Software Inspections
a
Case Study
by
Ronald A. Radice

Abstract:

This paper is addressed to individuals who are familiar with formal software Inspections. A case study is presented to demonstrate how data resulting from software Inspections can be used to determine the quality of a product under development and to get management to take actions to improve the quality of a product before delivery to a customer.

The case study is based on software development performed on a subsystem that is part of three software operating systems. The data has been masked to prevent identification of the specific product. The case study provides three releases of product data across the traditional life cycle from requirements through systems test. Inspections were used in all activities producing work products prior to the beginning of unit test. Data is also shown for levels of test; i.e., unit test, function test, component test, and system test, and for the projected remaining defects at the time of delivery to the marketplace.

The data is analyzed and it is shown that the release of the product under development will not meet expected quality objectives at the time of delivery. Initially the reasons for this are not apparent, but the data reveals a compelling story which gets the attention of management. A study is set up performing various statistical and data analysis to learn how deep and pervasive the problem may be. It is decided to run a sample set of re-inspections with modules that are already in function test to understand the latent defect density in the product. It is shown how a reasonable sample of the product was chosen for re-inspection to ensure conclusions that represented the product as an entirety and not only within the samples chosen.

After the sample inspections have been proven to be sufficiently representative, it is shown how the latent defect density is calculated. Then it is shown how the product development team took action to physically remove defects before allowing the product to enter system test. The cause of the quality problem is derived and it is shown that the data had integrity and was consistently interpreted. It is also noted that the product under development not only delivered on schedule, but at quality objectives after the extra latent defects were removed. Various data is provided in this case study to permit the reader to come to the same conclusion that the process organization reached.

Preamble:

This paper was originally submitted to an international conference for software engineering in the early 1980's based on the results of software projects during 1976-1980 in a large computer company in the USA. The paper was not accepted at that time. This case study has been used during training for both software Inspections and metrics. The author believes the case study demonstrates interesting information regarding data analysis and control of quality goals by way of example for other practitioners in software still today in year 2000.

Introduction:

The author has three purposes for this paper. The primary purpose is to demonstrate statistical and/or data analysis as applied in a software Inspection environment can be performed with less than the best of

conditions. The secondary purpose is to represent the software development environment as a system wherein the system output is determinable not only by the system input but by the characteristics of the development system environment. The third purpose is to add to the empirical basis of software engineering through the use of the case study approach.

In this case study the system we explore is an actual development environment for a software product, the input is the history of previous software developments within the same proximate environment, the system characteristics are represented partially through the data resulting from the Inspection process, and the output is the quality of the product as defined in defects/KLOC.

The case study format was chosen since it lends itself to discussing this instance of the use of data with statistical and data analysis methods to project quality goals and to control them dynamically during the development process to manage quality goals for a product. Admittedly, more is possible than this case study demonstrates, but we must remember the analysis was made in 1979. Today we can and are doing more. Finally, it is hoped that others will use the case study approach to advance software engineering development and education.

As the engineering of software continues to move along the vector from a highly skilled craft to a repeatable discipline, data about the environment and the system in which the engineering occurs becomes essential for software engineers to learn the vital data factors that determine a well engineered product. We should want this data for two primary reasons. First, to be able to compare the work between projects to learn which is quantifiably better and then through analysis learn why it is better. Second, to be able to commit and manage improvements in quality, productivity, schedules, and costs in future software projects. Without data we cannot do either in any approximation to a discipline and certainly we cannot be in engineering or business control of the products we produce.

This paper shows that with a minimal set of data we can begin to attack both the need to compare and the need to commit and manage improvements.

It is not intended through this paper to suggest that the set of variables enumerated is a complete and fully sufficient set of data that needs to be gathered. However, it is a set which demonstrates that we can start to progress with what is already available in our software environments. Over time, the data we now have available will expand as the need for delivering better products will not cease. The data for the projects in this paper was gathered during the late 1970's, and because it had proven beneficial these products had gone on to gather more detailed data and other data. The minimal set of data of the past becomes the foundation for the future improvements.

Jay W. Forrester, who's work was a primary influence on my thinking during this case study time frame, has said, "Gathering data is not a science; it is, however, the first step towards a science evolving from an art."

The Environment:

When we look at the environment within which software is produced, we note a few major aspects which mutually intersect and which have an effect on how well the other aspects will contribute to a successful software product solution. These aspects are shown in Figure 1, and include the following: people, technology resources including software and hardware, processes including information of various types, and funding,. Jay W. Forrester of MIT when writing of systems dynamics calls this set Men, Machines, Materials, and Money, all four of which are common to all systems. The interaction of these aspects defines how well the system performs and how well it meets its objectives. In this case study the system is the environment in which software is produced.

