Presentation Notes
Paper
Bio
Return to Main Menu

PRESENTATION

T15

Thursday, November 4, 1999
1:30 PM

DEFECT MANAGEMENT IN DEVELOPMENT AND TEST

Ed Weller
Bull Worldwide Information Systems

INTERNATIONAL CONFERENCE ON
SOFTWARE TESTING, ANALYSIS & REVIEW
NOVEMBER 1-5, 1999
SAN JOSE, CA

STAR'99 West

Defect Management in Development and Test

Ed Weller

Fellow, Software Process

Bull Worldwide Information Systems

Phoenix, AZ

e.weller@bull.com

Worldwide Information Systems

Bull

An Industry Perspective

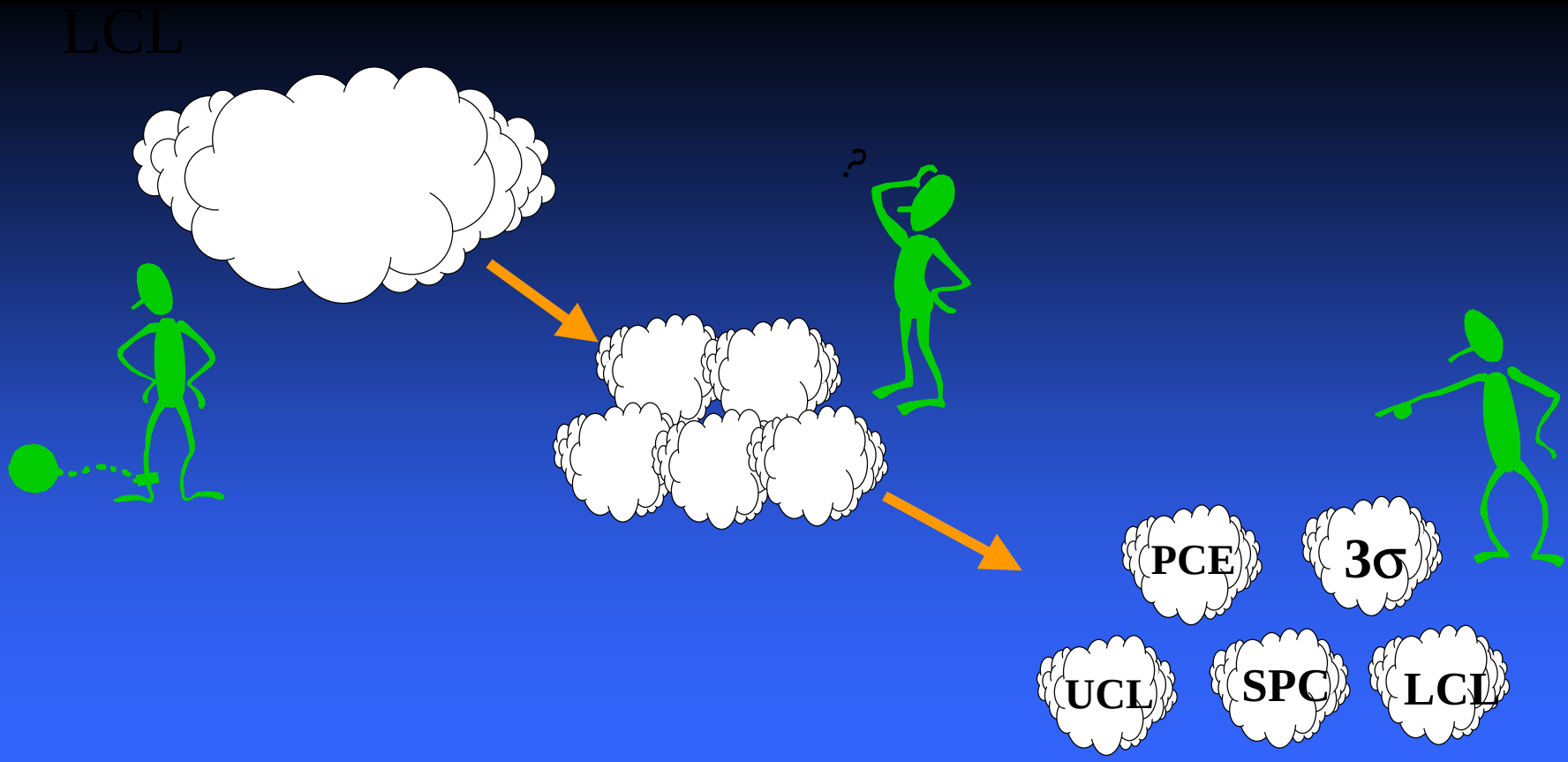
- Few organizations use data to manage their development and delivery process
- Fewer organizations use data to predict events in development and test
- Those that do maintain a significant advantage over those that do not

State of the Practice

- Do you test your product?
- Do you have a defect tracking system?
- Do you capture the cost to find and fix defects?
- Do you capture source of defect?
- Do you use this data to plan your next project?

