METHODOLOGY EVALUATION:
EFFECTS OF INDEPENDENT VERIFICATION AND INTEGRATION ON ONE CLASS OF APPLICATION

Jerry Page
COMPUTER SCIENCES CORPORATION
and
GODDARD SPACE FLIGHT CENTER
SOFTWARE ENGINEERING LABORATORY

Prepared for the
NASA/GSFC
Sixth Annual Software Engineering Workshop

METHODOLOGY EVALUATION:

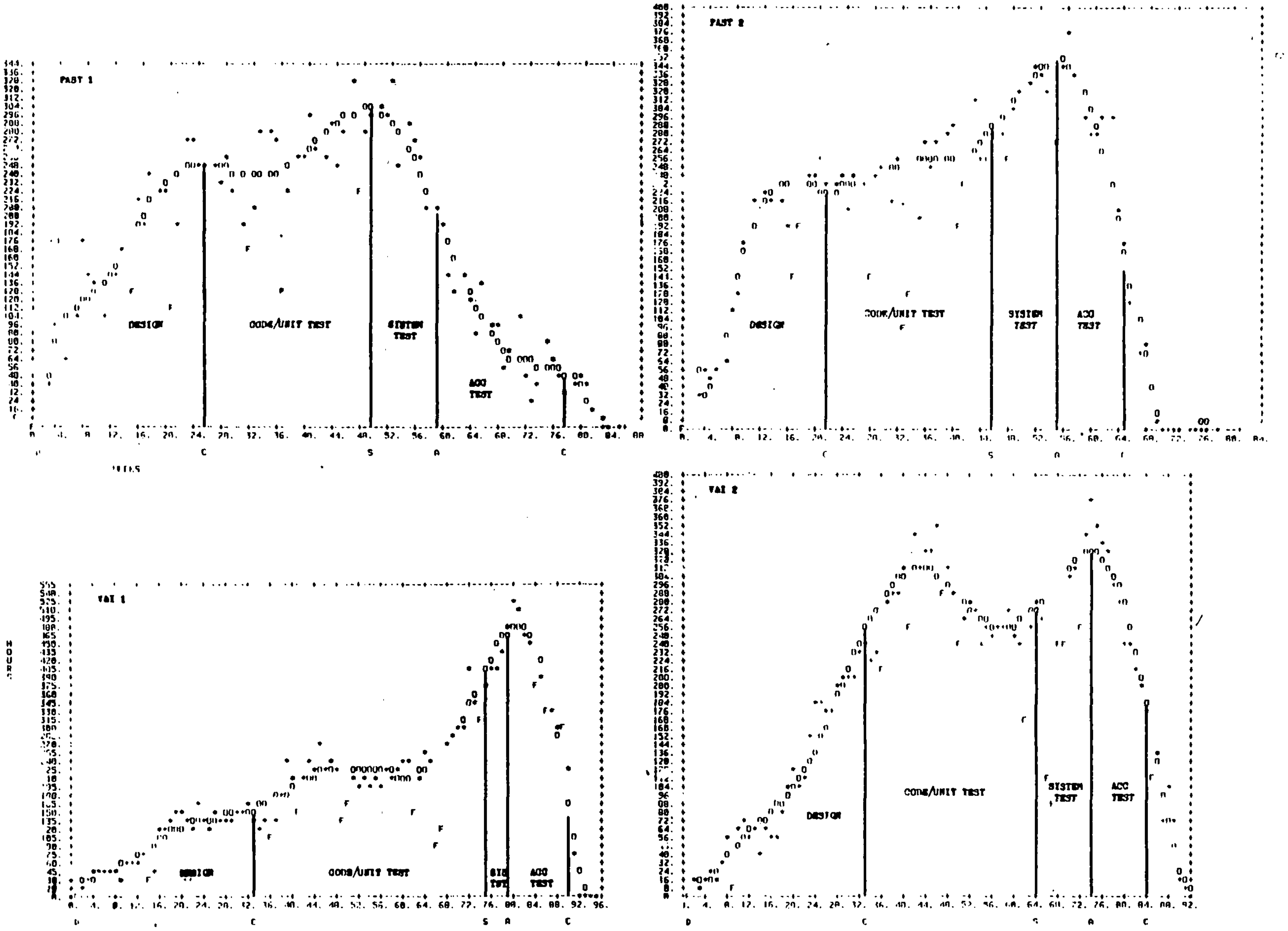
EFFECTS OF INDEPENDENT VERIFICATION AND INTEGRATION ON ONE CLASS OF APPLICATION

J. Page
CSC

Viewgraph 1: Title

One area of study in the Software Engineering Laboratory (SEL) is methodology.[1] This presentation describes the effects of an independent verification and integration (V&I) methodology on one class of application. V&I is the name that we will use for what some call independent verification and validation (IV&V) and others call verification and validation (V&V). "One class of application" means the development of solutions for a set of similar problems (ground-based support for satellite operations) that are developed in the same computing environment--simply put, a specific problem in a specific environment.

[1] Goddard Space Flight Center, SEL-81-104, "The Software Engineering Laboratory" (Software Engineering Laboratory Series), D. N. Card et al., February 1982.



J. Page
CSC
3 of 47

174-SEL-(33)-2

Viewgraph 2: Resource Profiles

Why use a V&I methodology? Why have we experimented with a V&I methodology? To introduce V&I methodology, let me show you resource profiles for four real projects developed for the Goddard Space Flight Center (GSFC) by Computer Sciences Corporation (CSC) and monitored closely by the SEL. These resource profiles show technical hours charged to the projects by week. Technical hours are those hours charged by the programmers and the first-line managers. First-line managers are those managers who make decisions, set priorities, and solve problems daily, as opposed to higher level managers who receive weekly or less frequent progress reports. These resource profiles also do not include service charges, which amount to approximately 13 percent of the hours charged to a project. Service hours include those hours charged by librarian, secretarial, technical publications, and data technician support groups.

In these profiles, design activity starts at the far left-hand side and continues throughout the project at decreasing levels. The first vertical line indicates the conclusion of a series of requirements analysis and critical design reviews. It is the point at which implementation and corresponding testing are allowed to begin. The second vertical line is the point at which implementation (coding) is supposed to be complete and system testing starts. The third vertical line is the point at which the software is supposed to be ready (for operation) and acceptance testing starts. The fourth vertical line indicates the end of acceptance testing and the beginning of maintenance (by another group).

Most people who measure software products apply many measures to the software product from the point at which it enters the maintenance and operation (M&O) phase. We do too, but since we have no responsibility for the software once it is transferred to the maintenance group and because it is more difficult to collect data through another group, we apply many of our measures one or two phases earlier, i.e., from the beginning of acceptance testing or from the beginning of system testing.

As you can see from three of these four profiles (excluding the one in the upper left-hand quadrant), the peak effort is at the start of acceptance testing. Some of the reasons that the peak effort occurs at that point are

- All the projects grow between 15 and 40 percent after the start of implementation because of requirements escalation.
- These projects cross two or three funding periods. This puts some constraint on how much work can be done in any one funding period.
- Management problems exist. The profile in the lower left-hand quadrant shows the application of the "mythical man-month."
- There is a hard deadline (launch of a satellite).
- The computers are not very reliable (6- to 8-hour mean time to failure).

We know what we are doing during that peak effort (the peak at the third vertical line). A large fraction of our work there is correcting errors.

It is commonly accepted that the cost to correct an error approximately doubles as it enters each new phase of the development life cycle. For example, if an error originates in the requirements phase (the phase preceding design) and if that requirements error gets designed, the cost to correct the error during design will be one to two times more than to correct the error in the requirements phase. If the designed requirements error gets implemented, the cost to correct the error during implementation will be two to four times more than to correct the error in the requirements phase. If the implemented requirements error enters the system testing phase, the cost to correct the error will be four to eight times more. If the implemented requirements error enters the acceptance testing phase, the cost to correct the error will be 8 to 16 times more. If it enters the M&O phase, the cost to correct the error will be 16 to 32 times more (for one simplified example, see Figure 1).

The same progression holds for errors that originate in design and implementation. Therefore, during the M&O phase, even implementation errors are costly to correct; they cost four to eight times more to correct during the M&O phase than during the implementation phase.

We do not need a general hypothesis to know that it costs more to correct errors in the later stages of development. Our own data collected over the last 5 years shows that some increase occurs in the cost of correcting errors from one phase of development to the next. SEL data shows that (regardless of error type) the average error discovered during the acceptance testing phase costs more to correct than the average error discovered during the system testing phase and that the average error discovered during the system testing phase costs more to correct than the average error discovered during the implementation phase. The increase in the average effort to correct the average error from one phase to the next varies from project to project, but it frequently approximates a doubling of effort.