THE SOFTWARE PRODUCTION ENVIRONMENT

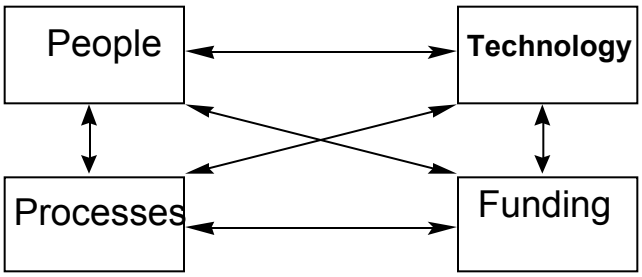

FIGURE 1

The Ecology Of Software:

In the world of software production, just as in the world at large, everything seems to affect everything else to some degree. This means that in the environmental relationships shown in Figure 1 each of these aspects can have an affect on each of the others through interaction. It also suggests that in the more granular view of the system of software production that there may be data variables which represent how these aspects affect each other. The problem which we must solve and which this case study partially explores is which are the few interesting and vital data variables and how might we begin to understand the relationships among them.

Systems:

What is a system? There are two types of systems; open and closed. An open system has a starting state, has "sets of elements standing in interrelations" [BER68], has states which change, and has a logical potential end which can be achieved "from different initial conditions and in different ways." [BER68] The end can be the completion and termination of the system itself or it can be the delivery of some end product from the system to another system or the evolution of the system to another system. Figure 2 shows some differences between closed and open systems.

CLOSED AND OPEN SYSTEMS

CLOSED	OPEN
-isolated from its environment -final state is unequivocally determined by the initial conditions -"if either the initial conditions or the process is altered, the final state will be changed" [BER68]	-has a starting state determined by its environment -"final state may be reached from different initial conditions and in different ways" [BER68]

FIGURE 2

Examples of systems include a seed, an embryo, an infant, a man, a developing third world country, a community, a country, and a software project. Each of these systems have three characteristics in common. First, they all go through stages of development, where succeeding stages are dependent on the completion of previous stages, and where each stage has a dependency on other interacting elements. Second, an observer can reasonably predict the final attributes of each system by observing its developmental stages and by understanding the affect and relationship of vital parameters within the environment. Third, an observer can mold the results by treating symptoms, reacting to pathologies, modifying interacting

elements, restoring the proper environment for development, or removing a caustic element in the environment, among other interventions

The Data Variables:

Is there a minimal set of data variables that we can use on all projects? What are the minimal set of data variables that are available today in probably all software environments, and which should be gathered into process databases for software products? There are basically six:[1]

Lines of Code:

There are many arguments regarding the value of Lines of Coda as a data variable. The arguments will not be extended or discussed in this paper; enough has already been debated on that subject. Furthermore it has been shown that alternate metrics such as Halstead's Volume, Function/Feature Points, and McCabe's Cyclomatic Number correlate to Lines of Code. Thus all these proposed alternatives seem to be only considered or opinionated alternatives to the admittedly weak measure of Lines of Code. For the time being I propose we use what is readily and easily available. Therefore, for our first variable we shall capture Lines of Code, which for this paper means specifically source lines of code (excluding commentary) added and modified for the product in question.

Defects:

The next variable we shall insist on capturing is defects. This is an admission that we as engineers do unfortunately create defects while we create programs. It is not apparent that this will ever be otherwise, so let us acknowledge that defects do result during the development of software. Given this admission, we should then begin capturing at a minimum the number of such defects as they are discovered in Inspections, reviews, tests, or by users. Defects are any error which would cause the program to work other than desired; i.e., a deviation from the product requirements statement or specifications.

Engineer Months:

The third important variable is the time spent by the engineers to create the product. Let us call this parameter Engineer Months. It should include all personnel months from requirements through delivering the product to the user. Additionally, we should be interested in the effort spent in maintaining the program once it is delivered.

Calendar Months:

If we wish to understand how well we might do on our next project, we should capture calendar time spent in producing software. This is different from Engineer Months in that we wish to measure linear time for the entire schedule used to develop the project, including all activities within the project. At a minimum this should include all time from the day we begin to work with the user requirements to the day we deliver the program to the first users.

Cost:

As we evolve into a repeatable business discipline we should know how much it has cost to develop products. This means that we must capture a variable such as dollars. Although this is correlated to Engineer Months, it is, in fact, different as it includes dollars for machine resources, travel, overhead, and other resources necessary to produce and deliver software.

Test Progress:

The last variable that we will discuss is Test Progress. Here we are interested in how well our testing efforts are proceeding. We can measure it in different forms: the number of test units, a normalized value for each test unit, or some other form of measure. Rather than try to settle which is the better way to

measure test progress, let us agree that the more important thing is to indeed measure the test effort and progress, especially since test remains such a high cost for most projects.

There are other factors that are of interest, but with just these six we can do much to bring software environments under control and influence the resultant quality of a product. With these six variables we can make major advances in managing software projects. We will make use of only three in this case study; Lines of Code, Defects, and Test Progress.