Bull 

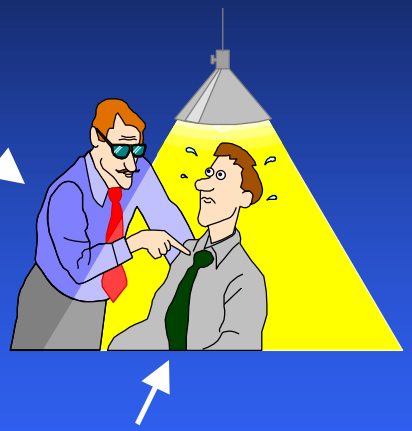
Organizational Use of Data



The Test Manager's Perspective
What Causes Heartburn?

- **When will you be through with test?**
- **How long will it take to fix THAT bug?**

Program Manager



Test Manager

What's wrong with this picture?

Predicting Test Finish Dates

- **What are the input parameters?**
  - **Number of defects**
  - **Defect discovery rate**
  - **Defect closure rate**
  - **Effort to close a defect**
  - **Elapsed time to close a defect**

How Many Defects Are in the Product?

- How big is the product?
  - KSLOC, Function Points, Forms, Reports
- How complex is the product?
  - Need your historical data
  - Complexity measurement tools
- What's the current project history for this product?
  - Defect removal before entering test

