

## Experimentation Using the Goal/Question/ Metric Paradigm: Studies on Preventing and Detecting Defects

When using the GQM for experimentation and analysis, we need to plan the data analysis techniques appropriate for the problem and data at hand. The first three steps of the Goal/Question/Metric paradigm defined, earlier express the purpose of the analysis, define the data that needs to be collected, and provide a context in which to interpret the data. There are several types of statistical and analytical methods available, which ones we use depend upon the kind of experiment we are running. For this purpose we add an extra step to the GQM given earlier, planning the investigative layout and analysis methods:

1. Formulate goals
2. Develop and refine subgoals & questions
3. Establish appropriate metrics
- 4. Plan investigation layout & analysis methods**
5. Design & test data collection scheme
6. Perform investigation concurrently with data validation
7. Analyze data

Two important factors that should be considered when experimenting in the software domain: the individuals applying the technology and what they are applying it to. These two factors will loosely be referred to as the *subjects*, a collection of possibly multi-person teams engaged in separate development efforts, and the *projects*, a collection of separate problems or pieces of software to which a technology is applied. By examining the sizes of these two factors, which we will call the *scope of evaluation*, considered in an analysis, we obtain a general classification of analyses approaches.

### ANALYSIS CLASSIFICATION: SCOPES OF EVALUATION

#Teams per Project		#Projects_	
		<i>One</i>	<i>More than one</i>
<i>One</i>		Single Project (Case Study)	Multi-Project Variation
<i>More than one</i>		Replicated Project	Blocked Subject-Project

## Figure 4.1

Figure 4.1 presents the four part analysis categorization scheme, blocked subject-project, replicated project, multi-project variation, and single project case study [Basili & Selby 84]. The approaches can be characterized by the number of teams replicating each project and number of different projects analyzed. We need to quantitatively understand and evaluate the benefits and drawbacks of each of the approaches. Blocked subject-project studies examine the effect of possibly several technologies as they are applied by a set of subjects on a set of projects. If appropriately configured, this type of study enables comparison within the groups of technologies, subjects, and projects. Replicated project studies deal with a set of subjects separately applying a technology, or a set of technologies, to the same project or problem. Analyses of this type allow for comparison within groups of subjects and technologies. Multi-project variation studies examine the effect of one or a set of technologies as applied by the same subject across several projects. These analyses support the comparison within groups of projects and technologies. Single project analyses involve the examination of one subject applying a technology on a single project. The analysis must partition the aspects within the particular project, technology, or subject for comparison purposes.

The approaches vary in cost and the level of confidence one can have in the result of the study. Clearly, an analysis of several replicated projects costs more money but will generate stronger confidence in the conclusion. Unfortunately, since a blocked subject-project experiment, and even a replicated project, is so expensive to run, the projects studied tend to be small. This avoids the problem of scale-up, which is a major source of complexity in software development. The size of the projects may increase as the costs go down so it is possible to study very large single project experiments and even multi-project variation experiments if the right environment can be found. These latter two approaches tend to be more effective to use on real projects under development by a software organization. Although the statistical confidence in the results are weak, these experiments can be combined with the blocked subject-project and replicated project experiments to provide a basis for both statistical rigor and scale-up ability.

In what follows, at least one example of each of these approaches will be given within the area of preventing and detecting defects.

### METHODOLOGY EVALUATION USING BLOCKED SUBJECT-PROJECT ANALYSIS

This type of analysis allows the examination of several factors within the framework of one study. Each of the technologies to be studied can be applied to a set of projects by several subjects and each subject applies each of the technologies under study. It permits the experimenter to control for differences in the subject population as well as study the effect of the particular projects. The sample study discussed here is a testing strategies comparison [Basili & Selby 85]. The goal was to compare the effects of code reading, functional and structural testing with respect to 1) fault detection effectiveness, 2) fault detection cost, and 3) classes of faults detected. A secondary goal was to

compare the performance of software type and expertise level but only the first goal will be discussed here. Stated as GQM goals:

GQM Goals:

Analyze **code reading, functional testing and structural testing** for the purpose of **evaluation** with respect to their **effect on fault detection effectiveness, fault detection cost and classes of faults detected** from the viewpoint of the **corporation**.

Analyze **code reading, functional testing and structural testing** for the purpose of **evaluation** with respect to their **relative performance with regard to software type and expertise level** from the viewpoint of the **corporation**.