History Repeats Itself:

If a engineer does not learn from previous mistakes, if a project manager does not have the benefit of previous project management knowledge or does not learn from previous experiences in management, if new engineers join an organization without sufficient industrial perspectives, then similar mistakes in project development will be made, and *History Will Repeat Itself*. We have all seen examples of this phenomenon where the same mistakes are being made again that were made in other projects.

Given, then, that in our world of software engineering, history will repeat itself, we must additionally understand that in our environment, just as in any system environment, inertia prevails and the path of least resistance will be usually taken. Basically everything in the system stays the same unless differences for choice can be demonstrably shown. This means, for example: that people will make the same mistakes unless they choose to do something to change the cause of the mistakes; that project estimates in Lines of Code delivered or resources expended will continue to be faulty, unless feedback is maintained between goals and actuals; that if we continue to dig in the same hole for a solution which cannot be found there, that we will not change that fact; that approximately the same density of defects injected during a programming development project will be evident in another project where the environment has not changed; i.e., where the people and the tools they have to work with, the resources that are made available to them, the methodologies they use, the processes they use, and the practices they employ are basically the same as they were before. Therefore, as we set goals for improvement, we must address the areas to change in the environment which will affect the goals we have set.

The Case Study: An Example From A Series Of Projects:

This case study is from a project which was under development during 1978-80. This project had a body of data across three prior releases which included defects, Lines of Code, engineer months, calendar time, cost in dollars, and test progress. All of the referent projects were developed in similar environments. That is, the process, tools, people, and practices were basically the same across the releases. There certainly were degrees of differences, but there was nothing which was apparently different in any significant way. The project in this case study was at the time of this evaluation about two thirds complete with its Functional Verification Test (FVT) on the first operating system of three to be delivered. FVT is a test after Unit Test and prior to component and system testing. The engineering project is considered complete when system test is finished.

THE PROBLEM:

The project under development had consistently higher error rates being reported through all of the life-cycle defect removal activities. See Figure 3 for a comparison of three product releases. There was no reason to believe that the increase in the error rate was about to decline in the remaining tests. Yet, the management and project team continued as if there were no problem and none was to be encountered during remaining tests. The Component and System Tests, which had not yet started, were viewed by our small process group as potentially completing 6-8 weeks later than planned with a questionable quality level. The density of defects removed during FVTwas already almost equal to that found in prior releases, and this was only the first of three FVTs to be executed for this product. The management jokingly suggested that more defects were being found because the Inspection process FVT were finally working better, so “What’s

the problem?", they asked. It was obvious that the final point for the three FVTs on this curve of actual defect data could only go higher, since they were not yet complete, but we did not known how much higher. The question to be resolved was whether the project team was actually doing a better job of finding defects earlier or if there were a significant potential quality and/or schedule impact waiting to be encountered.

THREE RELEASES OF DEFECT REMOVAL DATA ACROSS STAGES

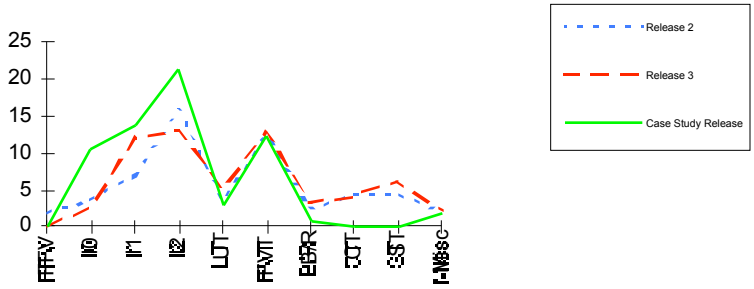

FIGURE 3

THE ANALYSIS:

How should one proceed to determine if indeed there were going to be any impact based on the information available? After all, it was possible that the development team had done a more effective job of removing defects earlier. In turn should this not lead to a better quality product in Component and System Tests and when it is delivered? The problem with this assumption is that to demonstrate that the Inspections were more effective in this release we should have been able to find some action which made them more effective; something that was sufficiently different in this project's environment. After interviewing the managers and the engineers on this project nothing could be shown to be different in the way the Inspections were being performed in this release. In fact the data suggested quite the contrary.

Therefore, we began to try to understand what the available data might be telling us. We attacked the data from different views and analysis to help ensure we were not building incorrect conclusions.

