

[Presentation](#)

[Paper](#)

[Bio's](#)

[Return to Main Menu](#)

P R E S E N T A T I O N

W6

Wednesday, Nov 10, 1999

Measuring the Efficiency of Introducing a Systematic Tool Based Testing Process

Reuven Gallant & Israel Yosha



Measuring the Efficacy of Introducing New Methods in the Software Development Process

TESTART: ESSI Project 23683

M. Winokur, R. Gallant, A. Grinman, I. Yosha, S. Harley

10-Nov-99



Content



- ◆ What is TESTART
- ◆ The "Base Project"
- ◆ Measurable Goals & Objectives
- ◆ TESTART Project Major Steps
- ◆ Methods and Tools Selection
- ◆ Insertion of methods and tools
- ◆ Data Item Definition
- ◆ Historical Data
- ◆ TESTART project status
- ◆ Results
- ◆ Key lessons learned



What is TESTART



- ◆ Experiment in software improvement process with emphasis in requirements management and software testing.
- ◆ Sponsored by the ESSI (European Systems & Software Initiative) .
- ◆ The project initiated in April 1997 - 24 month duration.
- ◆ Dissemination in IAI will continue for several months.



Organizational Background

- ◆ TESTART is performed in an avionics "base project" at the TAMAM division of Israel Aircraft Industries (IAI).
- ◆ TAMAM has been assessed at CMM level 2 in May 96 and CMM level 3 in November 97.
- ◆ The "base project" is typical of embedded systems development at IAI's divisions.



The "Base Project"



- ◆ The project includes a new mission computer and the integration of new and existing subsystems.
- ◆ The mission computer contains:
 - * Central processing card based on Power PC for computing and communication.
 - * I/O cards for aircraft interfaces.
 - * Video card for symbology and video capabilities.



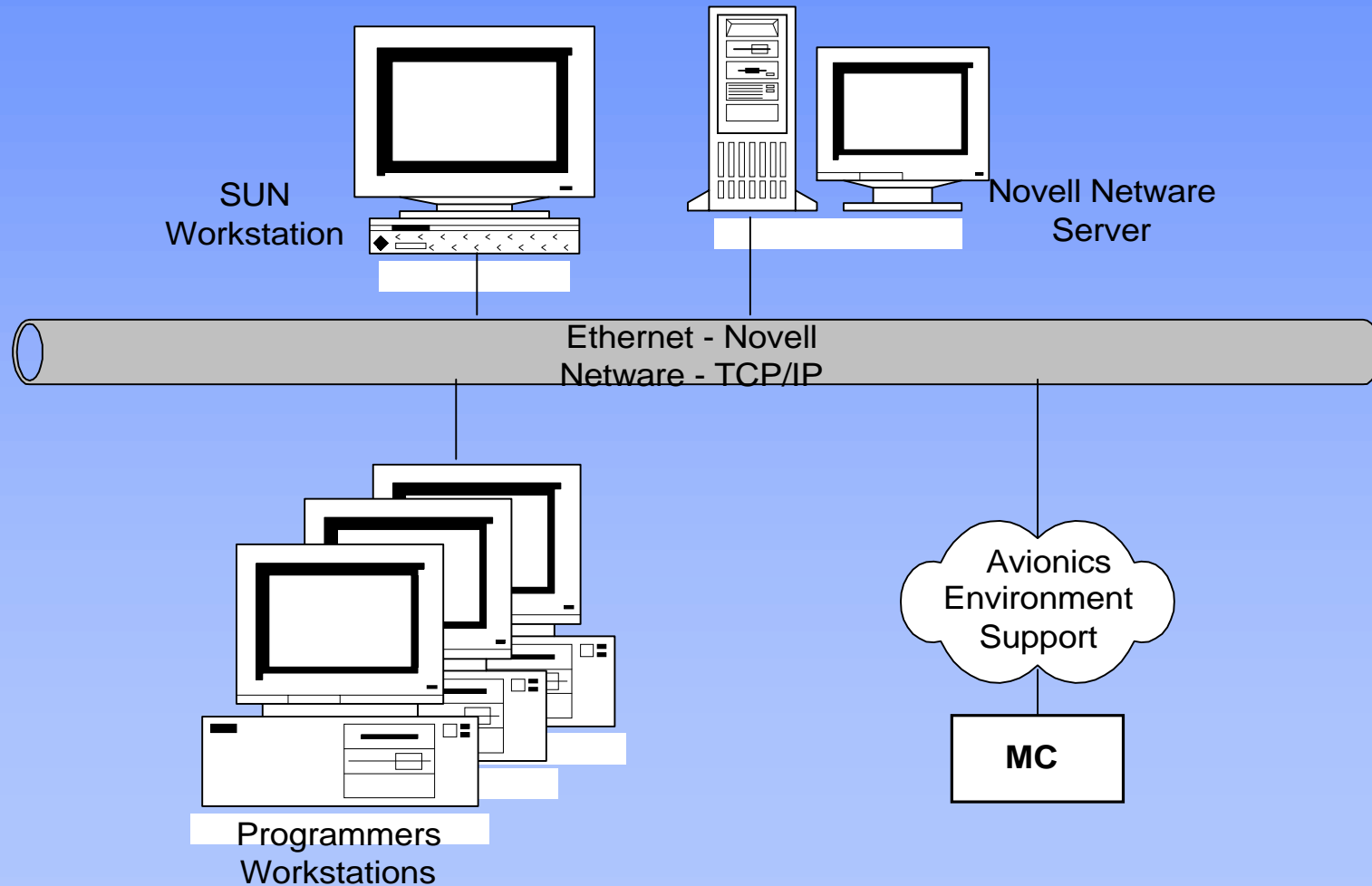
Base Project Development Environment



- ◆ Two main development environments:
 - * Phase 1 - Coding, unit testing and subsystem integration on PC workstations.
 - * Phase 2 - Subsystem and system integration on the real target.
- ◆ Coding languages: C and C++ .
- ◆ PC development environment: Windows, compiler - Borland C++ .
- ◆ Target environment: pRIZM+ , O/S - pSOS , compiler - DiabData.



Development Environment



10-Nov-99



Measurable Goals & Objectives

- ◆ Initial estimates of improvement:
 - * Increase requirements test coverage by 15%.
 - * Increase portion of code exercising in testing by 15% .
 - * Reduce integration phase by 5%.
 - * Reduce the overall software testing cost by 10% .



TESTART Project Major Steps



- ◆ Definition of methodology.
- ◆ Tools selection.
- ◆ Insertion of selected methods and tools into the base project.
- ◆ Definition and Collection of historical data for measurement reference.
- ◆ Collection of performance data from the base project.
- ◆ Analysis of results and drawing conclusions.