Common sense indicates that there will be cost increases for changes to the evolving product as development progresses through the life cycle. Certainly, in this environment there are several transfers of responsibility: from the requirements team to the development team, from the

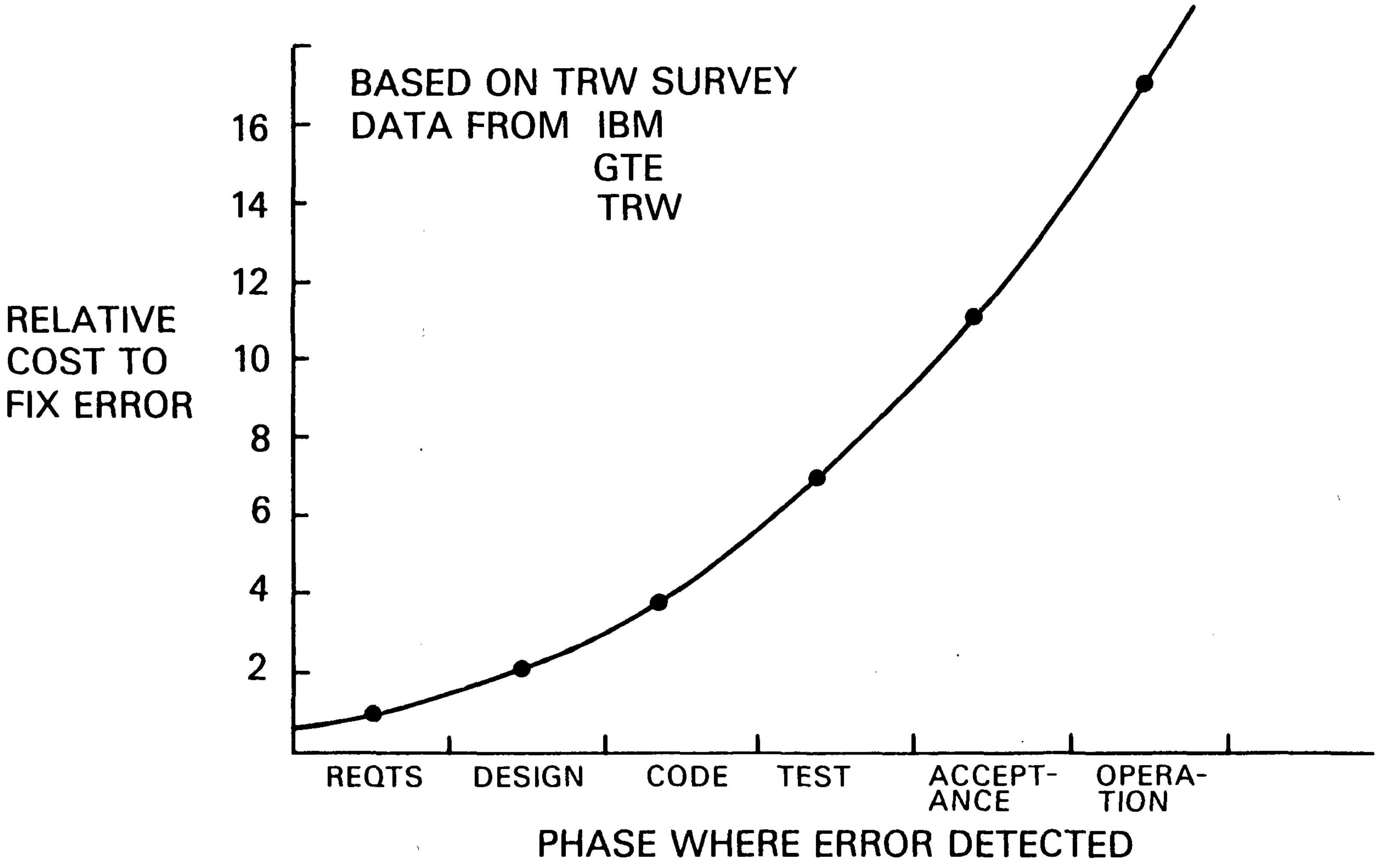


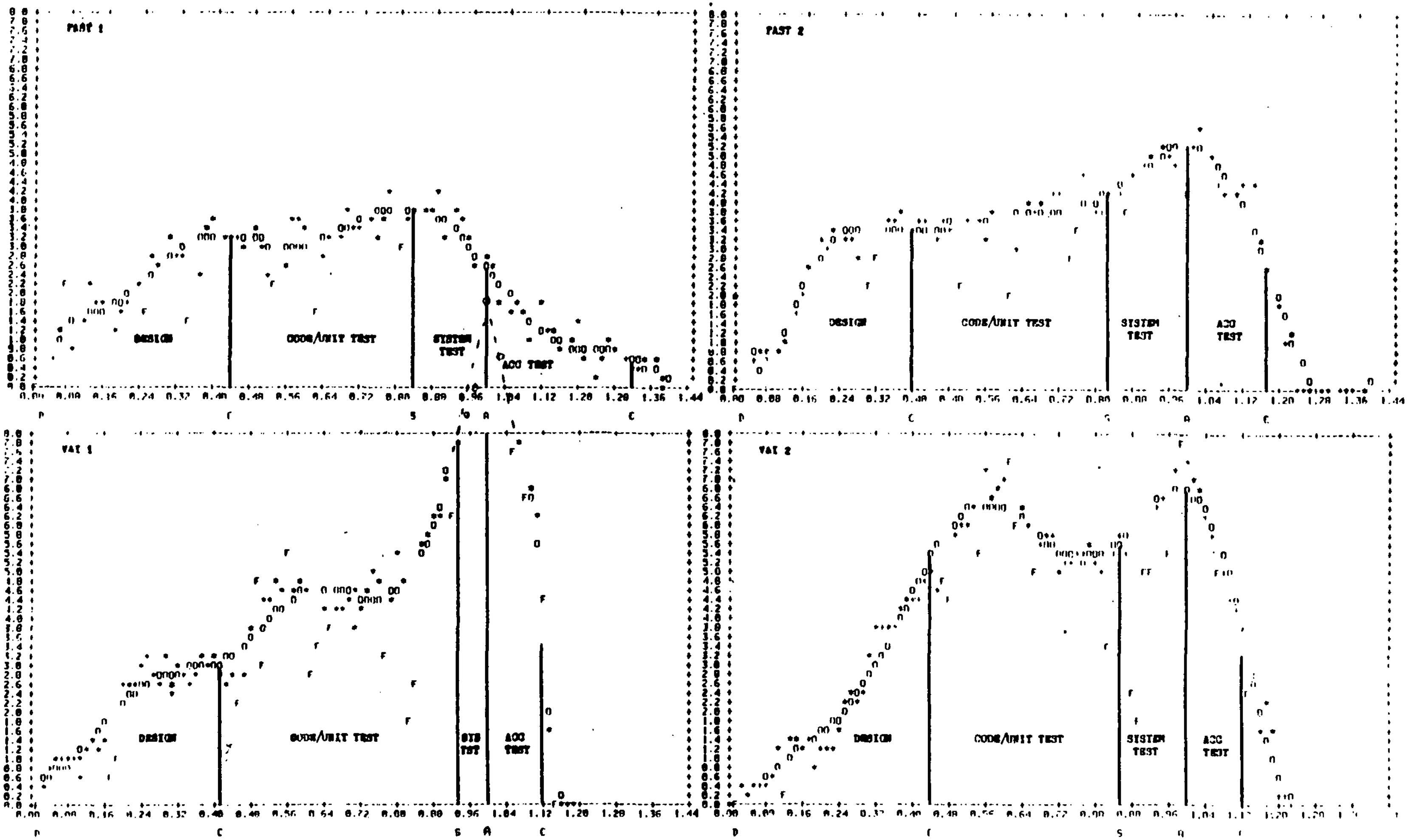
Figure 1. Cost of Correcting Software Errors

designers to the implementers, from the implementers to the testers, and finally, from the development team to the maintenance team. These are not complete transfers of responsibility; instead, the team size increases or decreases at different points in the development life cycle. Because a system is never 100-percent completely or accurately documented and because few people can instantaneously absorb the content of the documentation, new team members will require additional time to become familiar with the system. Therefore, functions will increase in cost when new members or groups become responsible for them.

Since the average development team size is six members, prematurely removing one member from the team always affects the schedule adversely. If the schedule cannot be adjusted (adjustments are more difficult late in the life cycle because of launch deadlines), then a replacement member must be added to the team. This replacement increases cost and it does not solve the schedule problem completely unless the replacement individual is more productive than the individual who was replaced.

We know that we have to improve our methodology, both in management and development practices, to move error-correction efforts earlier into the development life cycle, closer to the commission of the errors.