Inspection Effectiveness View:

The overall Inspection effectiveness for the three previous releases was 59% (release 2), 56% (release 3), and 41% (release 5). This was demonstrated by the actual results from the earlier three releases. These analysis show a diminishing effectiveness over the three releases through time. This was partially due to two reasons: 1) the focus to keep the Inspection effectiveness as high as 59% was assumed and had been removed, i.e., the process group was not in operation after the first release (actually the first release was as high as 65% for the parts inspected), and 2) the feedback of the Inspection data to management and the project team members was minimal, therefore they just did not know. We had been fortunate to be able to reform the process group for these products during the FVT period for the release in this case study.[2]

In so doing we were able to look at the data which had been accumulating. While there was no existent process group, the Inspection reports were still being completed and the data entered into the process database. Looking at the Inspection reports across the four releases we believed that the data which was available had a reasonable degree of integrity and could be used for comparisons. We ran some sample checks and found nothing to cause us to throw out the data. We made an assumption that the effectiveness had stabilized for this project at the 41% level; i.e., it was equivalent to that on the previous project. If this were true 2200 defects (59%) would be found in all the test stages from Unit Test through System Test across the three operating systems. Since 546 defects had already been found in FVT for the first operating system, this left 1654 defects that could still potentially be found prior to full product delivery. These numbers assumed an equivalent level of quality at delivery. We discounted UT found defects since they showed rough equivalence and there was noise in the data. These 2200 defects would be spread across three different operating system environments.

Three additional questions were asked of the data before proceeding:

One, were the defect types or distribution in this release significantly different from those found in the previous release? The analysis showed no.

Two, were more defects being found in the base code (prior releases)? No, in previous releases and in this case study release 98% of the defects were found in the new or changed code for the release under development.

Three, was the Function Test finding fewer defects due to reduced leakage of the Inspection process; i.e. were the Inspections more effective? Analysis showed that 43.1% of the defects being found in the Function Test could have been found by the Inspection process for the case study release. This was an independent evaluation made against every third error found in the Function Test. A similar evaluation had previously been made in release 2 of this product and showed that approximately 40% of the errors found in the Function Test could have been found by the Inspection process.

The conclusion drawn from this analysis was that the Function Test for this release was finding the same mix of problem types (Figure 4) and that the Inspection process effectiveness had not substantially changed, therefore, the projected number of defects to be found (1654) in the remaining tests became one boundary point to be used in assessing the impact, if any.

DEFECT TYPES

LOGIC 30%
DESIGN 14%
REGISTER USAGE 14%
ADDRESSABILlIY 11%
INTERFACE 11%
CB's/MACROS 7%
LANGUAGE 5%
MISC 5%
BAD FIXES 2%
MESSAGES 1 %

FIGURE 4

To derive an impact statement, we needed to understand what the potential disruption would be to the test stages after Function Test, which was approximately 70% completed at the time of this study for the first operating system of three supported by this project subsystem; i.e., 70% of the test cases had been executed successfully. The completion percentage was approximated through a measure of the code that had been tested and the test cases remaining to be executed. Since 511 defects had been found solely in FVT, then at 70% complete there would be another 219 defects found by the time FVT was finished. A sanity check was taken on this projection. The FVT was finding on average 30 defects a week, and as it was evident that seven weeks remained before testing would be complete in FVT, therefore 210 defects could potentially be found. These two simple projections were close enough in number. Of the 1654 projected defects for all tests, 210 would be found in the first FVT leaving 1444 defects to be found in the remaining test stages and defect removal activities for the operating systems 1, 2 , and 3. When we assessed what the test departments were prepared to handle based on history, staffing, and time in their schedules, this 1444 represented a potential 200% impact.

Trend View:

We felt we were working with data that had a trustable level of integrity, but which was less than desired. We believed that we needed another boundary point that was independently generated to help us understand the impact. Therefore, we took a straight forward trend view. To date the Inspections had apparently found 62% more errors than in the previous release, and since we had determined that the first FVT would find somewhere between 56% and 73% more errors than in previous releases, then we might assume that the tests after FVT would also find approximately 60% more defects, or 689 errors. This was certainly a lower number than the 1444 approximated by the effectiveness view, but it identified a 60% impact nonetheless. Furthermore, if this were a valid boundary point, then it made a tacit assumption that the Inspections in this release were 51.4% effective. This, however, was highly questionable, as was shown earlier. Therefore, it was judged as less likely. Nonetheless, we now had a boundary of impact from 60% to 200%. In either case, there was an impact to committed plans, assumptions for the remaining test stages, and impact to delivery dates and quality.

After we reviewed this analysis with both the managers and the engineers, we had reason to believe we were not off target. Both of those groups, as it turned out, felt uneasy about the volume of defects in FVT, but they had not quantified their feelings. They now told us that they believed there was an exposure, and then asked us if we could project where we thought the additional errors might be.

Predictions:

We took a look at all the available data for this release and previous releases to see if we could find any correlation. We used the following approaches in trying to determine where the errors might be:

- Defect volume by product sub-parts
- Defect rate by product sub-parts
- Defect volume based on high volume code module Inspection data
- Defect rate based on high volume code module Inspection data
- Modules which had not been through code Inspection
- Large modules (GT 300 LOC)
- Error frequency in modules tested in FVT
- Evaluations of the product developers
- Least squares method comparing Inspection errors in modules to test errors in the same modules (also with customer found defects)

After we had completed a list of potential modules for each of these analysis, we then rank ordered the modules which had the highest number of intersects between the lists.