Methods and Tools Selection

- ◆ Methods were defined, and supporting tools selected in the areas of:
 - * Requirement management.
 - * Software testing.
- ◆ These methods and tools are complementary to the existing development process at IAI/TAMAM.



Requirements Management Tool Selection



- ◆ Two commercial tools have been studied:
 - * DOORS by Quality Software Systems.
 - * RTM by Integrated Chipware.
- ◆ The selection process included:
 - * Definition of the requirements management process.
 - * Analysis of the impact of each tool's features on the defined process.
- ◆ As a result, RTM was selected



Features Supporting RTM's Selection



- ◆ Class definition diagram.
- ◆ Graphical Audit trail.
- ◆ Graphical Interface for query and reports.
- ◆ Based on commercial database ORACLE for tracking large projects.

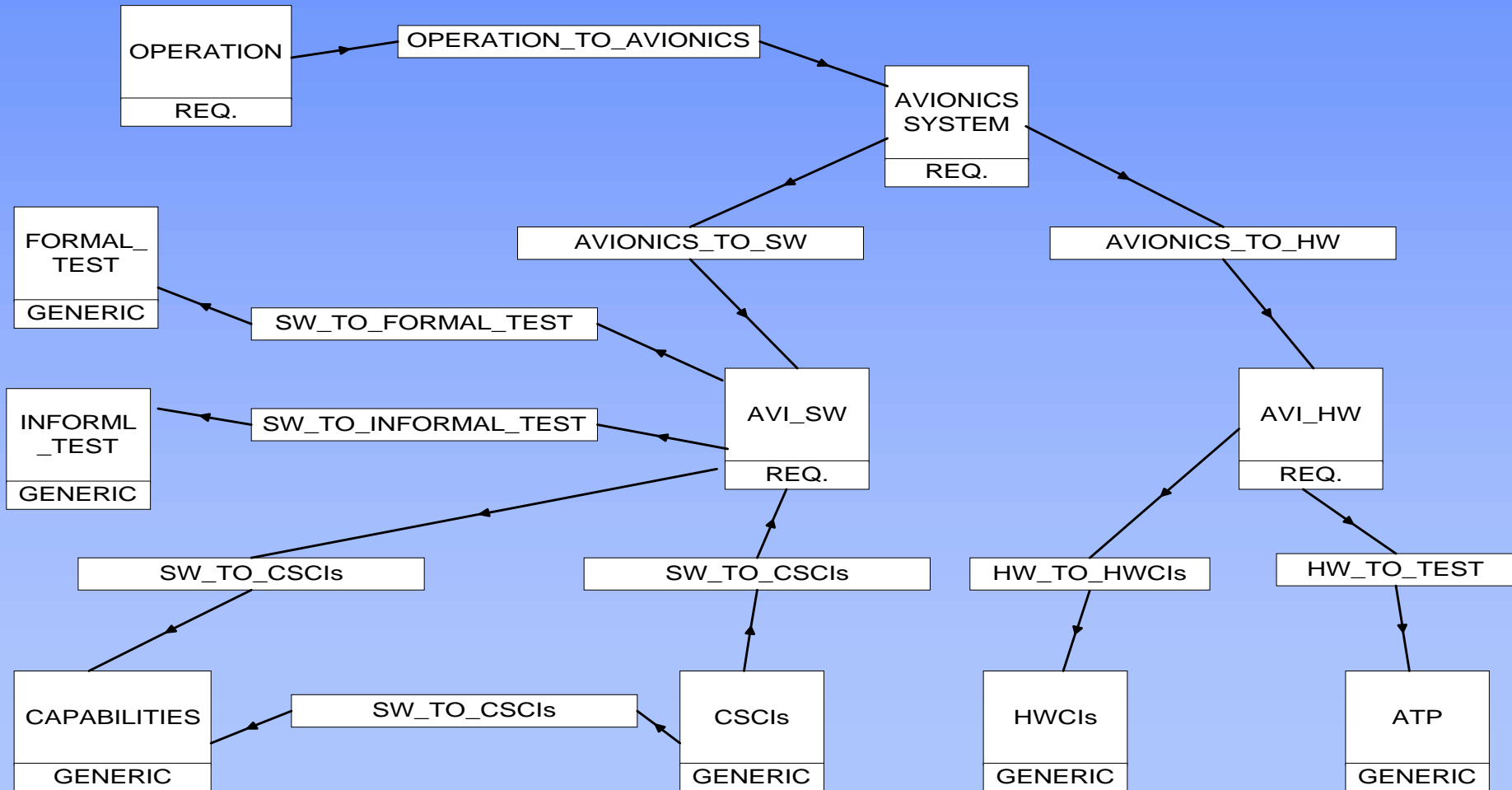


RTM Class Definition Diagram

- ◆ User defined project specific definition diagram:
 - * Classes and relationships (associations).
 - * Class attributes.
 - * Access rights.
- ◆ Provides utilization of project - tailored requirements management process.



Base Project class diagram



10-Nov-99



Software Testing Tool Selection

- ◆ Selection criteria for testing tool:
 - * Support static and dynamic testing.
 - * Support code coverage testing.
 - * Automatic code generation for drivers and stubs.
 - * Configuration management of test cases and results.
 - * User friendliness.
 - * Tool support by the vendor.
- ◆ The tool selected: Cantata from Information Processing Limited (IPL).



Use of CANTATA



- ◆ From the experience acquired in TESTART until now, we have defined the following:
 - * 50% of the code was tested in development environment at unit test level.
 - * The rest the code was tested on target at CSCI integration level.



Unit Test with Cantata

- ◆ 50% of the code was unit tested using the Cantata.
16.5 KLOC, 25 files and 243 functions.
- ◆ Parameters and data collected:
 - * Line of Code / Number of statements
 - ◆ Decision(if, Case), assertion coverage: 100%
 - * Complexity.
 - * Number of function calls.
 - * Number of errors found.
 - * Number of test cases.



Unit Test Analysis

- ◆ S/W measured parameters vs number of errors found.
 - * Function level analysis.
 - * File level analysis.
- ◆ S/W measured parameters vs testing duration.
 - * File level analysis.
- ◆ The non-normal distribution of results dictated use of non-parametric statistical analysis.



Unit Test Findings



- ◆ Positive correlation between error density and size of functions.
- ◆ Positive correlation between error density and number of calls to other functions.
- ◆ Correlation between error density and complexity found only in certain parametric ranges.
- ◆ No correlation found between size of unit tested and test duration, but learning curve found.