The experiments were run in two different environments: NASA/ GSFC and the University of Maryland. At NASA, we used professional programmers from NASA and Computer Sciences Corporation, at the University of Maryland, we used advanced undergraduate and graduate students many of whom has some professional programming experience. The projects were a text formatter, a plotter, an abstract data type, and a database program varying in length between 145 and 365 lines of code. The programs each contained software faults (9, 6, 7, 12 respectively) that were either made during the actual development of the program or were seeded based upon characteristic faults found in the local environment. Code reading was done by stepwise abstraction. Functional testing was done using equivalence partitioning boundary value testing. Structural testing was done with the goal of 100% statement coverage.

The experimental approach involved three replications of the experiment using 74 subjects on four different projects (32 from NASA/CSC, 42 from UM). The experimental design was a fractional factorial design, blocked according to experience level and the program tested. Each subject used each technique and tested each program. Figure 4.2 gives a layout of the experimental approach for the 32 subjects at NASA/CSC.

To assure that each of the techniques were used in their purest form and that the subjects did not mix techniques, code readers were given only the specification and the source code, functional testers were given the specification and the executable, and structural testers were given the source code, the executable and a coverage tool. The structural testers were asked to achieve the goal of 100% statement coverage and when they were done were given the specification and asked to locate faults.

Questions generated, based upon the goals included:

What was the number of faults found by each technique and overall?

How well did each person perform in evaluating the quality of the product after using each technique?

What was the fault detection rate for each technique and overall?

What classes of faults were detected by each technique?

What was the effect of experience level on fault detection?

What was the difference in performance with regard to software type?

Each of these questions is answered briefly below. For a more complete analysis see [Basili&Selby 1987].

Question: What was the number of faults found by each technique and overall? That is, which of the validation techniques detects the greatest number of faults in the programs?

The data collected for this question is the number of faults found in each project by each subject. The results of the study were that 4 faults were found on the average. This was about half the actual faults and the standard deviation was 1.9. At NASA/CSC, code reading was more effective than both testing techniques but functional testing was more effective than structural testing. Reading found 5.1 faults on the average, functional testing found 4.5 faults on the average and structural testing found 3.3 faults on the average. At the University of Maryland, code reading did about the same as functional testing but both did better than structural testing.

Question: How well did each person perform in evaluating the quality of the product after using each technique?

After applying each technique, the subjects were asked to assess how well they did. The code readers were most accurate in assessing their performance, i.e., they recognized that they found

### **Functional Factorial Design**

Blocking according to experience level and program tested  
Each subject uses each technique and tests each program

only about half the defects. The functional testers were the least accurate and overestimated how well they had done. After all the subjects had applied all the techniques, they were asked which technique they thought was most effective. Over 90% thought that functional testing had been most effective. Their intuition was wrong.

Question: What was the fault detection rate for each technique and overall? That is, which of the techniques had the highest fault detection rate (number of faults detected per hour)?

The data collected to answer this question was the number of faults found and the time spent by the subject in detecting faults. The average fault detection rate was 1.82 faults/hour average (SD = 1.80). At NASA/CSC, code reading was more cost effective than functional and structural testing, but there was no significant difference between functional and structural testing. Code reading found 3.3 faults per hour on the average while each of the testing techniques found 1.8 faults on the average. At the University of Maryland, we found no statistically significant differences in the techniques.

Functional testing took more CPU and connect time than did structural testing (code reading took none) but there was no statistical difference in the number of runs.

What classes of faults were detected by each technique?

Both code reading and functional testing did better than structural testing with regard to faults of omission and initialization faults. Code reading did better than both testing techniques on interface faults, and better than structural testing on computation faults. Functional testing did better than the other approaches on control flow faults. There was no difference detected in data and cosmetic faults.

Question: What was the effect of experience level on fault detection?

With respect to fault detection effectiveness, advanced subjects performed better than intermediate subjects and junior subjects but it was not possible to differentiate between intermediate and junior subjects. With respect to fault detection time, intermediate subjects did better than junior subjects.

Question: What was the difference in performance with regard to software type?

The percent of faults found in both the data type and plotter programs was about the same and a greater percent was found in these programs than in the text formatter or the database program.

The experimental design for this study permits a great amount of statistical analysis and provides the experimenter with a fair amount of latitude in studying the different aspects of the project. The drawbacks to the study are that the projects studied are small module size projects and the results do not necessarily scale up to the acceptance test phase of very large projects. The interpretation is more accurate for the unit test phase. The study does not provide sufficient insight into how the techniques might work on larger projects. This drawback is of necessity because the cost of replication is too expensive.

## METHODOLOGY EVALUATION USING REPLICATED PROJECT ANALYSIS

The replicated project analysis involves several replications of the same project by different subjects. Each of the technologies to be studied is applied to the project by several subjects but each subject applies only one of the technologies. It permits the experimenter to establish control groups. The sample study discusses here is an application of the Cleanroom Approach to software development as proposed by Harlan Mills. The Cleanroom approach focuses on defect prevention, rather than defect appearance and subsequent removal, during the software development life cycle.