The recommendation to the project manager was that 25.9 KLOC be re-inspected to flush out the impact due to increased error volume. Needless to say this was viewed with some skepticism by the project team, as this represented a reasonable portion of this product release, and would take almost a full person year of effort to complete.

The Case For Re-inspections:

Since our objective was to get the overrun in defects removed as soon as possible, we began to build a case which would use a subset of the recommended 25.9 KLOC as a sample to determine the remaining error density.

Based on where the product was at this point in the test cycle, we had projected a remaining defect volume of 24/KLOC. This is to say that there was a latency of this amount in the product. We now needed to prove that this projection was credible. We argued that if the sample re-inspection would find ten errors per KLOC against the projected 24, the effectiveness would be 41% or about what it apparently was for the primary Inspections in this release. If we would find 12, it would be 50% effective. If it would find 16, it would be 67% effective, which was the best this product had historically been able to show. We, of course, had no way to pre-determine the effectiveness of these re-inspections. An effectiveness higher than 41% would be suspect, however, without reasons to demonstrate it should be higher.

In order to try to achieve a maximum effectiveness and minimize the debate about how effective the re-inspections really were, we asked that the Inspection moderators be chosen from a list of people we knew ran effective Inspections. We asked that we be able to insure that each Inspection team member be properly prepared for the re-inspection, that they had been trained as Inspectors, and that they understood that the objective was to find as many defects as possible in the allotted Inspection time. Thus we were trying to ensure that rigorous Inspections were performed. We tried to force the level of effectiveness. This was done to try to get a consistent approach to the re-inspection process. Without this we would be contending with too many unknowns to draw acceptable conclusions from this sample. We asked that the moderators try to execute the Inspections at the recommended Inspection rate of 125 LOC/Hour for preparation and 100 LOC/Hour for the Inspection. Now we had to contend with the possibility that any finding below 10 defects/KLOC might indicate that the product was indeed under quality control.

We additionally wanted the re-inspections to represent a cross current of the complete product rather than to choose from only the highest potential modules. The sample must be representative of the product. Nothing else could suffice.

Four Weeks Later:

After about four weeks, nineteen modules representing 4030 LOC had been re-inspected. In these modules, 59 major defects had been found. This represented 14.6 defects/KLOC, which was well above what had been deemed the least acceptable indicator at 10 defects/KLOC.

Analysis of these 59 defects showed that 5 defects were at the high level design point, 25 were at the detail design point, 26 were at the code point, and 3 were base errors in old code. Therefore, the defects were dispersed across the product work in a reasonable manner. Detail design defects may have been a bit higher than anticipated, but the dispersion was across all activities.

We now had to show that we indeed did have a true sample of the product and not the most error prone modules. Figure 5 shows the decile distribution of error rates for the previous release at two points in time: the end of the first FVT and the end of all testing. It also shows how the data on the case study release compared to the previous product's release.

DEFECT DENSITY (errors per KLOC)

Previous ***Present***

Product KLOC Decile	*Release* FVT End	All Test End	*Release* FVT to date
1st	55.5	114.1	70.2
2nd	19.9	41.6	26.1
3rd	13.7	27.0	20.6
4th	9.9	19.7	15.6
5th	7.0	15.7	7.6
6th	5.0	11 .1	5.4
7th	2.0	7.9	1.5
8th	0	5.2	0
9th	0	0	0
10th	0	0	0
Avg	11.3	24.4	14.6

FIGURE 5

This data offers a number of interesting insights. First, 20% or more of all code shipped never had an error found during the test cycles. Later we found that this same code also never had problems in the users environments. Second, the top ten per cent of the error prone code is almost three times as dense as the next ten per cent in defects. Third, the case study release's data shows a higher error density throughout all of the deciles, except the 7th and 8th, than did the previous release at a similar point in the project's life cycle (FVT). We must remember that the case study FVT was only 70% complete at this point in time.

Of the 19 modules re-inspected the dispersion was as follows:

Defect Density	Decide	Number of Modules
	1st	3
	2nd	4
	3rd	2
	4th	4
	5th	1
	6th	1
	7th	3
	8th	2
	9th	0
	1 0th	1

FIGURE 6

Although it could be argued that this is not an even distribution across each decide the sample was deemed to be distributed evenly enough to suggest that the re-inspection rate was a valid suggestion of a high volume of errors which needed to be removed from across the product. The defects were not all in the most error prone modules. Plans were put in place to remove a predetermined volume of defects prior to the official start of the system test stage concurrent with component testing.

Plan To Remove The Defects:

The product management team accepted that 200 defects needed to be removed immediately to achieve a desired level of quality for stability in the remaining tests for the first operating system. We were asked to re-evaluate our rank ordered list of modules and to resolve with the engineers an agreed to list for re-inspection. We never achieved a list which fully satisfied all the engineers, so the project manager asked

each development team to find a percentage of the 200 defects based on the percentage of the code for which they were responsible. They were also told they could use any method they determined would find the defects over the next four weeks; unit test, re-inspection, or whatever. Most chose re-inspections.