Insertion of Methods and Tools

- ◆ Requirements management and testing methodologies training.
- ◆ Tools training:
 - * provided by senior vendor representatives at user facilities.
 - * includes hands on exercising.
- ◆ Training was provided to the project technical staff and to core people within the organization.
- ◆ For each tool we tailored user manuals for project needs.



Insertion of Methods and Tools (cont)

- ◆ Tool interoperability:
 - * Interface between RTM and TAMAM's existing metric and requirements change management tool (CDSD).
 - * Interface between tools and existing PC development environment (Windows, Word, ...).



Data Item Definition

- ◆ To evaluate the quantitative impact of the experiment the following data items were defined:
 - * Relative cost of software testing [test cost / total cost].
 - * Software integration effort [man months / k-line of code].
 - * Cost of requirements change [hours / change].
 - * Coverage of requirements in software testing.
 - * Software code coverage.



Historical Data



METRIC	Proj. A	Proj. A1	Proj. B	Proj. C
1. Testing as a part of the overall project effort	= 15.3%	= 11%	= 16.6%	=8.57%
2. Integration effort per line of code (man months /kloc)	= 0.4			
3. Cost of Requirement Change [man hours / change]	23.3	16.7	13	8.75
4. Functional Coverage	Coverage =80%		Not available	Not available
5. Code Coverage	Coverage =55% According to literature			



TESTART Project Accomplishments



- ◆ Definition of methodology.
- ◆ Tools selection.
- ◆ Insertion of selected methods and tools into the base project.
- ◆ Definition and Collection of historical data for measurement reference.
- ◆ Collection of performance data from the base project.
- ◆ Analysis of results and drawing conclusions.



Results



- ◆ RTM is an integral part of the “base project” development environment:
 - * Formulation of the requirements baseline including 470 objects.
 - * Requirements change management using CDSD and RTM: 33 major requirements changes approved, 31 major requirements in process for approval. Total 64 requirement changes.



Results (Cont')

- ◆ Use of CANTATA for the unit test for 50% of the line of code.
 - * 16500 lines were tested, 64 errors found.
 - * Code coverage of the tested code was near 100%.
 - * Unit complexity (McCabe) generally less than 10, with few exceptions.
 - * Nesting level generally less than 3, with few exceptions.



Results (Cont')



- * Relative cost of software testing [test effort / total effort]: $2,000/18,000=0.11$
- * Software integration effort [Man Month / K-line of code]: $21/39=0.54$
- * Cost of requirements change [hours / change]: $2,200/33=66.6$ (not final).
- * Coverage of requirements in software testing: near 100%
- * Software code coverage: 76%



Key lessons - Organizational

- ◆ Interoperability of new tools with the existing environment has practical and cultural impact.
- ◆ Trends in future environment as well as portability / migration are key factors in tools selection.
- ◆ Early tangible benefits of tool usage are critical to acceptance by development staff.
- ◆ Gradual acquaintance with tool features increases the willingness to use them.



Key lessons - Technical

- ◆ Using RTM was found beneficial to the current project and future derivatives.
- ◆ The Unix based RTM is less convenient in Windows oriented environment.
- ◆ Interconnection between RTM and other CASE tools such as system architecture and design tools is required.



Key lessons - Technical (Cont)



- ◆ Cantata found to be effective in testing C language, less in C++.
- ◆ Code coverage was easy to achieve in logic and algorithmic functions, not practical for communication and I/O functions.
- ◆ Our experience until now with automatic unit testing shows that a suitable effort must be invested.
- ◆ Cantata for target found to be ineffective for real time I/O intensive applications.

Measuring the Efficacy of Introducing New Methods in the Software Development Process

Michael Winokur (mwinokur@hdq.iai.co.il), Reuven Gallant (rgallant@hdq.iai.co.il), Arie Grinman, Israel Yosha (iyosha@tamam.iai.co.il), Shlomo Harlev (sharlev@tamam.iai.co.il)
Israel Aircraft Industries
Ben Gurion Airport, Israel

Abstract

This paper is based upon a software Process Improvement Experiment (PIE) under the auspices of the European Systems and Software Initiative (ESSI) being conducted at Israel Aircraft Industries. The objective of the PIE is the application of advanced requirements management and software testing methods and tools in order to improve software productivity and quality in a measurable way. This controlled experiment was initiated in April 1997 and performed on a large ongoing avionics upgrade project. The paper describes the methods and tools selected for the experiment, the systematic process for their selection, the process of their deployment within the context of the existing organizational culture, and the data selected for collection to demonstrate the process improvement quantitatively. It presents lessons learned that can be applied by organizations that are investing in software process improvement.

1 Introduction

Israel Aircraft Industries (IAI) established a Corporate Software Process Improvement Program (SPIP) that defines, specifies and continuously updates, in cooperation with the operating divisions, the software processes to be employed in projects, and allocates tools to be used in implementing the improved processes. The program has been in operation since 1992.[1],[2]

In the framework of SPIP, a proposal was submitted and accepted by ESSI (European Systems and Software Initiative) to conduct an experiment using advanced requirements management and software testing methods and tools in a “real life” embedded project in order to improve software productivity and quality in a measurable

way. The experiment, performed on a large avionics upgrade ongoing project, identified as the “base project,” is organized as a separate project called TESTART (ESSI Project 23683).

The objective of TESTART is to test the hypothesis that systematic requirements management coupled with increased and improved code exercising at the unit test level, will reduce the overall time and effort required for testing activities without reducing product quality. The quantitative goals we want to demonstrate in the project are:

One. Increased requirements coverage: 15%;

Two. Increased portion of code exercised in testing: 15%;

Three. Reduced integration phase time: 5%;

Four. Reduced cost of overall software testing: 10%.

The TESTART project consists of the following major steps:

One. Definition of methodology (reuse of existing IAI methods and definition of new ones);

Two. Selection of appropriate tools;

Three. Insertion of the selected methods and tools into the base project;

Four. Collection of historical data for measurement reference;

Five. Collection of performance data from the “base project” (provided mainly by the new tools applied);

Six. Analysis of results and drawing conclusions.

The experiment was formally initiated in April 1997, and lasted 24 months. Preliminary results were published last year. [3] The present paper, which includes data and findings from the later development phases of the project, describes the methods and tools selected for the experiment and the systematic process for their selection and deployment within the context of the organizational culture, presents the type of data selected for collection

(historical and from the “base project”) and shows interesting, “fresh from the oven,” lessons learned. In particular, the paper emphasizes:

- One. The application of a success oriented and tightly controlled approach to the research of new development methods : pros and cons;
- Two. Problems encountered during the definition and collection of historical data, and the approach taken to overcome the problem.
- Three. Experience gained in using IAI’s requirements management method with a commercially available tool;
- Four. The method developed for automatic code exercising at the unit test level and the experience that we are gaining with a tool that supports this approach.
- Five. Quantitative results.
- Six. Insights into the significance of the gathered data and its analysis.
- Seven. Plans for exploitation of the gained experience throughout Israel Aircraft Industries.