Cleanroom uses a structured development process with an emphasis on human discipline via program verification, rather than computer aided program unit test debugging. The structured development activities include the use of formal methods, state machines as the specification notation for modules, functions as the specification notation for programs, and provides a strategy for verifying the consistency of specifications with design or code.

Top down design and development (with the use of stubs) are used to create a number of relatively small increments (or builds). This allows developers to concentrate on small parts of the system at any point in time. The increments are designed and coded together, with developers proceeding to the next build when the present build is submitted to the testers. Development progresses through stepwise

refinement, while each successive step is verified by stepwise abstraction to ensure correctness. Review/inspection activities that occur at various milestones. Reading during these activities is also supported by the use of formal methods.

In Cleanroom, there is a complete separation of development and certification (test) activities and personnel. The purpose of testing is for quality assessment, rather than for debugging or finding defects. Test cases are statistically generated on the system level, according to the operational profile of the target system. This allows the reliability of the system to be assessed according to its Mean Time To Failure (MTTF).

Top down development allows system testing to begin once the first increment is submitted, as each increment is fully executable when integrated with previous increments. The rate of growth of system reliability, as well as the final reliability at the end of a test suite, helps determine when sufficient testing has been completed.

The goal of this particular replicated analysis was to evaluate the effects of the Cleanroom approach on the process, product and subjects. More formally, the study stated as GQM goals was to:

GQM Goal:

Analyze the **Cleanroom process** in order to **evaluate** it with respect to the **effects on the process, product, and developers with respect to differences from a non-Cleanroom process** from the point of view of the **corporation**.

With respect to the environment, the study was performed at the university of Maryland, using upper division undergraduates and graduate students, several of whom had industrial software development experience. The problem was to develop and electronic message system of about 1500 LOC.

The experimental design was a replicated project using 15 three-person teams, 10 of which used Cleanroom. The teams that did not use Cleanroom used the same basic methods but were allowed to test their own program as well as having it tested independently. In both cases, the teams were allowed anywhere from three to five test submissions. Data was collected on the subjects backgrounds and their attitudes about using the approaches, all on-line activities performed, and the test results from the independent tests.

Specific questions posed for this study included:

What was the effect of Cleanroom on the application of the reading technology?

What was the effect of Cleanroom on computer resources?

What was the effect of Cleanroom on schedule?

What was the effect of Cleanroom on various static properties of the product?

What was the effect of Cleanroom on the reliability of the resulting product?

What was the effect of Cleanroom on the developer's attitude?

With regard to the effects on the development press, the Cleanroom developers felt they more effectively applied off-line review techniques, while others focused on functional testing. The Cleanroom developers spent less time on-line and used fewer computer resources. The Cleanroom developers also tended to make all their scheduled deliveries, while the non-Cleanroom developers tended to be late, including with the final delivery.

With regard to the effect on the delivered product, we can separate the results into static and operational properties of the product. The products from the Cleanroom developers had less dense complexity, a higher percentage of assignment statements, more global data, and more comments. Operationally, the products from the cleanroom developers more completely met the requirements and had a higher percentage of test cases succeed.

With regard to the attitude of the developers, the cleanroom developers said they missed program execution and modified their development style to satisfy the inability to test, i.e., they read more effectively. All but two said they would use it again.

The benefit of the study is that the results were soundly supported statistically because of the number of replications, and the projects were of a more reasonable size than the modules studied in the testing experiment. The drawback to this study again is that the projects were still smaller than most projects one might encounter in a practical environment and so it is not clear if the results would still hold if the project sizes were increased by an order of magnitude.

## METHODOLOGY EVALUATION USING MULTI-PROJECT VARIATION ANALYSIS

Multi-project variation analysis involves the measurement of several projects where controlled factors such as methodology can be varied across similar projects. This is not a controlled experiment as the previous two approaches were, but allows the experimenter to study the effect of various methods and tools to the extent that the organization allows them to vary on different projects.

The example study here is to characterize the errors by error origin for a series of projects that involve ground support software for satellites. More specifically the GQM goal was:

GQM Goal:

Analyze a **set of projects** in order to **characterize the errors by error origin** from the point of view of quality assurance.

The study was conducted in the Software Engineering Laboratory, at joint project between NASA Goddard Space Flight Center, the University of Maryland, and Computer Sciences Corporation. The application domain is ground support software for unmanned spacecraft control. Projects range from 50K to 120K source lines of Fortran code and the life cycle goes from design through acceptance test, i.e. the developers are given a set of functional specifications from which to develop the system and the test data. Defect data was collected from the time code was baselined in the code library through acceptance test.