Costs To Re-Inspect:

Inspection preparation proceeds at 125 LOC/hour, the Inspection itself runs at 100 LOC/hour, and the average Inspection for code requires four people. Therefore, for every 1 KLOC of code re-inspected the project would need to expend 88 engineer hours for planning, preparation, inspection, rework and follow-up. We had projected that a minimum of 200 additional defects should be removed prior to the start of the next test stage. This would bring the product back under quality control and would remove the impact to test. This product still had more code to test, so more defects would be discovered, but the percentages found by the test stages should be the same unless some action were taken to change the effectiveness. No determinable actions were evident at this time to change test effectiveness, although a number of them were instituted later as a result of this case study work.

We assumed that we would not be able to find defects at the rate of 14.6 as was done with the sample, because this was a rigorous environment with selected moderators. This situation would not necessarily exist throughout the project's remaining re-inspections. Therefore, we assumed that the general population would find about 10 errors/KLOC or that they would function at about 41% effectiveness for their environment as they had done in the primary Inspection for this release.

Thus, 1760 engineer hours would be needed to re-inspect the 20 KLOC to find the additional 200 defects. This was a cost which was not easy to accept, even if it was only an up front cost and if sufficiently productive would actually save time later. We estimated after interviewing a number of engineers that it cost about 10 hours to isolate a fix, to make the fix, to test it, to reship the fixed code to the code control group, to retest it in the test environment, and to carry other necessary controls. We determined that the cost to fix the average defect found through Inspections ran at less than 2 hours per defect on average. This difference was primarily due to the random nature of encountering errors during test. During an Inspection all errors are found in the same logical period, there are no retests as the problem has not been found in test yet, there are no patches or temporary fixes to carry and control, and the code had only to be shipped once, not every time a test error is repaired.

Accounting for the savings in fix time for the 200 errors, 1600 hours of time would be saved. This savings, however, comes later in the cycle. The real cost for the re-inspections, therefore, was 160 hours, but 1760 had to be invested now. The testers had a saying which communicated this situation: "Pay me now or pay me later." The challenge then was to demonstrate that an up front cost now was in the best interest of the product. Product management accepted the conclusion and committed to find the 200 defects prior to starting the next test stage. The savings due to defects not having to be found by customers would have been much greater.

Interestingly, if the effectiveness were to be equivalent to the sample re-inspections with 14.6d/KLOC, the costs would only be 1284 hours to find 200 defects. This would be a real and immediate savings of 316 hours compared to FVT.

At one point the question was raised: Why not more testing in FVT? At that time FVT was finding roughly 3 defects/tester week. 20 KLOC of code re-inspected would cost 44 engineer (tester) weeks. 44 tester weeks of FVT would find 132 defects, which was 68 defects fewer for an equivalent amount of calendar time in re-inspections. These 68 defects are the leakage to CT, ST, and customers where the costs are significantly higher. The answer was obvious: Inspections were more productive, and would not require machine time to find the defects.

Refined Projections:

The question of how much of an impact existed to test we felt still needed to be addressed in more precise projections. Two viewpoints were taken to try to put more exact boundaries on the possibilities.

The first argued that if the 14.6 defects/KLOC found in the sample represented 50%t effectiveness then we were potentially facing 29.2 defects/KLOC for a volume of 1226 defects remaining in all the new and changed code.

The second viewpoint argued that if 28.2% of all defects were found in the last release by the end of FVT, then the same relationship would hold true for this release. This would suggest that 1425 defects remained to be found.

We now had a boundary of 1226 to 1425 which was reasonably close to the previous worse case projection of 1444. This boundary was tighter than the previous estimate prior to the sample re-inspection, but was still open to too many questions.

> First, the true internal Inspection error rate and effectiveness relationships were not exactly determinable.
>
> Second, as a result of re-inspecting a number of the modules it was learned that the true total KLOC under development was still in question. If it were off by more than a few KLOC, it could explain some of the differences in apparent Inspection defect finding rates.

Figure 7 shows the data for the previous release and a mapping using the same detection percentages in the release under study.

The projections in Figure 7 are based on the assumption that FVT will find 709 defects in the first system tested. This product required testing for three different operating system environments which it supported. Each operating system had different interfaces and functions which caused different and unique detects to be found. Most defects, however, would be common across the operating systems. At the time of this projection the first FVT was not complete, but was within 95 defects of the 709 projected. The second assumption was that the three operating system tests would occur in the same order as they had in the previous product releases. This was not true, so a revision was made in the defect projections based on the first operating system now being the more difficult one to test and the one which seemed to cause the most different set of errors to be flushed out during test. We were not sure how much this affected defect distribution and whether we had significantly missed something in our thinking about the apparent effectiveness of the primary Inspections, but we proceeded. The revisions to the projections are shown in Figure 8. These revisions, since they had no data history, were made with the best available knowledge of people in the test groups. Also important to note is that as the percentage of defects found by the first operating system in test increases, the total absolute defects across all three tests decreases. This is due to efficiencies in test and common problems being found early. Figure 9 shows the actuals after all testing was completed. We see that the first (most difficult operating system) actually did flush out more defects than projected and that for all three operating systems the total was less than projected.