2 Selection of tools

The tool selection process for the two areas, requirements management and software test, was a function of the depth of existing experience with such tools in the organization.

2.1 Requirements management tool selection

Regarding requirements management, two well known commercial tools (Quality Software System’s DOORS and Integrated Chipware’s RTM), each with a very different methodological emphasis have been studied, compared, [4] and deployed at IAI. The project was briefed by personnel well acquainted with the details of both tools. The project then defined its requirements management process. Next the project analyzed the impact that the features, strengths, and weaknesses of each tool would have on its ability to effectively implement the defined process. Detailed comparison of the tools in question is beyond the scope of this paper. However, several features of the selected tool, RTM, were crucial to its selection:

- User-defined and created Class Definition Diagram (see figure 1) provided a sound conceptual basis for visualization and solidification of the project-tailored requirements management process structure.
- The tool’s structured, graphical audit trail made it easy to understand and analyze the transformation

from free text customer document sentences to atomic testable requirements. (See figure 2).

- The tool’s support for focusing of duplicate and overlapping requirements was judged to be very important, since redundancy management is one of the most vexing problems in requirements management.
- The graphical interface for query and report generation did not require the user to learn a script language. This contributed to user perception of tool “friendliness” and thus to successful deployment.
- The tool’s commercial database (Oracle) and proven track record on large projects (projects with over one million requirements) was a highly significant risk reduction factor.

2.2 Software testing tool selection

Selection of a software testing tool was an entirely different matter. The organization did not have the benefit of prior experience with automated test tools. Consequently, a relatively large number (21) criteria were used to compare four different tools. Ultimately a smaller number of features proved to be crucial to selection of a tool:

- support for static and dynamic testing
- support for coverage testing
- support for code complexity analysis
- user-friendliness
- quality of support for the tool by the vendor
- price
- automatic code generation of code for stubs and drivers
- configuration management of test cases and results
- testing on target hardware

The intended use of the selected tool would be to invest in systematic unit testing in order to reap measurable benefits in integration testing. Thus only tools designed for white box developmental unit and integration testing were considered.

The tool ultimately selected for the PIE was Information Processing Limited’s Cantata.

3 Tool Training and Cultural Issues

Technical staff involved in the PIE received the following training:

- Intensive training in requirements management methodologies.
- Vendor-provided training in use of the requirements management tool and consultation

to the PIE on how to apply the tool within the context of the project.