We know this from the advocates of V&I methodology, from our own SEL data, and from common sense. To save money, we must move the peak effort away from the start of acceptance testing (the third vertical line in the resource profile) and nearer to the design phase (between the first and second vertical lines in the resource profile). For example, we spend approximately 30 percent of our dollars for system and acceptance testing (the area between the second and fourth vertical lines). If 50 percent of that expenditure is for error correction (15 percent of dollars), then by moving that error-correction effort into the implementation phase, we will reduce the cost of that effort by approximately one-half; i.e., we will save approximately 7.5 percent of our development cost.


PAST 1
PAST 2
VAT 1
VAT 2
DESIGN
CODE/UNIT TEST
SYSTEM TEST
ACC TEST
SYS TEST
0.00 0.08 0.16 0.24 0.32 0.40 0.48 0.56 0.64 0.72 0.80 0.88 0.96 1.04 1.12 1.20 1.28 1.36 1.44
0.0 0.2 0.4 0.6 0.8 1.0 1.2 1.4 1.6 1.8 2.0 2.2 2.4 2.6 2.8 3.0 3.2 3.4 3.6 3.8 4.0 4.2 4.4 4.6 4.8 5.0 5.2 5.4 5.6 5.8 6.0 6.2 6.4 6.6 6.8 7.0 7.2 7.4 7.6 7.8 8.0
D
C
S
A
C

174-SEL-(33)-3

Viewgraph 3: Scaled Resource Profiles

These resource profiles are scaled so that the start of acceptance testing is 1 on the x-axis. The technical hours spent each week (the y-axis) are scaled by the developed lines of code (in thousands). The scaled resource profiles show technical hours per thousand lines of developed code by fraction of development life cycle. The unscaled resource profiles (see viewgraph 2) show technical hours by week of development life cycle.

DEVELOPMENT ENVIRONMENT

CHARTER: DESIGN, IMPLEMENT, TEST, DOCUMENT

TYPE OF SOFTWARE: SCIENTIFIC, GROUND-BASED, NEAR-REAL-TIME, INTERACTIVE GRAPHIC

LANGUAGES: 85% FORTRAN, 15% ASSEMBLER MACROS

MACHINES: IBM S/360-75 AND -95, BATCH WITH TSO

PROCESS CHARACTERISTICS:	AVERAGE	HIGH	LOW
DURATION (MONTHS)	15.6	20.5	12.9
EFFORT (STAFF-YEARS)	8.0	11.5	2.4
SIZE (1000 LOC)			
DEVELOPED	57.0	111.3	21.5
DELIVERED	62.0	112.0	32.8
STAFF (FULL-TIME EQUIV.)			
AVERAGE	5.4	6.0	1.9
PEAK	10.0	13.9	3.8
INDIVIDUALS	14	17	7
APPLICATION EXPERIENCE			
MANAGERS	5.8	6.5	5.0
TECHNICAL STAFF	4.0	5.0	2.9
OVERALL EXPERIENCE			
MANAGERS	10.0	14.0	8.4
TECHNICAL STAFF	8.5	11.0	7.0

Viewgraph 4: Development Environment

I will talk about four projects today. Two went into operation about 2 years ago; the other two went into operation about 3 months ago. A V&I methodology was applied to the last two. The last two projects will be labeled V&I 1 and V&I 2 on the following viewgraphs. The projects that became operational 2 years ago will be labeled Past 1 and Past 2.

Date	Past 1	Past 2	V&I 1	V&I 2
Development start	May 1978	June 1978	Oct. 1979	Oct. 1979
Maintenance start	Oct. 1979	Aug. 1979	June 1981	May 1981
Operation start	Feb. 1980	Oct. 1979	Aug. 1981	Aug. 1981
M&O end	Active	Sept. 1980	Active	Active

This viewgraph shows the average value of each development characteristic and the high and low values of the development characteristics from 12 projects in one class of application. The high or the low values themselves do not represent one project but show the most and least of any characteristic attributed to any of the 12 projects. The four projects that I will talk about are included in these statistics.

What is our development environment like? Our development teams design, implement, test, and document software that is scientific, ground-based, near-real-time, and interactive graphic. The software is 85 percent FORTRAN, 1 percent assembler, and 14 percent assembler macros. The assembler macros are required for the graphics capability. The software is developed on the IBM S/360-75 and -95, which are batch oriented with a timesharing option (TSO).

This is an operations environment, not a development environment. In this environment, the developers have access to the IBM S/360-95 via a Remote Job Processing (RJP) terminal and via TSO terminals. The developers use the IBM S/360-75 primarily in programmer-present blocks of time for integration and system testing via a graphics device. The IBM S/360-95 is the primary day-to-day satellite operations machine. When a hardware failure occurs, the developers lose access to the machine via the RJP and TSO terminals and must immediately relinquish their programmer-present time (if they have it) on the IBM S/360-75 so that operations activities can continue with minimal interruption. Since programmer-present blocktime is scheduled weekly and since the schedule is usually fully booked, IBM S/360-95 hardware failures always affect the development schedule adversely, especially late in the development life cycle.

In addition, the IBM S/360-75 is the primary satellite launch and launch-simulation operations machine. It is not unusual to have launches monthly, and frequently they are delayed on a day-by-day basis for 1 to 2 weeks or on a week-by-week basis for 2 to 4 weeks. When this happens, additional simulations are scheduled and/or additional mission planning machine time is required. Again, the developers must relinquish scheduled programmer-present blocktimes.

We estimate that 20 to 40 percent of scheduled programmer-present blocktime is lost because of hardware failures on both machines and because of launch delays. When frequent hardware failures and launches occur during the later stages of a development project, you can see how they can contribute significantly to the peak effort at the start of acceptance testing because of the need to make up lost machine time to complete the development project on schedule.

On the average, the development process takes 15.6 months, requires 8 staff-years of effort, develops 57,000 lines of code, and delivers 62,000 lines of code. Some amount of old code is used in each of these projects. The average staff size is 5.4 people and peaks at 10 people (full-time equivalents). Fourteen individuals are usually involved; this figure includes the first-line managers, i.e., those managers who make decisions, set priorities, and solve problems on a daily basis. For this application, on the average, the managers have 5.8 years of experience and the technical staff has 4 years. The technical staff includes the managers (approximately 30 percent). The managers have 10 years of professional experience overall, and the technical staff has 8.5 years of professional experience.

V&I EXPERIMENT

DOES INDEPENDENT V&I IMPROVE DEVELOPMENT PROCESS AND PRODUCT?

EXPECTATION	MEASURE
DECREASE	REQUIREMENTS AMBIGUITIES AND MISINTERPRETATIONS
DECREASE	DESIGN FLAWS
DECREASE	COST OF CORRECTING FAULTS
DECREASE	COST OF SYSTEM AND ACCEPTANCE TESTING
INCREASE	EARLY DISCOVERY OF FAULTS
INCREASE	QUALITY OF SOFTWARE PUT INTO OPERATION
MAINTAIN	PRODUCTIVITY/COST

174 SEL-(33)-4

Viewgraph 5: V&I Experiment

Why use a V&I methodology? It has often been claimed that the use of a V&I team would solve some of our problems. What we want to know from this experiment is "Does the use of an independent V&I team improve our development process and product?" To test this hypothesis, we will apply seven measures. These measures, however, are not completely independent of each other. They measure, in different ways, the occurrence of two basic properties:

1. When errors are discovered earlier, they are less costly to correct.
2. The use of a V&I methodology helps to discover errors earlier.

The seven measures with explanations follow.

1. Decrease requirement's ambiguities and misinterpretations. This will save time and money, especially in later stages of development. Overall, these are the most expensive errors to correct because requirements are the starting point for the development life cycle.

To evaluate this measure, the development error data that is collected by the SEL from the development and V&I teams from the start of implementation through the completion of acceptance testing will be examined. In this experiment, the use of a V&I methodology is not expected to reduce the development error rate; rather, it is expected to help discover errors earlier. If the use of a V&I methodology provides this benefit, a larger fraction of requirements errors will be detected during the design phase, in which the SEL has no formal process for recording errors, and therefore, fewer requirements errors (a smaller percentage of total errors) will remain to be discovered during the formal reporting period.[1] Compared with the past projects, a 50-percent decrease in the percentage of requirements errors reported by the development and V&I teams will be a clear indication of success for this measure. In addition, since the V&I team will pursue the resolution of unspecified and ambiguous requirements, fewer of these requirements problems are expected in the later stages of development.

2. Decrease design errors. This will save time and money in later stages of development. Design errors are the second most expensive to correct.