ORIGINAL PROJECTION of
DISTRIBUTION of DEFECTS

		Previous Release		Case Study Release	
1ST SYSTEM		%	#	%	#
	FVT	28.2	499	28.2	709
	CT	9.7	171	9.7	244
	ST	12.9	227	12.9	324
	B/R	5.3	93	5.3	133
	Misc	10.6	187	10.6	266
subtotal		66.7	1177	66.7	1676
2ND SYSTEM					
	FVT	8	141	8	201
	CT	2.6	46	2.6	65
	ST	7.1	126	7.1	178
	B/R	4.5	79	4.5	113
	Misc	3,7	66	3.7	93
subtotal		25.9	458	25.9	650
3RD SYSTEM					
	FVT	1.9	33	1.9	48
	CT	0.1	2	0.1	3
	ST	1.4	25	1.4	35
	B/R	2.6	46	2.6	65
	Misc	1.4	25	1.4	35
subtotal		7.4	131	7.4	186
TOTAL		100	1766	100	2512

FIGURE 7

While these new projections were taking shape, the developers were busy trying to find the 200 defects set as a target, and the FVT was working itself towards a completion. Most of the developers chose to use re-inspections, but a few tried an enhanced Unit Test, and one group developed a new version of an Inspection combined with a flow analysis. When all the efforts were completed against the designated code in the four allotted weeks, 209 defects had been discovered predominantly in re-Inspections and fixed prior to the component (CT) test starting. This quality focus resulted in the CT and ST finishing on schedule, which was the first time this had been accomplished in the history of this product. The product shipped to the users on schedule, and had proven to be the best of the product levels in the field from a quality perspective.

NEW PROJECTIONS DUE TO CHANGE IN TEST ORDER

		Change In Test Order	
1ST SYSTEM		%	#
	FVT	30.5	709
	CT	10.7	249
	ST	16.9	393
	B/R	6.3	146
	Misc	10.6_	246
subtotal		75	1743
2ND SYSTEM			
	FVT	5.7	133
	CT	1.6	37
	ST	3.1	72
	B/R	3.5	81
	Misc	3.7	86
subtotal		17.6	409
3RD SYSTEM			
	FVT	1.9	44
	CT	0.1	2
	ST	1.4	33
	B/R	2.6	60
	Misc	1.4	33
subtotal		7.4	172
TOTAL		200	2325

FIGURE 8

Undoubtedly other factors also led to the quality success of the product, but the ability to focus attention on the quality problem early and to put management actions in place to ensure the quality level was a key factor. Imagine what might have been possible had more data with better integrity existed at the time of this study.

By the time all the testing was completed on all three operating systems, the error dispersion looked as shown in Figure 9. When this is compared to Figure 8, we see that the projections were within a tolerance of acceptability for a product with limited data, and that the biggest factor affecting the actuals versus projections was due to the volume and percentage found in the first operating system test. It was lower that the original or refined projections. How much of this was due to the 200 defects flushed out by taking early action cannot be clearly known, but I suggest it was a major contributing factor. The 462 defects shown in the Misc section of the Actuals for the first operating system were due to the defects found in the re-inspections.

ACTUALS

1ST SYSTEM:				
FVT	34.9	766		
CT	14.4	316		
ST	7.6	166		
B/R	5.5	1 20		
Misc	18.3	462		
I/P	0.9	1 9		
Subtotal	81.6	1849		
2ND SYSTEM:				
FVT	4.2	92		
CT	3.4	75		
ST	1.7	37		
B/R	1.4	31		
Misc	2.2	48		
I/P	0	0		
Subtotal	12.9	283		
3RD SYSTEM:				
FVT	1.5	34		
CT	0.8	17		
ST	0.8	17		
B/R	1.2	26		
Misc	1.2	26		
I/P	0	0		
Subtotal	5.5	120		
TOTAL:	100	2197		

FIGURE 9

Effect Of Lines Of Code As A Parameter:

During the same period that the defects were being removed with re-inspections, we began to look at the LOCs actually added and modified for this release. We found, not surprisingly, that the total KLOC was higher than projected or understood to be in the code. As it turned the total KLOC in the product was the major factor for the defect density appearing to be too high at the time of the initial analysis.

If we look at Figure 10 the data for the Inspection defects has been corrected with the right KLOC for the product parts. We see that the curves for the three releases of this product look more reasonably the same. Is this an accident, or is history repeating itself in this environment?

Interestingly, the quality of the product was actually improved due to a mistake in the count of LOC allocated in this case study product release, and we see that there are relationships between these vital parameters that help us control the quality of the product.