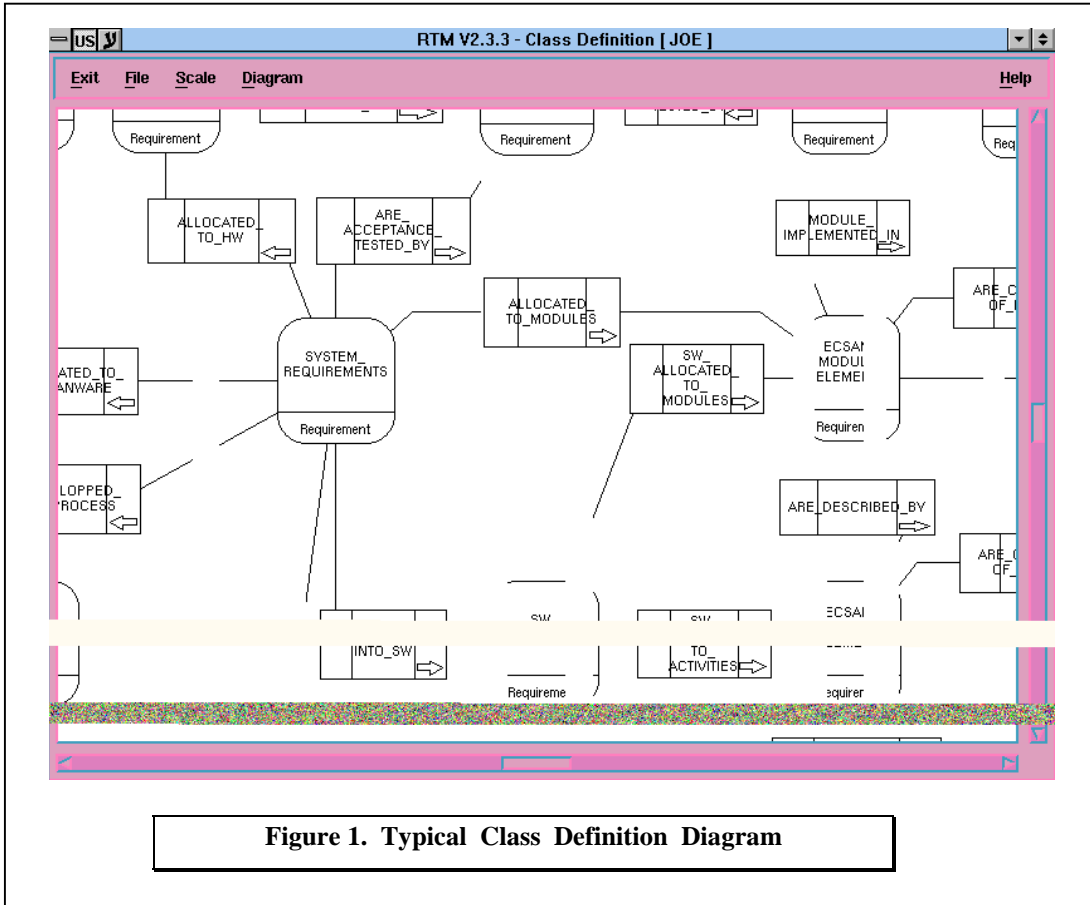


Figure 1. Typical Class Definition Diagram

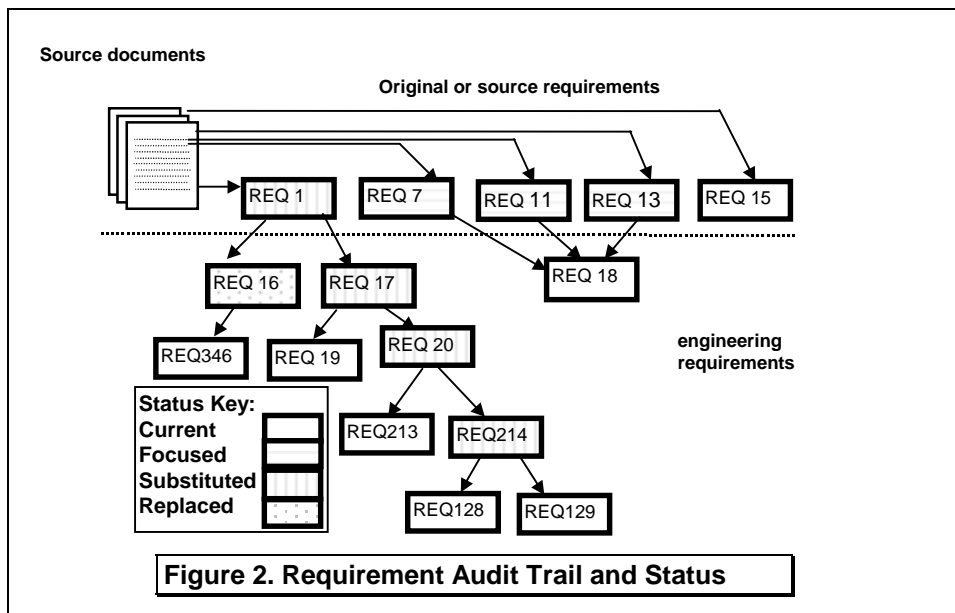


Figure 2. Requirement Audit Trail and Status

- Similarly, vendor-provided training in use of the test tool and consultation to the PIE on how to apply the tool within the context of the project.

The decision to provide this training to most of the PIE's technical staff, and not just to those tasked with utilizing the tools proved to be very beneficial. It produced a core of people within the organization who became well-conversant with subtle and powerful requirements management concepts (such as focusing, expansion, replacement, audit trail) and equally subtle and powerful test concepts (stubs, drivers, code complexity and coverage). As a result, people, some of whom were initially hostile to the use of complex tools, acquired a good understanding of the benefit of such tools, the overhead notwithstanding.

Involvement of senior people from the vendor organization in the training proved to be very beneficial. The persons tasked with using the tool were able to open up a dialog with the vendor that has continued, and contributed both to user buy-in and "ownership" of the tool. The users and vendors were able to understand each others approaches and priorities. Commitments were forged. The users committed themselves to understanding the tool features and underlying concepts. The vendors committed themselves to addressing user needs that were not addressed by these concepts. In the course of the project the dialog has continued: the vendor has benefited from continued insights into the users' needs. The users have requested and received enhancements that addressed their more crucial needs.

4 Results and Key Lessons Learnt

There were two prerequisite activities upon which PIE data gathering and analysis were based: gathering and analysis of historical data; and formulation of the requirements baseline. The requirements baseline was approved in January 1998, subsequent to the Preliminary Design Review of the base project. Henceforth the project entered a phase of intensive use of the requirements management tool in conjunction with the software test tool to manage derived requirements, plan and execute tests, and to gather data regarding requirements, tests, and the relation between them. As the effort proceeded, it provided an increasingly rich basis for analysis.

Even prior to completion of the integration phase of the project, there were a number of benefits from the PIE and a number of rather interesting lessons learned, of a qualitative, process-related nature, discussed in 4.1. These were complemented by quantitative results as the project advanced toward completion, discussed in 4.2. The broader question of impact of the PIE on the

organization and the course of future projects is addressed in section 5.

4.1 Tool Interoperability and Process Issues

1. The personnel tasked to use the tools underwent a cultural metamorphosis. They had been used to "managing requirements" manually within a Microsoft Word [5] document, and had to learn how to reap both the costs and the benefits of the tool's complexity. There was a certain amount of cross-fertilization between the "Microsoft" and upper CASE cultures. The users found that they could build a simple and easily adaptable interface between the tool's query generator and Word: they inserted "Desk Top Publishing tags" into the automatically generated query, and by "interpreting" these tags as user-defined paragraph styles in a Word Template. This provided a simple mechanism for control of the "look and feel" of documents based on tool data. Use of this simple scheme allowed the user to automatically generate system and software requirements documents, as well as complex traceability reports, a benefit which more than justified the initial overhead on entry of data into the tool database.
2. The organization's metrics management tool is based on a user-friendly Microsoft Access [6] based application. This tool is used to manage Requirements Change Requests. Because of its user friendliness and integration with the entire metrics management system, it was decided to continue entering requirements change data into this tool rather than RTM. In order to provide traceability between change requests and affected existing requirements, it was necessary to make provisions for transfer of requirements change request data from the metrics tool into RTM. This required a certain amount of interface coordination, but only negligible development effort, since both tools easily support ASCII export/import protocols.
3. The above two points address tool interoperability in batch mode. However the users identified a need for "on line" interaction between the "Microsoft" and "upper CASE" environments. Specifically, users wanted the ability to view and manipulate RTM data with a Microsoft Windows front end. When data is modified via this front end, update of the underlying RTM database should occur immediately and transparently. The vendor provided a Microsoft front end in the form of an extension to Microsoft Word supporting operation of RTM via an additional Word menu. Although

the Word interface did provide transparent and immediate RTM update, the developers identified several limitations that discouraged them from using it. Their comments have helped the vendor to prioritize planned tool enhancements.

The evolution of user attitude toward this interface is worthy of comment. Initially, the users eschewed this interface—preferring the full power of the pure RTM environment and not seeing the merit of putting up with the additional interface layer added by working within a Word environment. But as the Word interface matured in subsequent RTM releases, user interest heightened. Although certain tool features were not supported in the interface (e.g., requirement expansion), resourceful users soon found creative ways to use the interface as a workaround for tasks not readily supported within the RTM-only environment, such as copying of requirements from one class to another, and managing of changes in subsequent versions of customer documents. Paradoxically, as users became more keen about the potential for this interface, they also became more critical and discriminating regarding its shortcomings, most of which were due to limited depth of integration between RTM and Word.

4. Regarding the testing tool (Cantata), user interface, environment related problems and idiosyncrasies were also encountered. Specifically, the tool is primarily oriented to use in a non-Windows (i.e., character terminal mode) environment. Users have requested and have been receiving windows support of enhancements. Similar support was required to overcome compiler-related problems. The project's decision to develop new code in C++ further complicated matters, as IPL's C++ product (Cantata ++) was (at that time) less mature than its C product. The C product was adequate for checking of procedural code, but could not address object-oriented features such as constructors and abstract classes. However, after the initial labor pains the development staff arrived at an acceptable *modus operandi*, achieving near 100% code coverage on all code that was unit tested.
5. The major adjustment to use of the tool was not cultural and environmental but intrinsic to the testing activity. Specifically, the development staff had to come to grips with the realization that "there is no free lunch." The tool performs the requisite dynamic and static testing, supports automatic instrumentation, and provides generation of stubs and drivers. However, formulation of test cases, resolution of compilation problems, and test run execution duration have resulted in a greater than anticipated testing effort attributable to detailed

formulation and execution of test cases. These problems notwithstanding, the resources for these efforts have been allocated in the project schedule, and are used as a basis for measurement data analysis.