The experimental design was a multi-project variation, i.e., the projects used programmers and managers from the same population.

Questions associated with the goal are defined below:

Process conformance:

What is the life cycle model?

The life cycle model is basically a waterfall model. However since most satellites have similar functions, a fair amount of code is reused from prior projects. Thus the project library is often seeded with code from a similar project which is analyzed for its reuse potential.

How well is it being applied?

The methods are quite standard and the process model is well understood by the developers.

Domain conformance:

How well do the developers understand the application?

Most of the developers were trained as mathematicians and had a reasonable familiarity with the application domain.

How well do the developers understand the requirements?

The requirements documents are not well developed, in that they are not always explicit. There are a lot a statements of the form: this function is defined similarly to the function for satellite X, except that ...

Quality perspective:

What is the distribution of errors by error origin (i.e., according to the misunderstanding that caused them)?

See figures 4.3, 4.4, 4.5. The phases for misunderstanding are labeled as requirements, functional specification, the design or coding of multiple components, the design or coding of a single component, programming language, environment, other. Note that design and code were combined because the nature of the development process where code reuse and design are intermixed.

You will notice that there are patterns in the histograms of all three projects. The majority of the errors or misunderstanding occur during the design or coding of a single component. This is because the overall design for satellites is reasonably well understood, by the time the system reaches code in the library, requirements errors have already been eliminated, and most of the lower level components are coded by new hires.

The benefit to this approach is that it does not require special experimental projects but allows for the evaluation of methodology in the normal development environment. The improvement algorithm discussed earlier can be applied to the environment in order to improve both the productivity and the quality of the software.

However, there are drawbacks to the approach. First, it requires that there are enough projects to recognize a pattern, i.e. there must be enough of a sampling to generate a statistical result. Second, since the experiment is not controlled, there is always the possibility of making mistakes in the interpretation, i.e. other factors that have not been controlled for may be causing the patterns we see. Third, if the Quality Improvement Paradigm is being used, we are changing our approach based upon our understanding of the kinds of errors we are making and changing our patterns over time.

#### METHODOLOGY EVALUATION USING SINGLE PROJECT/CASE STUDY ANALYSIS

Unfortunately, this is where most methodology evaluation begins. There is a project and the management has decided to make use of some new method or set of methods and wants to know whether or not the method generates any improvement in the productivity or quality. A great deal depends upon the individual factors involved in the project and the methods applied.

This sample study had a goal of trying to better understand the effects of design inspections on the kinds of faults detected. Specifically the GQM goal was:

GQM goal:

Analyze the inspection process in order to characterize it with respect to the error data and error correction effort from the point of view of the manager.

The environment is that of a major mainframe manufacturer. The product is the next release of a library tool which involved the development or modification of 40K source lines of code, with a total size of the final product of 100K SLOC. The system was developed in a PL/1 like language.

The experimental design is a single project case study. The specific question we will limit ourselves to in this case is:

What was the isolation and fix effort and total error correction effort for errors of omission and commission?

### **FAULT CORRECTION EFFORT (ISOLATION EFFORT PLUS FIX EFFORT) IN HOURS BY FAULT CLASS**

Average Effort	Commission	Omission	All
Isolation Effort	7.2	4.2	6.3
Fix Effort	3.7	3.9	3.7
Total Correction Effort	10.9	8.1	10.0

**Figure 4.6**

Based upon the data in figure 4.6, isolating a fault took almost twice as much effort as did fixing it. Correction (isolating and fixing) a fault of commission required more effort than correcting a fault of omission. Isolating a fault of commission required twice as much effort as a fault of omission. It should be noted that if we counted the effort for fixing a missing error as development effort, then the differences are even more dramatic. Thus, it is less costly to leave out a design or code segment than to include an incorrect one.

It should be noted that since this is a case study, we are looking at the data from one project in one environment and it is hard to generalize the conclusions. This is a drawback of a case study, it is hard to have a great deal of confidence in the conclusions we draw.

Combining the results of the four studies covered here, we might argue that we have learned something about preventing and detecting faults. Summarizing these results as lessons learned, we have:

Reading by stepwise abstraction is more effective and cheaper in uncovering defects than testing.

Readers have a better perception of the quality of the product than testers.

Developers rely on testing to find faults and when they know testing will occur will do a less competent job of reading.

There are fault patterns associated with specific software development environments.

Faults of omission are less expensive than faults of commission.

These are important pieces of information to know. Are they true in the environment you are working in? Clearly it is something you should know.