To evaluate this measure, the development error data will be used to compute the percentage of the design errors that are complex design errors. Complex design errors are many-component errors, whereas simple design errors are single-component errors. A component is a subroutine or shared block of code. Simple design errors are frequently related to (1) wrong assumptions about data values and structures, e.g., integer versus real variables, 2-byte versus 4-byte variables, location in buffer, or length of a format; (2) lapses in memory, e.g., missing items (declarations, dimensions, subscripts, statements, or counter incrementers) or incorrect variable names (not misspellings); or (3) incorrect interpretation of computations, e.g., wrong sense of direction (sign operator), factors of 2 or root 2, or wrong order of steps. Complex design errors are frequently

[1] Formal error reporting for development is keyed to machine-readable code that, in this environment, is the executable source code. Therefore, formal error reporting occurs only from the start of implementation through the completion of acceptance testing. Maintenance error data is collected from the maintenance group in a slightly different form.

related to interfaces and operational considerations and, therefore, they affect modules (several components). Since interfaces and operational aspects receive more scrutiny and high-level attention, they are more likely to be discovered during design reviews, which for the most part occur outside the formal error reporting period. The simple design errors, which are found in the detail of the design, are less likely to be found by a small V&I team (approximately 15 percent of development effort). If the use of a V&I methodology helps to discover complex design errors earlier, a larger fraction of the complex errors will be detected during the design phase, and therefore, fewer complex design errors (a smaller percentage) will remain to be discovered during the formal reporting period. Compared with the past projects, a 50-percent decrease in the percentage of complex design errors reported by the development and V&I teams will be a clear indication of success for this measure.

3. Decrease the cost of correcting errors. According to those who advocate the use of a V&I methodology and from our own SEL data, we know that correcting errors one life cycle phase earlier will produce a significant savings.

To evaluate this measure, the relative cost of correcting errors before and after acceptance testing started will be computed.[1] If the use of a V&I methodology reduces the cost of correcting errors, the developers will spend less effort per error in the later stages of development. Compared with the past projects, a 20- to 25-percent reduction

[1] Here, the relative cost of correcting errors is computed by tabulating the effort to correct errors (reported by the development teams) in each phase, computing the percentage of error-correction effort that occurred in each phase, and then dividing the error-correction effort percentage of each phase by the corresponding percentage of errors found in that phase.

in the relative cost of correcting errors after acceptance testing started will be a positive indication of success for this measure. Maintenance error data that is collected by the SEL from the maintenance groups will also be used.

4. Decrease the cost of system and acceptance testing. If the first three items occur, less effort will be required in these phases.

To evaluate this measure, the percentage of the development cost required to complete system and acceptance testing will be computed.[1] If the use of a V&I methodology helps to discover errors closer to the phase in which they originated, (1) the development teams will spend less time correcting errors during system testing and the system tests will be completed sooner, reducing the cost of system testing and (2) the development teams will need only to prepare for and to demonstrate the acceptance tests, reducing the cost of acceptance testing. Compared with the past projects, a smaller percentage of development cost for system and acceptance testing will be a positive indication of success for this measure. If the cost is less than the average cost for this application, it will be a clear indication of success.

5. Increase the early discovery of errors. This will save time and money in later stages of development as stated aoove. It will also improve the reliability of the software or at least improve confidence in the reliability of the software, since error rates will be less (or the mean time

[1] The development cost is computed by weighting the hours charged to a project by the different responsibilities of the personnel assigned to the project. A manager's hours are multiplied by 1.5; a programmer's hours are multiplied by 1.0; support service personnel's hours are multiplied by 0.5.

between failures will be greater) in the later stages of development. To evaluate this measure, the development and maintenance error data will be used to compute the percentages of errors that were discovered before and after acceptance testing started. If the use of a V&I methodology helps to discover errors earlier, most of the errors will be discovered before acceptance testing starts. Compared with the past projects, a 50-percent reduction in the percentage of errors discovered after acceptance testing started will be a clear indication of success for this measure.

6. Improve the quality of the software put into operation. This will decrease maintenance costs. In general, the use of a V&I methodology will be most beneficial in the M&O phase, since systems with lifetimes greater than 1 or 2 years usually have maintenance costs that range from 30 to 100 percent of the development cost.

To evaluate this measure, the software and maintenance error data will be used to compute the error rate for the M&O phase. If the use of a V&I methodology improves the quality of the software put into operation, the error rate in the M&O phase will be smaller compared with the error rates of the past projects. An error rate less than the average error rate (0.5 to 0.6 errors per thousand lines of developed code) for this application will be a positive indication of success for this measure.

7. Maintain productivity and cost. Adding another interaction for the development team will slow them down and will, therefore, reduce their productivity and increase the cost of development. However, if requirements and complex design errors are reduced, if the cost of correcting errors is reduced, and if the time spent on system and acceptance testing is reduced, those reductions should offset the cost of interaction between the development and V&I teams.

Therefore, productivity and development costs should remain the same. We do not expect to offset the cost of the V&I team completely, but optimistically speaking, we hope to.

To evaluate this measure, the software and the weighted work hours charged to the projects by the development teams will be used to compute (in staff-months) the cost of 1000 lines of developed code. A cost less than or equal to the average cost (1.7 staff-months per thousand lines of developed code) for this application will be a clear indication of success for this measure. That is to say, an average cost for the development team plus an added cost for the V&I team is a clear indication of success; the development teams will have maintained productivity despite the interaction with the V&I team.

By one calculation, the cost of interaction with the V&I team is estimated to be 10 percent of the development effort. Therefore, if the development teams are average in performance and require only the average cost even though they are interacting with a V&I team, the use of a V&I methodology will have effected approximately a 10-percent savings in development cost. If the use of a V&I methodology works well, i.e., if the first six measures show positive indications of success, then the combined cost of the development and V&I teams will be close to the average cost of development for this application. Since the cost of the V&I effort will be approximately 15 percent of the development effort and the estimated cost of interaction with the V&I teams is 10 percent, a combined cost of the development and V&I teams that is near the average development cost will indicate approximately a 25-percent savings in development cost (15 percent real savings).

V&I TEAM[1]

CHARTER:

VERIFY REQUIREMENTS AND DESIGN
PERFORM SEPARATE SYSTEM TESTING
VALIDATE CONSISTENCY END TO END
FIX NOTHING
REPORT ALL

PROCESS CHARACTERISTICS:

DURATION (MONTHS)	14-16
EFFORT	15-18 PERCENT OF DEVELOPMENT EFFORT
STAFF (FULL-TIME EQUIV.)	
AVERAGE	1.1
PEAK	3.0
INDIVIDUALS	6
APPLICATION EXPERIENCE	
MANAGERS	7
TECHNICAL STAFF	4
OVERALL EXPERIENCE	
MANAGERS	14
TECHNICAL STAFF	8

[1] SAME CONTRACTOR AS DEVELOPMENT TEAMS, BUT IN DIFFERENT OPERATIONAL AREA.

Viewgraph 6: V&I Team

What did we expect the V&I team to do in this experiment? The V&I team was supposed to

- Verify requirements and design
- Perform separate system testing
- Validate the consistency from start to end (from requirements to product)
- Fix nothing
- Report all findings

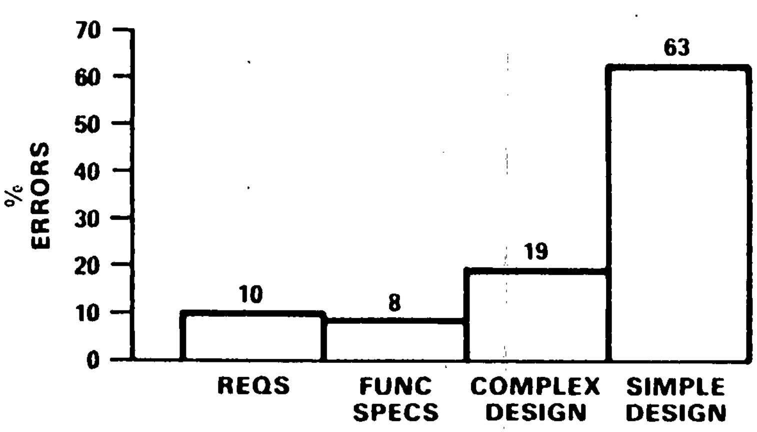
The V&I process lasted 14 to 16 months and required an effort of 16 to 18 percent of the development effort. The process required an average of 1.1 people and peaked at 3 people (full-time equivalents). Six individuals were involved, including the first-line managers. The application and overall experience of the technical staff was similar to that of the development teams (viewgraph 4); the managers, however, had a little more experience.

The V&I team was associated with the same contractor as the development teams but came from a different operational area.