6. In addition to unit testing on the host (with "Cantata") unit testing on the target hardware (with "Cantata for target"). The principle objective of unit testing on the target hardware was to extend testing coverage to communication modules that interact directly with I/O devices and to extend testing to include operation in the real time multitasking target environment. However, the code instrumentation inserted by Cantata interfered with interaction with the physical devices and real time execution. A partial solution would have been to implement the communication protocol as stubs and Cantata test cases. Given the complexity of the protocol, this was judged not to be cost effective. In fairness to the vendor, the main use for which Cantata for target is intended, is to repeat tests that were performed on the host, to check for target-specific errors such as byte misalignment. The development staff felt that detection of such problems would surface during integration testing and could easily be tracked down and corrected with the aid of the target debugger. The advantage of earlier detection during unit testing did not justify, in the eyes of the group the overhead of target unit testing, which required configuring a dedicated serial port for test monitor and control.
7. To sum up points 4-6, Cantata was used effectively for unit testing in the host environment for 50% of the code. 64 errors were detected during unit testing. The remaining 50% of code, communication modules, was not unit tested. Integration testing, with ready access to the real environment was used in lieu of, and judged to be more effective than unit testing for these modules.

4.2 Historical data

Before presenting the historical data it is vital to discuss two caveats:

- (1) Today, a comprehensive metrics collection tool and system are in place throughout the organization. This allows the organization to measure and analyze a greater variety of data with a greater degree of precision than was possible for the "historical" projects. In particular, confidence for measurements of effort per development phase and activity is much higher for the base project than for "historical" projects.

- (2) The scope and complexity of the base project is much greater than for past projects. These differences tended to obscure the significance of comparisons

between past and present results. Historical data was collected, not in the belief that comparisons with it are necessarily valid, but rather to provide some perspective, however, imperfect, from the vantage of past experience.

Data was collected and analyzed for four prior projects. The collected data is consolidated in Table 2. The following defines the measurements that appear in Table 2, and explains the notation used therein:

1. Relative cost of Software testing=

$$\frac{\text{testing effort}}{\text{overall_project_effort}} \%$$

where both efforts are recorded in man months. The overall character of the testing, whether Formal Qualification Testing (FQT) or unit testing or both, is also indicated.

2. Software Integration effort=

$$\frac{\text{integration effort}}{\text{code size}} \%$$

where effort is recorded in man months and code size is measured in kilo lines of code.

3. Average cost to implement a single requirement change where cost is recorded in man hours (hrs).

4. Percentage of CSCI “atomic” functions covered by the CSCI FQT

5. Percentage of Software Code covered by software test cases

Both historical data collection and analysis posed interesting challenges, as discussed forthwith.

4.2.1 Historical data collection. When gathering historical data, the investigator had to reckon with the common phenomenon that “today’s” metrics were not necessarily measured in “yesterday’s” projects. In particular, this can be expected when the measurements are based on tools (requirements management and test) that have only recently been deployed in the organization. Of the aforementioned measurements, this was indeed the case for measurements 4 and 5.

In the absence of a requirements management tool, in the past, traceability between FQT test cases and requirements was managed at the paragraph level, rather than the atomic requirements level. Specifically, at the Software Requirements Specification level there was 100% test coverage. Atomic requirements were manually reconstructed by analysis of paragraphs;

test coverage of these requirements was estimated to be at the 80% level.

Code coverage metrics posed an even more daunting problem. No previous data existed. Thus it was necessary to extrapolate from figures reported in the technical literature, i.e., code coverage for complete functional blackbox testing is about 55% [7] [8].

4.2.2 Historical data analysis. Of the four projects chosen to provide historical data, two (projects A and B) were of the Full Scale Development (FSD) type, and two (projects A1 and C) had a strong reuse characteristic. Project A1 was an enhancement of A, and project C was based on a previous project, that, because of its research and development character, lacked complete documentation and testing.

All of the four projects were developed in accordance with DOD-STD-2167 or DOD-STD-2167A, with very stringent documentation and testing requirements.

These project attributes provide insight into the etiology of several historical data anomalies:

1. For both FSD projects, the testing effort included unit testing. For projects A1 and C, with strong reuse characteristics, only modified units were unit tested and unit test results were not recorded. Accordingly, test effort percentages on projects A and B were 50-100% higher than on projects A1 and C.
2. The unavailability of integration effort per kilo lines of code reflects the maturational process of metrics in the organization. Presently measurement of effort per development phase and measurement of code size is regularly performed on projects and recorded in the organizational metrics database. In the past, such data had to be collected manually, which was not always done. While measurements of integration duration could be reconstructed from the schedule, this was not judged to be very meaningful, since duration was very sensitive to project management issues (e.g., hardware delivery schedules) rather than to software development *per se*.
3. An average of 18 hours per requirement change, was calculated for all changes in the four projects considered. As discussed in section 4.3, this statistic must be approached with considerable caution, since it may reflect administrative issues, such as the number of changes grouped together in a single change request, rather than technical issues. Additionally the latitude given to customers to change requirements relative to the original contract varies greatly from project to project.

4. The problem of the granularity at which requirements were managed on the prior projects was discussed above. In the prior projects, requirements were managed at the SRS paragraph level for allocated requirements only. To estimate the *de facto* FQT coverage of detailed or derived requirements, identification of these requirements was performed for one CSCI, and the estimate was 80% as expected.

4.3 Analysis of TESTART Measurements

The true complexity of measurement issues soon became apparent. As experience and skill with use of both tools increases, there has been an increase in confidence that goals set for coverage and cost savings will be achieved. However, the most important measure is the overall effect on development cost and quality. Thus there is a potential for some measurement paradoxes that are worth noting:

1. One of the goals of the PIE was to increase percentage of requirements covered during testing. In fact, this measurement *per se* became a non-issue. Use of a requirements management tool with an underlying database virtually ensures 100% requirements coverage. Once a given text or graphic fragment has been identified as a distinct requirement in the database, the absence of an associated test will show up in routine database queries and be corrected by the addition of the necessary test or link.

