statecharts (deterministic model). Owing to the dependency between the functions, it was difficult to obtain the exact operational profile. But the obtained result seemed to be sufficiently similar to be considered satisfactory.

The size of the final model is rather big: 14 services modelled, 34 charts, 148 states, 343 elementary data.

An example of chart modelling the behaviour of the service "SEND" is given in the following figure:

The "SEND" service verifies that preliminary conditions to the sending of a message are verified, then it sends the message and does the associated updates.

The test application building process implementing the whole of the test scenarios obtained by execution of the model was automated, in order to allow for the high number of tests to be performed (12 different scenarios, each one including 240 invocations of services of the pilot application).

All of the detected failures did appear at the beginning of the test (i.e. before the 500 first calls). Using software reliability modelling (see the following paragraph) allowed us to
decide to stop statistical testing because the probability to get new occurrences of failures was to low with respect to the effort necessary to pursue the testing.

As it was expected the experimentation did demonstrate a productivity gain:

- the Exotest experimentation did reveal 7 defects among them 4 were not detected by the pilot team,
- on the other hand the pilot team did detect 10 defects among them 1 should have been detected by the Exotest experiment, the 6 others were out of the scope of the model (initialisation, limit cases out of the operational profile),
- the Exotest team did consume 20% less effort than the pilot team.

As a learnt lesson it appeared that a mixed strategy combining statistical testing and deterministic testing (for the limit testing) could lead to the best efficiency.

2.4. Software reliability modelling

Experiment

Software reliability modelling's purpose was to define and set up a pilot application reliability measurement and prediction campaign.

The main steps for the reliability modelling were the following:

- specify a relevant metric to measure execution time of the pilot application,
- record (using the M-élopée tool from Mathix [MLP]) and sort the failure which have occurred during the validation phase,
- select the reliability model that is the most representative of the application's behaviour,
- predict the application’s evolution.

Main results

In the case of the experimentation, the selected relevant metric for this type of application was the number of calls activated through the statistical testing. During the experimentation 12 scenarios of 240 calls each have been executed: this leads to a total number of calls of 2880.

The number of defects revealed by the test scenarios (7) was far too small to perform the model selection process. So it has been decided to choose the most credible model to do reliability prediction.

The following figure gives the modelling of the cumulative number of failures for the 3 pre-selected models:
The use of software reliability modelling has allowed in the Exotest project to answer to two concerns.

The first concern was to decide whether to stop or not (statistical) functional testing with respect to its predicted efficiency. In our case all off the model predicted a weak probability (less than 50% in average) of having one new failure in the following 12 scenarios. Considering the effort needed to do this complementary testing, it has been decided to stop the statistical testing after this experiment of 2880 calls.

The second concern was to predict the number of failures to be experienced in the next period when the software would be used by other applications. A rough evaluation of the number of calls to be experimented during the next 4 months was of $180.10^6$ calls. The most credible model that was the Musa-Okumoto model gave the 80% confidence interval [9,14].

In reality, 10 failures have been experienced during the next 4 months what was consistent with the prediction.

The experimentation did attain its objectives showing that this technique could be used as relevant stopping criteria for the functional testing and to predict operational reliability. However because the sample of failures experienced by the project was very small, further verification is needed before implementation.

### 3. Lessons learnt

Among the numerous learning from this experimentation, it is interesting to highlight the following points:

**A partial answer to the requirements of SEI/CMM maturity model**

These techniques are best suited for projects that are carried out by organisations with a certain maturity level. As a matter of fact the assessed techniques have a positive, direct or indirect effect on the formalisation, capitalisation and reproducibility aspects of the development process improvement undertaken in Detexis.
In particular, the adoption of these methods offers a partial answer to the requirements of the CMM/SEI model. They can be correlated to the model's activities (level 3 and 4) such as Measurement database, System test, Measures and analysis or Statistical control.

**Need to adapt our testing approach**

There has not been any real problem in the application of these techniques to the Exotest experimentation. However proper implementation of the Statistical testing and Software reliability modelling techniques requires that we significantly change our approach to the test and the test purposes. We should shift progressively from a uniform and exhaustive test strategy (detect as many defects as possible in a uniform way in all the code) to a reliability enhancement strategy (detect the defects where they are the most disturbing, taking the risk of leaving some others undetected in other places).

**Costs of implementation**

These modelling techniques are costly and difficult to implement (modelling and statistical tools reserved for use by specialists and generally not available). In fact, project leaders and developers working on 'conventional' software development processes (and particularly in the test and validation phases) are not supposed to have this knowledge.

But it should be noted that:

- The modelling associated with the code quality measurement technique is only to be performed once for each given class of project (i.e.: projects having the same characteristics). Return on investment should thus be substantial if these types of projects do not change too rapidly and if these classes are truly representative (in terms of number of projects belonging to the classes).

- On the other hand, the modelling associated with the statistical test technique must be performed for each project. This additional specification/design effort should be offset, however, by the fact that the requirements will be better managed and that the development costs (in the test/validation phase) and the maintenance costs will be reduced.

- Finally, an expert group should be set up to work on the various modelling tools.

**About the experiment**

Working on a real pilot project was definitely the weak point in the experimentation, as synchronisation was required between the pilot and the Exotest teams. With strong industrial constraints, the pilot project that was retained initially has undergone significant changes - in terms of scheduling and development process - which prevent the study to be carried out properly within the allocated lead-time.

Additionally, the frequent exchanges of information that were needed to re-synchronise the two teams could have affected the results. Since the experimentation program did not provide for the correction, by the Exotest team, of the pilot application source code when an error was detected, the context of the error had to be communicated to the pilot project, for it to make
the code change required. We have tried to take into account the most objectively possible this problem when making the results analysis.

4. **Conclusions**

The experimentation has been conducted in a satisfactory way, as the objectives have been attained.

Our ambition is that our project could offer a convincing demonstration of the application of statistical software reliability testing and checking methods likely to improve the ruling practices within our company and in the European industry.

These techniques could be applied, to help to demonstrate to the outside that the software development processes are under full control, allowing the quality of the products to be improved.

5. **Acknowledgement**

The experiment described in this paper has been funded by the EEC under ESPRIT project 24266 : EXOTEST, Experiment Of new Testing Strategies.

6. **References**

[MLP] M-élopée: a case tool for software reliability study
F. Vallée, B. Gayet, H. Derriennic and G. Le Gall
ISSRE Proceedings, 1995

[MSQ] M-SQUARE : a tool for software-risk analysis and management
B. Gayet and L. Briand
RAMS Proceedings, 1994

[STA] Statecharts: A visual formalism for complex systems
D. Harel
Sciences of computer programming 8, 1987

[STU] Study of the test contribution to software reliability quantitative assessment for nuclear-safety critical systems
F. Vallée, J.Y. Henry and F. Daumas
ISSRE Proceedings, 1998
Michel Cheimanoff, IT Engineer having graduated from ENSEEIHT in Toulouse (France), has over 16 years experience in software development within Thomson-CSF Detexis (previously Dassault Electronique). He has been involved during 7 years in Software Testing and has been both project and technical manager in charge of the design and the development of testing tools, such as DEVISOR (software testing and debugging tool, marketed since 1986). He works now in project management roles, mainly in the embedded real-time systems applications.