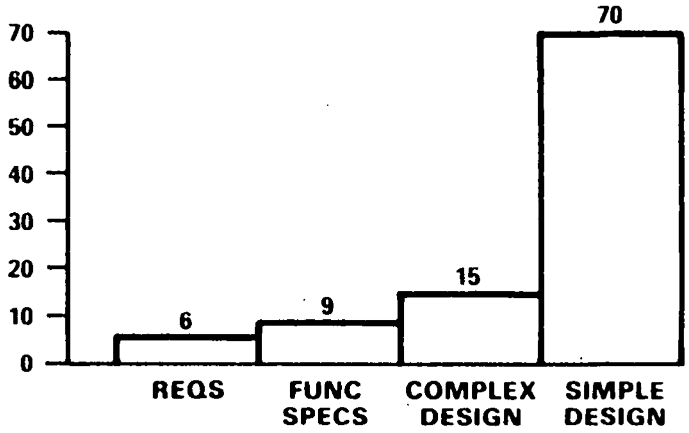
Next, we will examine the results of the experiment.

MEASURE 1—REQUIREMENTS PROBLEMS

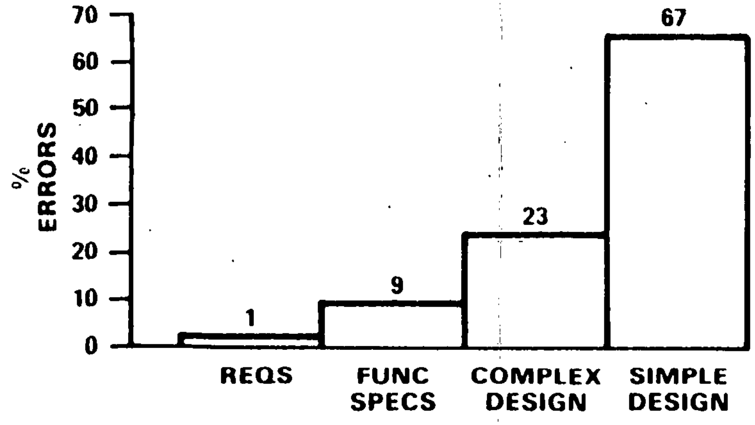
MEASURE 2—DESIGN FLAWS



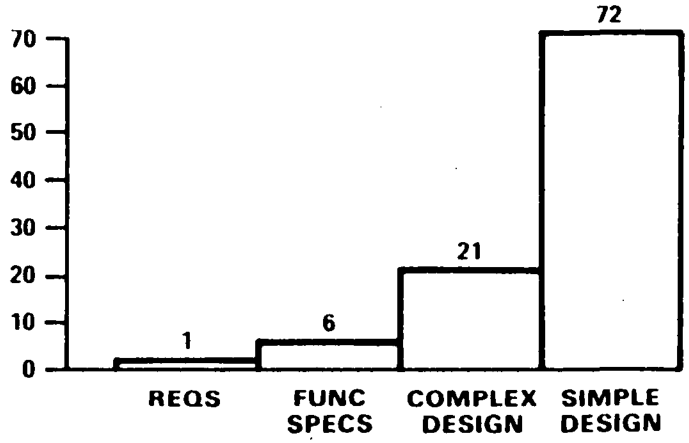
PAST 1



PAST 2



V&I 1



V&I 2

Viewgraph 7: Measure 1 - Requirements Problems and Measure 2 - Design Flaws

This viewgraph shows the breakdown, by percentages, of all the requirements and design errors detected from the start of implementation through the end of acceptance testing.

1. Requirements Errors

Expectation:

For requirements errors, we expect to see a 50-percent decrease in the percentage of requirements errors.

Findings:

From the bar graphs, you can see that the percentage of requirements errors for both V&I projects was reduced 84 to 90 percent compared with the past projects. In addition, very few requirements remained unspecified in the later stages of development. Hence, there were very few late surprises in terms of requirements problems compared with the past projects.

Conclusion:

The use of a V&I methodology did significantly decrease requirements ambiguities and misinterpretations.

2. Design Errors

Expectation:

For design errors, we expect to see a 50-percent decrease in the percentage of complex design errors. Complex design errors are those involving many components. Simple design errors are single-component errors. A component is a subroutine or a shared block of code.

Findings:

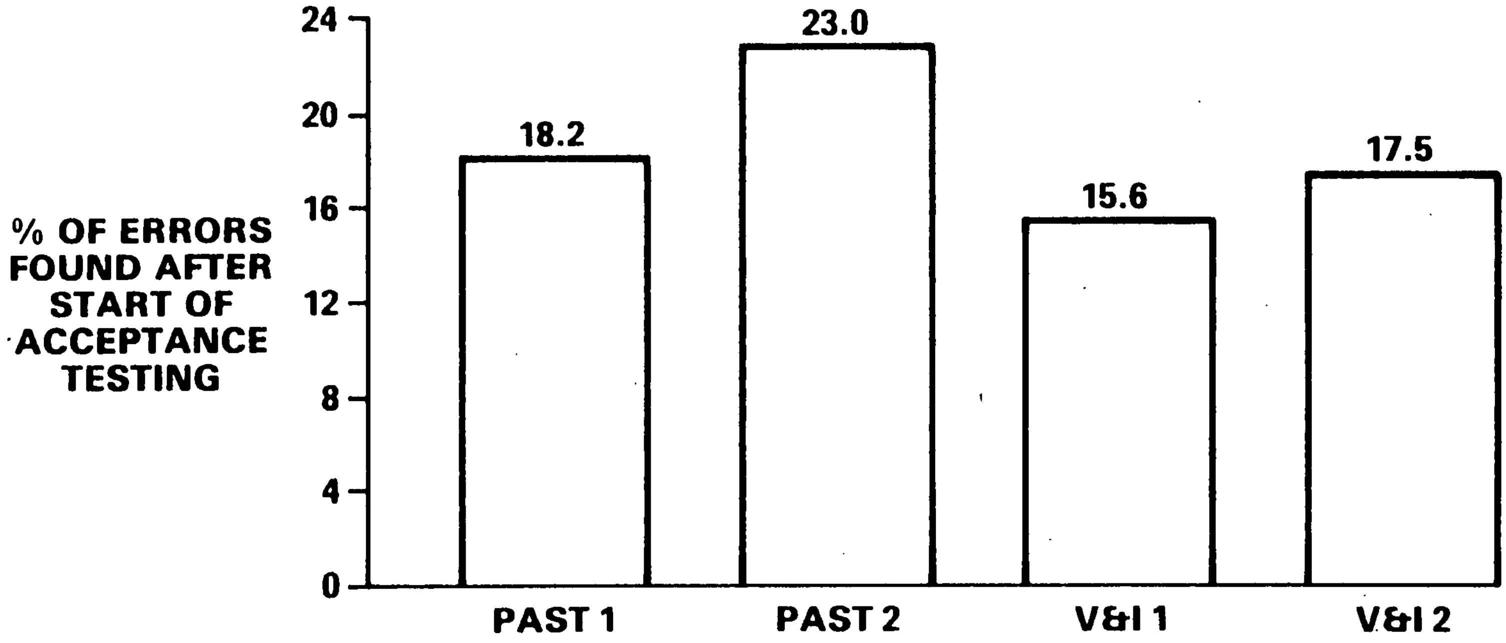
From the bar graphs, you can see that the percentages of complex design errors for the V&I projects are 26 and

23 percent of the total design errors. It is a little less for the two past projects (23 and 18 percent).

Conclusion:

The use of a V&I methodology did not decrease complex design errors.

MEASURE 5—EARLY DISCOVERY OF FAULTS



Viewgraph 8: Measure 5 - Early Discovery of Faults

This viewgraph shows the percentage of errors of the total that were found after acceptance testing started.

Expectation:

We expect to see a 50-percent reduction in the percentage of errors found after acceptance testing starts.

Findings:

You can see that for the two V&I projects there was a slight decrease (less than 30 percent) in the percentage of errors found after acceptance testing started.

Conclusion:

The use of a V&I methodology did not sigificantly increase the early discovery of errors.