Once requirements coverage is relegated to a non-issue, other measurements become significant: total number of requirements tested, requirements classification types, and level of detail of these requirements. Whereas the introduction of unit testing should tend to reduce integration and formal testing costs, the increased visibility given to detailed requirements of different types might tend, paradoxically, to increase the cost of FQT testing, since a greater number of detailed requirements will be tested.

Indeed, for the base project, as compared to the historical FSD projects (A and B), the integration effort was comparable (.52 man months per kloc for the base project vs. 0.4 for project A), and the relative cost of software testing was lower (11% for the base project vs. 15.3% and 16.6% for projects A and B, respectively).

Thus the integration and testing effort figures for the base project do not dramatically differ from the historical data. But even if they did, the verdict is not yet in. Typically, for complex projects, the initial

development costs are overshadowed by post-deployment maintenance costs. Thus, "the proof of the pudding" will be the costs incurred by the base project during the maintenance phase. For this reason, the organization has made a commitment to continue collecting and analyzing data during system integration and post-deployment maintenance phases, even though these phases are beyond the scope of the PIE's commitments.

2. Regarding testing effort, after introduction of the test tool and user education as to the gains to be achieved from more thorough and systematic unit testing, testing activity cost estimates and budget allocations had to be revised upward. Initially, developers complained that unit testing effort per module was approaching the original coding effort in its magnitude. However, after two significant problems in a single module (see table 3) were detected, the development staff became ardent advocates of tool-based unit testing, firmly believing that the heavy investment in testing was fully justified.

What was the benefit of detection of these errors during unit testing? Was it worth it. Here, one must bear in mind Lloyd S. Nelson's edict (cited by Demming [9]): "...the most important figures that one needs for management are unknown or unknowable, but successful management must nevertheless take account of them."

One can only speculate as to cost of not finding these errors during unit testing. Would they have been detected during unit testing, during flight test, during FQT, or after release? How many customers might have deserted? These figures are truly unknowable, yet they are the truly important ones.

3. Having said that, the knowable figures should not be discounted. The identified metrics, including requirements detail and requirements test coverage, code coverage, integration time and test effort are useful in that they provide a succinct snapshot of project and organizational trends relative to important and expensive development activities. The measurements to date from the base project summarized in table 1 below.

Table 1: Base Project Measurements Summary

Measurement	Result
Total requirements	466
System requirements	170
Avionics system requirements	131
Operational requirements	35
Avionics software requirements	130
Testing effort	2k man hrs
Overall development effort	18k man hrs
Testing effort relative to overall effort	0.11
Integration effort	21 man mon
Integration effort per kilo lines of code	0.54
Major req. changes approved to date	33
Average cost per req. change	66 man hrs
Functional req. coverage	near 100%
Total kilo lines of code	39
Main cpu kilo lines of code	35
I/O cpu kilo lines of code	4
Kilo lines of code unit tested	16.5
Errors found in unit testing	64
Unit Complexity (McCabe)	Generally < 10
Nesting level	Generally < 3
Code coverage of unit tested code	Near 100%
Code coverage	76%

As expected (or at least as is not surprising), comparisons with the historical data are not in and of themselves conclusive, for the reasons already alluded to above. In particular the base project had one significant characteristic that distinguished it from the prior projects. The customer of the base project was granted wide latitude to change requirements, above and beyond that formally granted in the contract. This accounts for the large effort per requirement change and mitigated against reduction of testing and integration efforts. The greater complexity of the base project relative to prior projects further mitigated against reduction of testing and integration efforts.

Mitigating factors notwithstanding it is nonetheless noteworthy that the relative integration effort of the base project was comparable to and the relative cost of testing was lower than prior FSD projects. The reduction in cost of testing is particularly noteworthy, given the increased overhead of rigorous unit testing.

The data from the base project was further analyzed with the objective of identifying trends that could be of help in improving performance on subsequent projects. Data pertaining to error density, function size, number of calls to other functions, unit size and test duration were analyzed with the aid of SPSS. [10] It soon became apparent that the non-normal distribution of the data dictated the use of non-parametric statistical analysis [11], in which the data is divided into a number of ranges

for the purpose of determining correlation. Application of this method yielded the following observations:

- There was a positive correlation between error density and function size.
- There was a positive correlation between error density and the number of calls to other functions.
- There was a correlation between error density and complexity, but only if particular thresholds for classifying error rates as “moderate” or “high” were selected.
- No correlation was found between unit size and test duration, but there was a learning curve that reduced test duration as the project progressed.

These results suggest that it is important to manage function size and code complexity, and where the complexity is justified, to subject “at risk” modules to greater scrutiny via inspections and testing.

5 Technology Transfer

Dissemination of results throughout IAI has already begun. The development personnel of the base project frequently host visits from other divisions, who want to learn at first hand of the PIE’s experience with advanced requirements management, metrics and test tools. The PIE participants have turned out to be both discerning critics and cautious advocates of these tools. The discernment and cautiousness has enhanced the PIE participants credibility, and helped other division begin introduction of advanced tools in a realistic manner.

The measurement data has provided up to date benchmarks regarding a wide variety of important metrics, spanning, requirements, coverage, complexity, errors and development effort, as well as applicable statistical analysis methods. The complexity of the base project, which is typical of the increasingly challenging projects being undertaken throughout IAI, adds credence and relevance to these benchmarks for other organizations.

Finally, the ongoing analysis and enrichment of the measurement data in the light of ongoing and future system integration and post-deployment maintenance is made available to the IAI Software Process Improvement Program (SPIP), which through its divisional representatives is disseminating the PIE results

6 Summary

The following are the key lessons learned to date:

1. Tool evaluation criteria and selection processes should be tailored to the level of experience of the organization with the relevant tools and methods.
2. Tool and method training is crucial for the successful deployment of tools and methods. Involvement of the vendor in this training is an excellent opportunity to forge a mutually beneficial partnership between the vendor and the users.
3. It is inevitable that introduction of a new tool into an existing environment will result in culture conflicts. In particular the culture's preexisting tool environment must be taken into account. Tool interoperability, and the ease with which the user can tailor this interoperability to his needs are crucial factors in the resolution of cultural conflicts.
4. On the other hand, as developers become more acquainted with the power of a tool, they better appreciate its added value, and are more willing to tolerate and even enthusiastically accept the increased overhead.
5. Motivation to apply new tools and methods in the face of difficulties cannot be based on theoretical predictions of down stream return on investment. Immediate, tangible benefits are crucial to acceptance by the development staff. In fact, these benefits emerged early in the PIE. The ability "at the touch of a button" to produce any variety of complex requirements traceability reports and to automatically generate requirements documents "sold" the development staff on the requirements management tool. For testing, the immediately perceived benefits were attainment of 100% code coverage, detailed static analysis, and most importantly, the early detection of two major defects in the code at the first trial of the tool on a "real life" software unit.
6. Comparisons with historical data are problematic, but are nonetheless useful for setting a context of expectations and "stretch goals" for process improvement. Accurate assessment of benefit in this project will require careful evaluation of experience during system integration and maintenance phases.
7. Statistical methods must be carefully chosen for suitability to the data characteristics. The results of appropriate statistical analysis can point to potential hazards and improvement trends that can be expected and monitored on subsequent projects taking advantage of advanced requirements management, testing and statistical analysis tools.
8. A multidimensional quantitative and qualitative snapshot of a project whose complexity is comparable to other leading edge projects throughout the organization, is potentially very useful. This snapshot and its ramifications must be carefully

disseminated with the aid of the corporate process improvement program mechanisms.