CONCLUSION:

We see in this case study that when one variable diverted from plan then at least one other variable changed. That is, when the KLOC was in error the apparent Inspection detection error rate was incorrect and misleading. The signal was given when the recorded error rate was very high in the Inspections, but a false assumption was made that this was good because the Inspections were more effective. Upon investigation it was found that there was no reason for an improved effectiveness in the Inspections. This should have led to questioning of the KLOC estimates. However, not enough was known about the integrity of the data at that time. Subsequently it was shown that the KLOC estimates were indeed in error, and when accurately represented that the defect rates in the Inspections were roughly no more than history had suggested they would be. However, due to the poor estimating and tracking of KLOC the impact to test time and the quality of the product were still real.

The work resulting from this study demonstrated once again that when a development group chooses to focus on quality it can indeed deliver a higher quality product. The resulting re-inspections and the defects found all helped produce a product release which was substantially the best in quality compared to previous product releases.

We see that quality is controllable even with rudimentary data. We do not need to wait to get to Level 3 to begin to do quantitative process management. Our capability will improve with better data, but we do not need to wait.

Finally, the information was presented in a case study format so it could be used as a learning tool. It is not intended to be conclusive, some questions remained in 1979 and may even remain today about the completeness of the data and analysis. But we have learned since then, we have done more with data analysis, we know more is possible, and we will do more yet as we begin to apply statistical rigor to our data analysis.

Caution is necessary, as this case study's data is relevant to its system environment, and should not be used defacto in any other environment. Adjustments are required if the data in this study are used by others in their projects. The data can, however, be used to start comparisons in other environments, if nothing else exists. The data, though, must be used judiciously.

CORRECTED DEFECT RATES

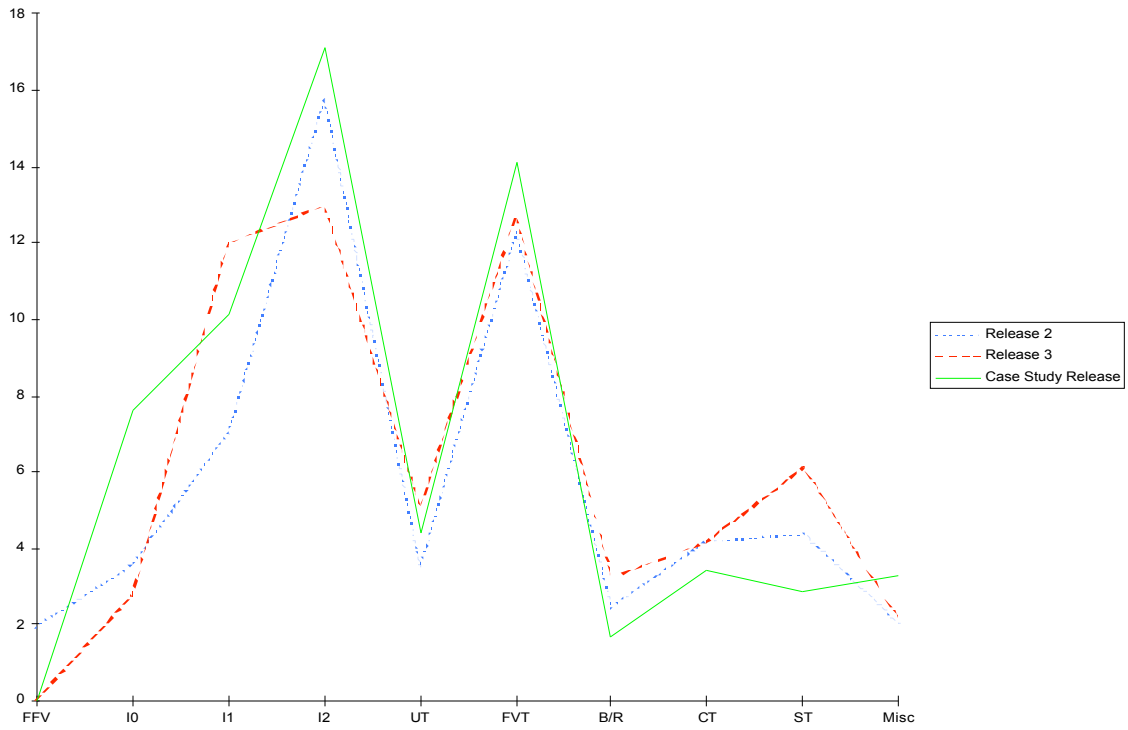

FIGURE 10

References:

[BER68] von Bertalanffy, Ludwig, General System Theory, George Braziller, New York, 1968.
[FOR61] Forrester, Jay W., Industrial Dynamics, The M.I.T. Press, Cambridge, Massachusetts, 1961.

Footnotes:

[1] It is interesting that the CMM describes these variables in Level 2 and 3 Key Process Areas, with the exclusion of test progress.

[2] In retrospect it appears that too often a process group which has demonstrated effectiveness is disbanded and the advances made are reduced or worse the old habits come back to life.