Additional Data:

The percentage of errors found in each phase is as follows:

Phase	Past 1	Past 2	V&I 1	V&I 2
After Acceptance Testing Started	18.2	23.0	15.6	17.5
Before Acceptance Testing Started	81.8	77.0	84.4	82.5
Maintenance and Operation	3.4	5.3	5.0	6.9
Acceptance Testing	14.8	17.7	10.6	10.6
System Testing	14.8	4.8	8.2	18.9
Code/Unit Testing	67.0	72.2	76.2	63.6

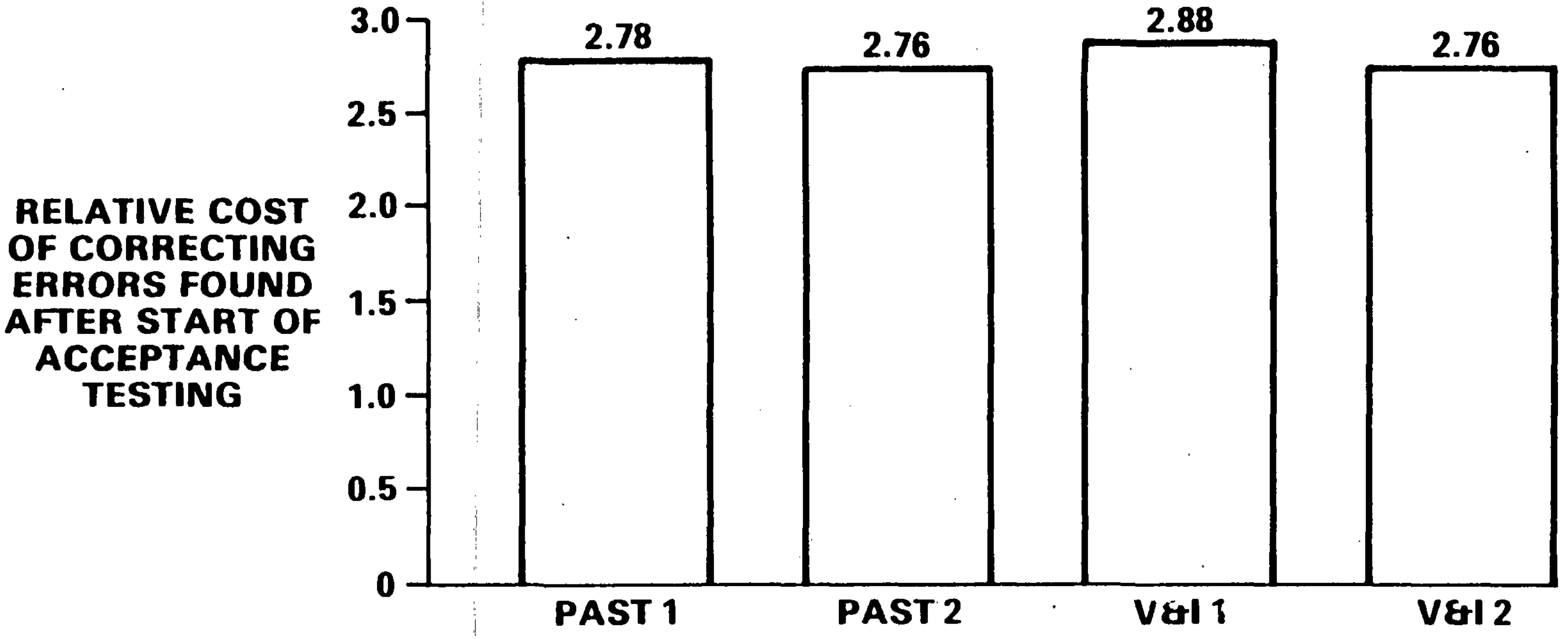
This viewgraph and viewgraphs 9 through 11 contain M&O data through November 20, 1981. The length and status of the M&O phases are as follows:

M&O Phase	Past 1	Past 2	V&I 1	V&I 2
Months	25	14	5	6
Status	Active	Complete	Active	Active

Except for project Past 2, which has ended, the results presented in viewgraphs 8 through 11 can only become worse with further operation. However, the results are not expected to change appreciably because of the characteristics of the environment. Typically, in this environment, 95 to 100 percent of the postacceptance error corrections and enhancements occur during the first 6 months of M&O. For example, the supposedly last-planned modification of the source code for both V&I projects occurred a few days before November 20, 1981.

After the first 6 months of M&O, typically, the software is changed only to support a degradation in satellite hardware performance, e.g., failure of a primary sensor. However, to support a launch, the software is engineered to support these types of contingencies but not always accurately enough for day-to-day operation. Since the usual lifetimes of these projects range from 1 to 3 years, the users must weigh the cost of extensive development to support serious or critical degradation in satellite hardware performance with the benefit to be gained during the expected (and usually shortened) life of the satellite. For example, about a year ago, the satellite of project Past 1 (25 months M&O) had a critical hardware failure that seemed to end the project prematurely; however, relatively simple modifications to the software allowed the users to keep the satellite active in a degraded mode of operation.

MEASURE 3—COST OF CORRECTING FLAWS



J. Page
CSC
31 of 47

Viewgraph 9: Measure 3 - Cost of Correcting Flaws

This viewgraph shows the relative cost of correcting errors found after acceptance testing started. This number is the ratio of the fraction of effort required to correct the errors that occurred after acceptance testing started to the fraction of errors that occurred after acceptance testing started. For example, if 50 percent of the effort to correct errors was expended after acceptance testing started and if that effort was needed to correct 5 percent of the errors, this number would be 10.

Expectation:

We expect to see a 20- to 25-percent lower relative cost to correct errors after acceptance testing starts.

Findings:

From the bar graphs, you can see that the relative cost to correct errors after acceptance testing started was the same as that for the past projects. The relative cost to correct errors before acceptance testing started was approximately 0.5. This indicates that the cost to correct errors after acceptance testing started was between 4.4 and 4.9 times more costly than the cost to correct errors before acceptance testing started.

Conclusion:

The use of a V&I methodology did not decrease the cost of correcting errors in the acceptance testing and M&O phases combined.

Additional Data:

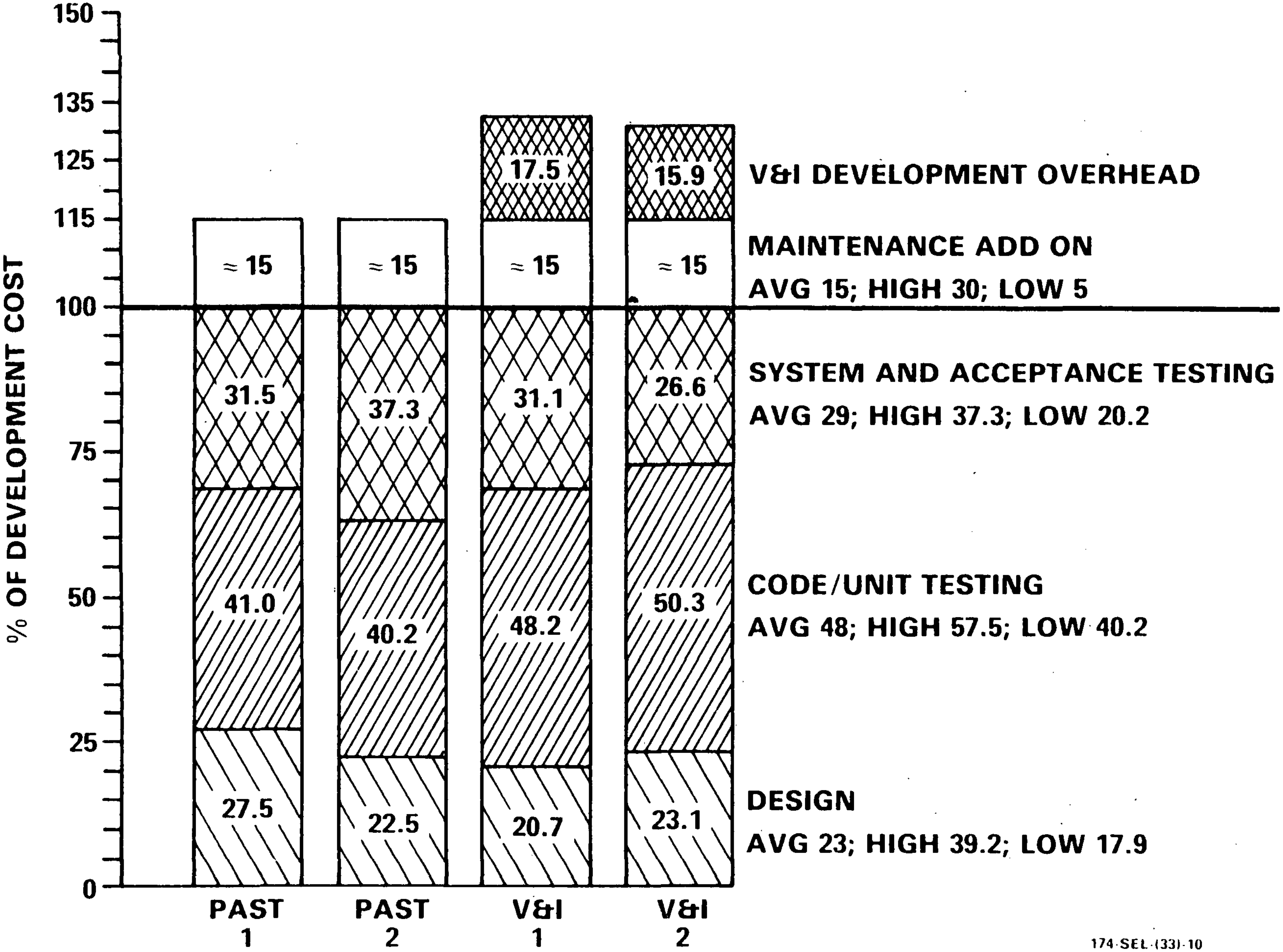
The relative cost of correcting errors in each phase is as follows:

Phase	Past 1	Past 2	V&I 1	V&I 2
After Acceptance Testing Started	2.78	2.76	2.88	2.76
Before Acceptance Testing Started	0.60	0.47	0.59	0.63
Maintenance and Operation	4.85	4.53	4.09	3.54
Acceptance Testing	2.31	2.23	2.31	2.26
System Testing	1.00	1.09	1.30	1.08
Code/Unit Testing	0.47	0.43	0.58	0.49

These figures, in part, validate the common belief (advanced by proponents of V&I methodology) that errors are more expensive to correct when they are discovered later in the development cycle. You can also see from these figures and from the figures in the previous viewgraph that the results are different for different phases; but, remember that we do not have responsibility for the maintenance phase, and data is more difficult to obtain from the group who has responsibility. Therefore, we measure things one or two phases earlier, i.e., during acceptance testing or system testing.