Acknowledgments

The formulation of this paper would not have been possible without the contribution of data and analyses by the participants in the TESTART project, including Shlomo Glick, Doron Cherkovsky, Baruch Hibner, and Itzhak Lavi. The statistical analysis was provided by Henia Berkovitz.

References

- [1] I. Lavi, M. Winokur and S. Glick, "Two Tiered Organization Process Focus Implementation," *Proceedings of the European SEPG Conference*, London, 1998.
- [2] I. Lavi, M. Winokur and R. Gallant, "Results of a Software Process Improvement Program in a Large Corporation," *The Twelfth International Conference of The Israel Society for Quality*, Jerusalem, Israel, 1998.
- [3] M. Winokur, A. Grinman, I. Yosha and R. Gallant, "Measuring the Effectiveness of Introducing New Methods in the Software Development Process," *Proceedings of the 24th EUROMICRO Conference*, IEEE Computer Society, Vaesteras, Sweden, 1998.
- [4] R. Gallant, M. Winokur and J. Kudish, "Tool and Method Reciprocity: A Case Study in Requirements Management," in *Proceedings of the 7th Israeli Conference on Computer Systems and Software Engineering*, I.E.E.E. Computer Society Press, 1996.
- [5] Microsoft Word for Windows 95, Version 7.0a, Copyright © 1983-1996 Microsoft Corporation.
- [6] Microsoft Access for Windows 95, Version 7.00, Copyright © 1989-1996 Microsoft Corporation.
- [7] R. Grady, *Practical Software Metrics for Project Management and Process Improvement*, Prentice Hall, 1992.
- [8] F. McGarry, "Re-use of Software Engineering Assets in a Production Environment," Australian Software Engineering Conference, 1997.
- [9] E. Demming, *Out of the Crisis*, M.I.T. Center for Advanced Engineering Study, 1986, p. 121.
- [10] *SPSS Base 8.0 for Windows User's Guide*, SPSS Inc., Chicago, 1998.
- [11] D. Card and R. Glass, *Measuring Software Quality Design*, Prentice-Hall, Englewood Cliffs, New Jersey, 1990.

No.	Metric	Project A	Project A1	Project B	Project C
1	Relative Cost of Software Testing (as part of overall project effort)	Test Type: FQT + Unit Testing = 1500 man hrs testing + 2500 man hrs documentation and SQA Total effort = 26000 man hrs Percentage: 15.3%	Test Type: FQT Mainly Testing = 810 man hrs Total effort = 7300 man hrs Percentage: 11%	Test Type: FQT + Unit Testing = 2000 man hrs Total effort = 12000 man hrs Percentage: 16.6%	Test Type: FQT Mainly Testing = 1100 man hrs Total effort = 14000 man hrs Percentage: 8.57%
2	Integration effort per kilo line of code	= 0.4			
3	Cost of Requirements Change	16 changes 373 man hrs 23.3 man hrs / change	6 changes 100 man hrs 16.7 man hrs / change	2 changes 26 man hrs 13 man hrs / change	8 changes 70 man hrs 8.75 man hrs / change
4	Functional Coverage	New Video Card, 28 of 35 functions tested Coverage = 80%		-----	-----
5	Code Coverage	According to literature, code coverage is about 55% when blackbox test only is executed [7], [8]			

Table 2: Historical Data from Prior Projects

	A				B									
Lines in file	182				167									
Code lines	104				139									
Functions	4				10									
Function lines	12	8	45	26	21	7	9	9	24	7	22	7	17	9
For loops														
IF statements			4	2	3				2		2		3	2
Switch statements			1											
Return statements		1	1			1	1			1		1	1	2
Break statements		9												
Case Labels		9												
Go to Labels														
Decision outcomes			17	6	6				4		4		6	6
Function calls			15	7	2	1	1	1	2	1	2	1	2	1
McCabe	1	2	13	4	4	1	1	1	3	1	3	1	4	4
Decision coverage	100%				100%									
Statement coverage	100%				100%									
Assertion coverage	100%				100%									
Errors found					2									
Testing time (hrs)	26				35									

Table 3: Results of Unit Testing on Two Modules

Reuven Gallant & Israel Yosha

Reuven Gallant has been a staff member of the Embedded Computer Systems Department of the Corporate R&D Organization at Israel Aircraft Industries (IAI) since 1989. His chief responsibility is research and project consulting in the area of Software Engineering methods and tools. As a participant in Israel Aircraft's Software Process Improvement Program (SPIP) he assists IAI divisions in their Capability Maturity Model (CMM) based process improvement efforts. He is an authorized Lead Assessor in the Software Engineering Institute (SEI) Appraiser Program for CMM-Based Appraisal for Internal Process Improvement (CBA IPI). Prior to emigrating to Israel from the U.S. in 1987, he was employed at Sikorsky Aircraft, where he was a Digital Systems Group Leader and chaired the software tools committee.

Gallant received B.S. and M.S. degrees in electrical engineering from M.I.T. and a Ph.D. in Religious Studies from Yale University.

Israel Yosha is presently the avionics system engineer and software development manager for a helicopter upgrade program, at the TAMAM division of Israel Aircraft Industries, Ltd. During his 18 years at TAMAM, he has served as software development manager and technical project manager for a number of electro-optics systems. For several years he served as the senior software and hardware design engineer for HADAX Electronics Inc., New Jersey, where he developed microprocessor based embedded communication systems.

Yosha received a BS in electrical engineering from Tel Aviv University, Israel, and an MS in computer science from Fairleigh Dickinson University, Teaneck, New Jersey.

The helicopter upgrade program on which he is presently serving is the base project for the PIE discussed in this paper.