The relative cost of correcting errors in the M&O phase was less for the V&I projects mainly because of fewer requirements errors in that phase. The past projects had at least twice as many requirements errors in that phase.

MEASURE 4—COST OF SYSTEM AND ACCEPTANCE TESTING



J. Page
CSC
34 of 47

Viewgraph 10: Measure 4 - Cost of System and Acceptance Testing

This viewgraph shows the cost for time spent in various development calendar phases (not activity phases). Design activity takes place in the design calendar phase, in the code/unit testing (implementation) calendar phase, and even in the system and acceptance testing calendar phase. Detailed SEL data shows that design activity ranges from 30 to 45 percent of the development effort. On the average, however, only 23 percent of the development effort occurs during the design calendar phase, i.e., the phase in which only design-related activity is performed. The remaining design activity is performed primarily during the implementation phase because requirements change, previously missing information is acquired, and design errors exist. Since it is not unusual to receive requirements changes during the system and acceptance testing phases, since some previously missing information may be acquired during these phases, and since design errors are also discovered in these phases, some design activity occurs here, too.

This viewgraph also contains the average cost for each phase and the highest and lowest cost for each phase for the 12 projects in our sample. The high or low costs themselves do not represent the cost of one project but show the most and least money spent for the various phases by any of the 12 projects.

Expectation:

We expect to see a reduction in the cost of the system testing and acceptance testing phases.

Findings:

On the average, we spend 29 percent of our dollars on system and acceptance testing. You can see that one V&I project was below the average (26.6 percent) and the other, above

(31.1 percent). Together, they were equal to the average. Both were less than our two projects from the past.

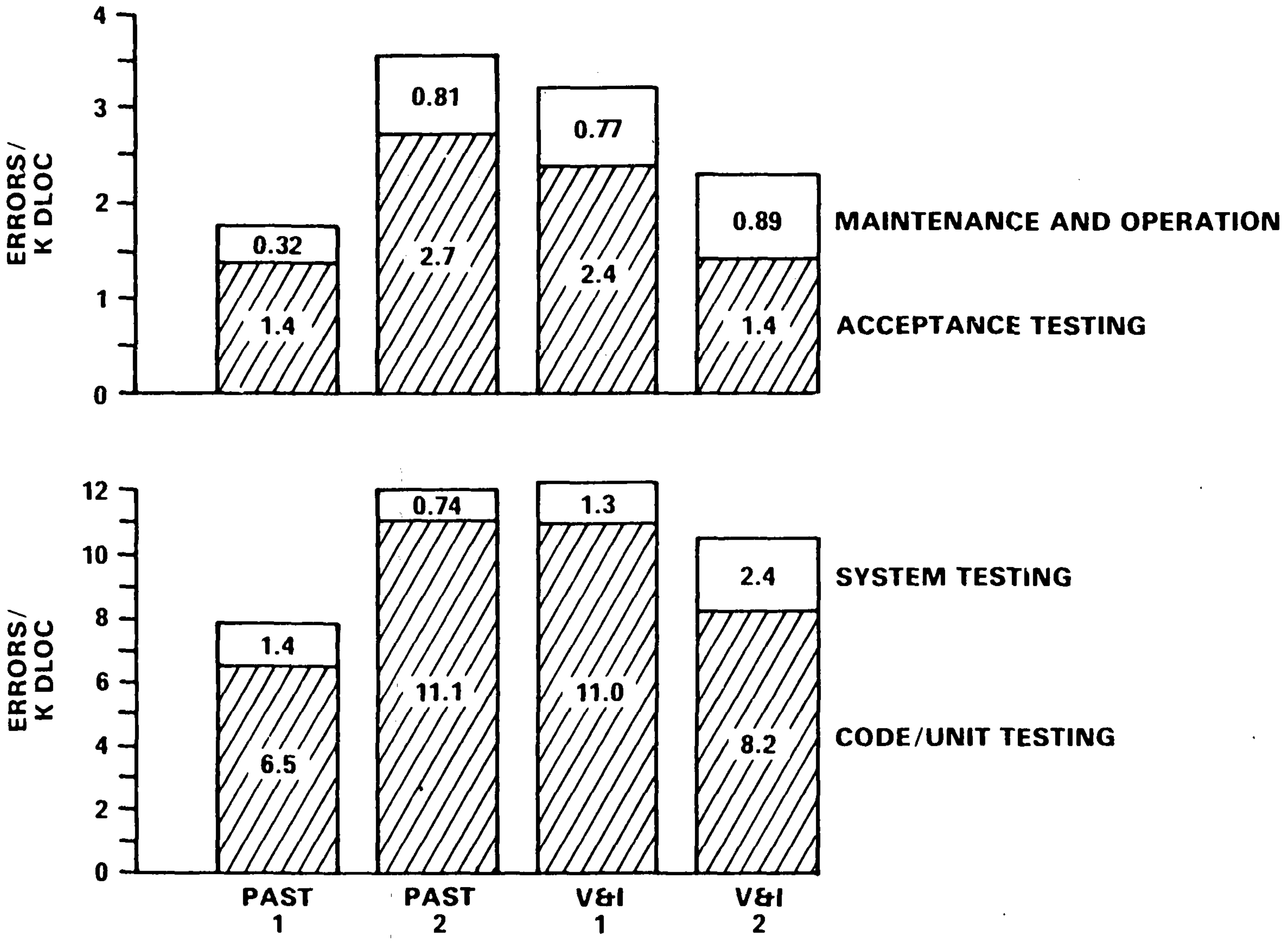
Conclusion:

The use of a V&I methodology did not significantly decrease the cost of system and acceptance testing.

Additional Data:

We do not have responsibility for the maintenance phase. Our best estimate is that the maintenance costs for the four projects are about 15 percent of the development costs. The V&I projects had approximately 16- to 18-percent overheads to pay for the V&I effort.

MEASURE 6—QUALITY OF SOFTWARE



MONTHS	PAST 1	PAST 2	V&I 1	V&I 2
MAINTENANCE	25	14	5	6
OPERATION	21	12	3	3

Viewgraph 11: Measure 6 - Quality of Software

This viewgraph shows the errors per thousand lines of developed code for various calendar phases. What is important here is the M&O phase.

Expectation:

We expect to see an error rate in the M&O phase less than the average error rate for this application.

Findings:

From the bar graphs, you can see that the error rates for the two V&I projects are not better than the error rates for the two past projects. The average error rate in the M&O phase is between 0.5 and 0.6 errors per thousand lines of developed code; both V&I projects had error rates higher than the average.

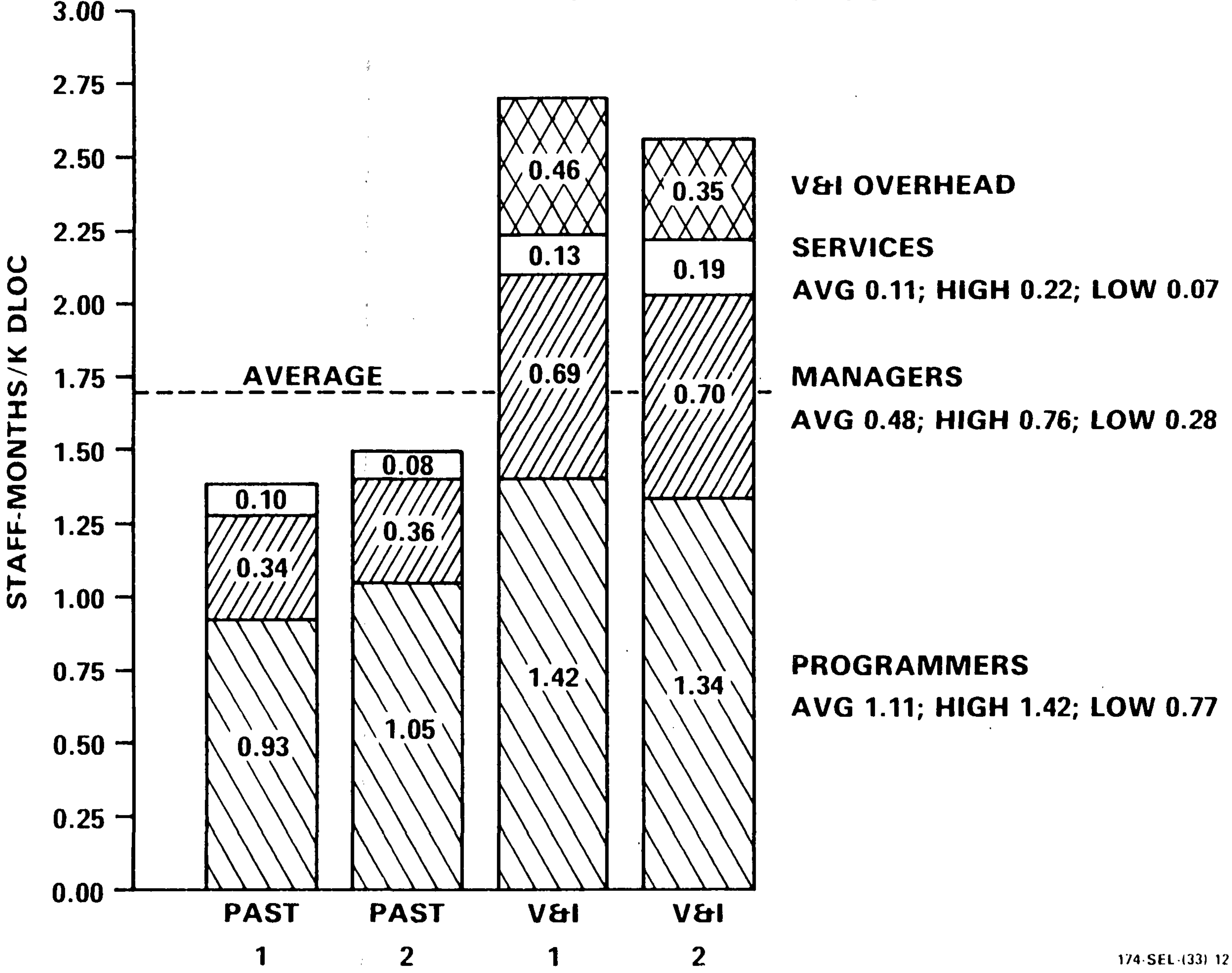
Conclusion:

The use of a V&I methodology did not improve the quality of the software put into operation.

Additional Data:

Error rates from the other phases are important track records. Hypothetically, let us say that projects Past 1 and V&I 2 were developing the same product. If we measured the acceptance testing error rates, we would see that both had error rates of 1.4 errors per thousand lines of developed code. We would not be able to tell too much about the projects from that viewpoint. However, if we examined those projects' error rates before acceptance testing, we would see that project Past 1 had a preacceptance testing error rate of 7.9 and project V&I 2 had a preacceptance testing error rate of 10.6. From this, we may be able to predict the worse M&O phase error rate for project V&I 2.

MEASURE 7—PRODUCTIVITY/COST



174-SEL-(33) 12

Viewgraph 12: Measure 7 - Productivity/Cost

This viewgraph shows the cost (in staff-months) per thousand lines of developed code (K DLOC).

Expectation:

We expect the V&I overhead costs to be an add-on cost to our average development cost.

Findings:

Because of the interaction with the V&I team and some other problems, we drove the productivity of the development teams to the low end of our productivity range. Together, the two V&I projects were about 85 percent more expensive than our two past projects. Since the quality of the products was not any better (see viewgraph 11), an 85-percent increase in cost for the same product is a very expensive penalty to pay. The cost of the development part of the V&I projects (2.2 staff-months per K DLOC) was approximately 30 percent higher than the average development cost (1.7 staff-months per K DLOC). This is three times as large as the estimated cost of interaction with the V&I team.

Conclusion:

The use of a V&I methodology is expensive.

RESULTS OF V&I EXPERIMENT

FROM THE DATA WE HAVE USED, WE HAVE

FOUND	MEASURE
LARGE DECREASE IN	REQUIREMENTS AMBIGUITIES AND MISINTERPRETATIONS
NO DECREASE IN	DESIGN FLAWS
NO DECREASE IN	COST OF CORRECTING FLAWS
SMALL DECREASE IN	COST OF SYSTEM AND ACCEPTANCE TESTING
SMALL INCREASE IN	EARLY DISCOVERY OF FAULTS
NO INCREASE IN	QUALITY OF SOFTWARE PUT INTO OPERATION
LARGE DECREASE IN	PRODUCTIVITY
INCREASE IN	COST

SCORE: 1 PLUS; 5 ZEROS; 1 DOUBLE MINUS

Viewgraph 13: Results of V&I Experiment

From the data we have used, which includes resource data, error data, and the software, we have found that a V&I methodology provided

1. A large decrease in requirements ambiguities and misinterpretations. There were very few late surprises in terms of requirements problems, and the number of requirements errors reported was significantly less than for the past projects.

2. No decrease in design errors. The fraction of complex design errors was similar to that of the past projects.

3. No decrease in the cost of correcting errors. The relative cost of correcting errors that occurred after acceptance testing started was the same as that for the past projects.

4. A small decrease in the cost of system and acceptance testing. One V&I project had a system and acceptance testing cost less than the average system and acceptance testing cost; the other V&I project was above the average cost. However, both V&I projects had costs below the costs of the past projects used in the comparison.

5. A small increase in early discovery of errors. For both V&I projects, the percentage of errors that occurred after acceptance testing started was less than the percentage of errors that occurred after acceptance testing started for the past projects.

6. No improvement in the quality of software put into operation. The error rates in the M&O phase for both V&I projects were higher than the average error rate for software put into operation for this class of application.

7. A decrease in productivity and an increase in cost. Because, in part, the interaction of the V&I and

development teams lowered productivity and because there was not a savings in correcting errors, the cost was high.

We scored a plus with the first measure (requirements problems); zero with the next five measures; and a double minus with the last measure (productivity/cost).

SUMMARY

- FIRST APPLICATION OF V&I IN THIS ENVIRONMENT
  - DID NOT IMPROVE PROCESS
  - WAS EXPENSIVE
  - WAS A MANAGEMENT HEADACHE
- HOWEVER, WITH VARIATIONS, WE WILL ENCOURAGE ITS USE FOR
  - THE RIGHT SIZE EFFORT
  - THE RIGHT RELIABILITY REQUIREMENT

Viewgraph 14: Summary

For our first application of a V&I methodology in this environment

- V&I did not improve the process
- V&I was very expensive
- V&I was a management headache

To qualify this, our experience with many methodologies has been as follows:

- The first time a methodology is applied, mistakes are made (and we made many mistakes), and many of the potential benefits or advantages of the methodology are not realized.
- The second time a methodology is applied, there is a tendency to overcompensate for the things that you did worst the first time, and the methodology still does not work as well as it potentially could.
- The third time a methodology is applied, you lower your expectations somewhat or modify them, and you home in on what is right for your environment.

In general, development teams are at the bottom of the totem pole in this environment. Because they work in an operations environment, they have low priority for accessing the machines. They have adversary relationships with the analysis/requirements team, the team that conducts acceptance testing, the people who schedule computer time, the computer operators, the programmer assistance center, and the customer. The V&I team members, who are like a development team but do not design or implement, have the same adversaries. Placing a V&I team in this environment creates another adversary for both the development team and the development-like V&I team. The manager who monitors both teams (the customer) has twice as many complaints, computer problems, priority decisions, schedule problems, cost problems, reporting problems, and conflicts to deal with. The V&I experiment was a management headache.

However, we believe that we know what changes are needed and how to moderate them to make the use of a V&I methodology more cost effective in this environment for

- The right size effort
- The right reliability requirement

Most of our projects require 8±4 staff-years of effort. We believe that a V&I methodology will be cost effective in the 10- to 12-staff-year range and that cost savings will be achieved for larger efforts. All our completed projects have been for ground-based software, but we have started to develop some onboard (flight) prototype systems. For these systems, which have a more stringent reliability requirement, we believe that a V&I methodology will be cost effective for 5- to 6-staff-year efforts. In both these cases, we believe that a V&I effort of approximately 15 percent of the development effort is sufficient for our work.