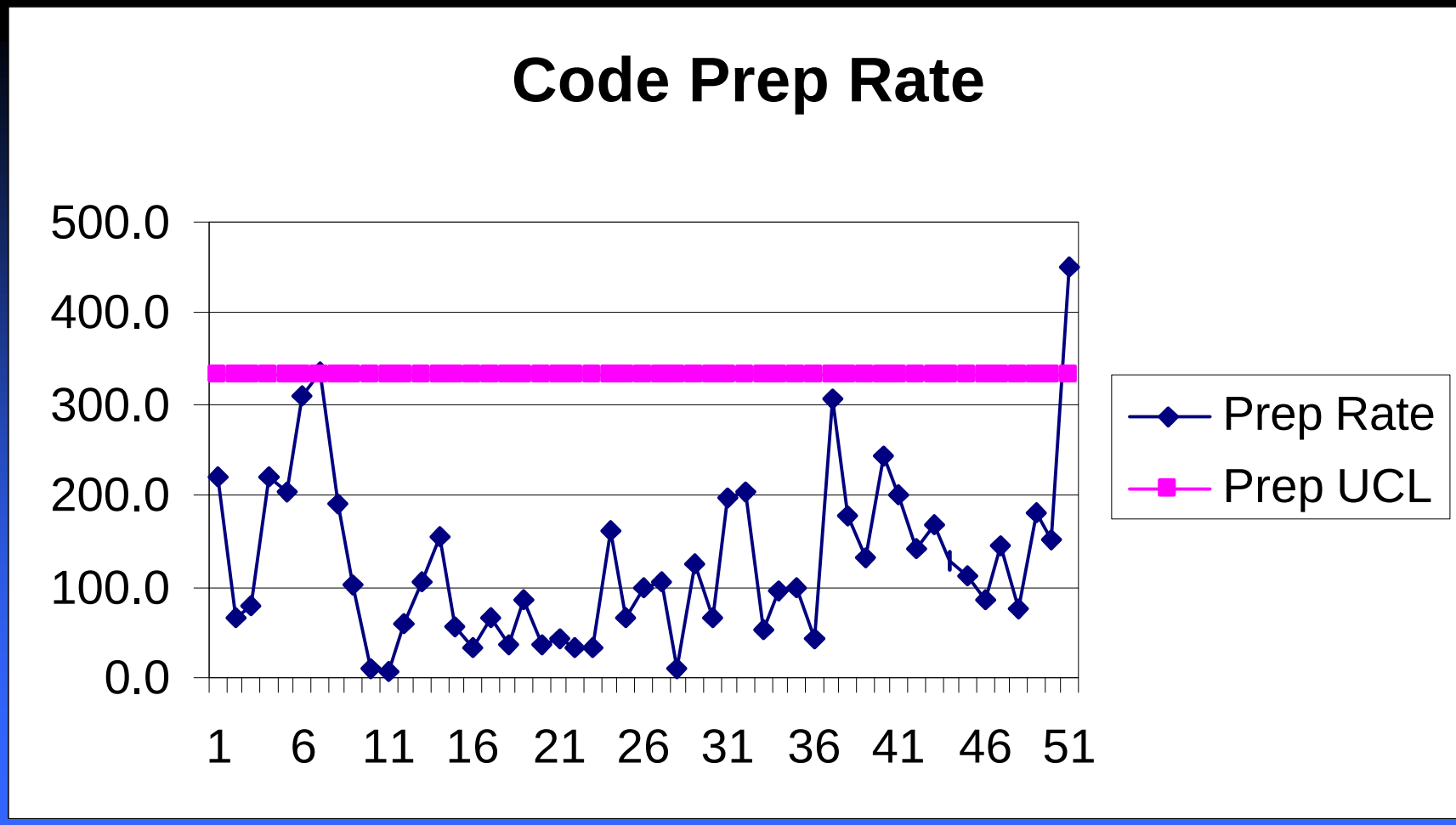
Inspection Defect Data

- What is your typical defect inspection effectiveness in code inspections?
- A first order approximation of defects remaining uses inspection effectiveness data
  - At 50%, you enter Unit Test with as many residual defects as were found in code inspections
    (Defects found/Inspection Effectiveness)
- Modified by product history and team capability

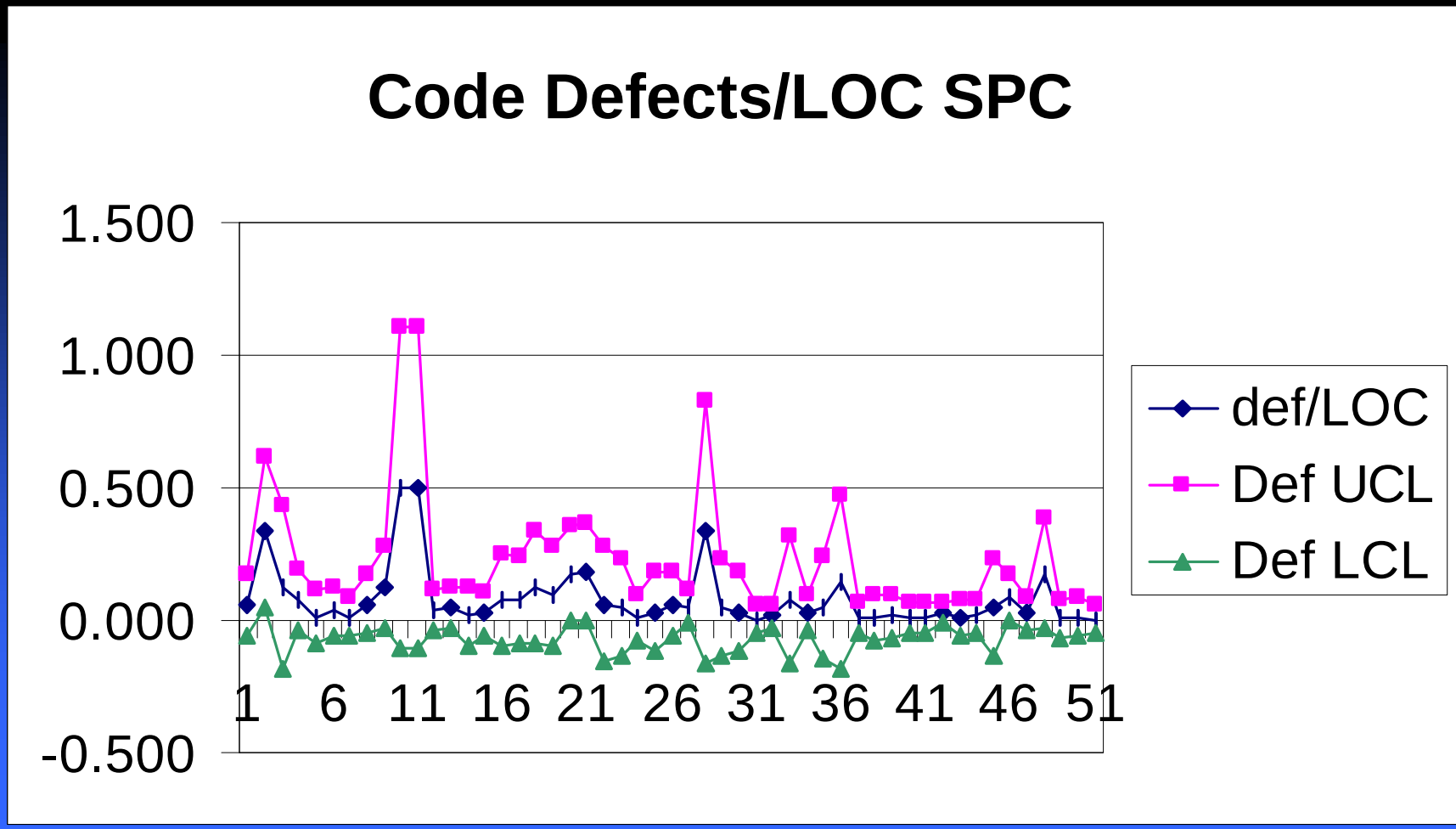
Was the Inspection Process Controlled?

- **Statistical Process Control can help**
  - **Are the Preparation and Inspection Rates within the Upper and Lower Control Limits?**
  - **Is the work product defect density within control?**

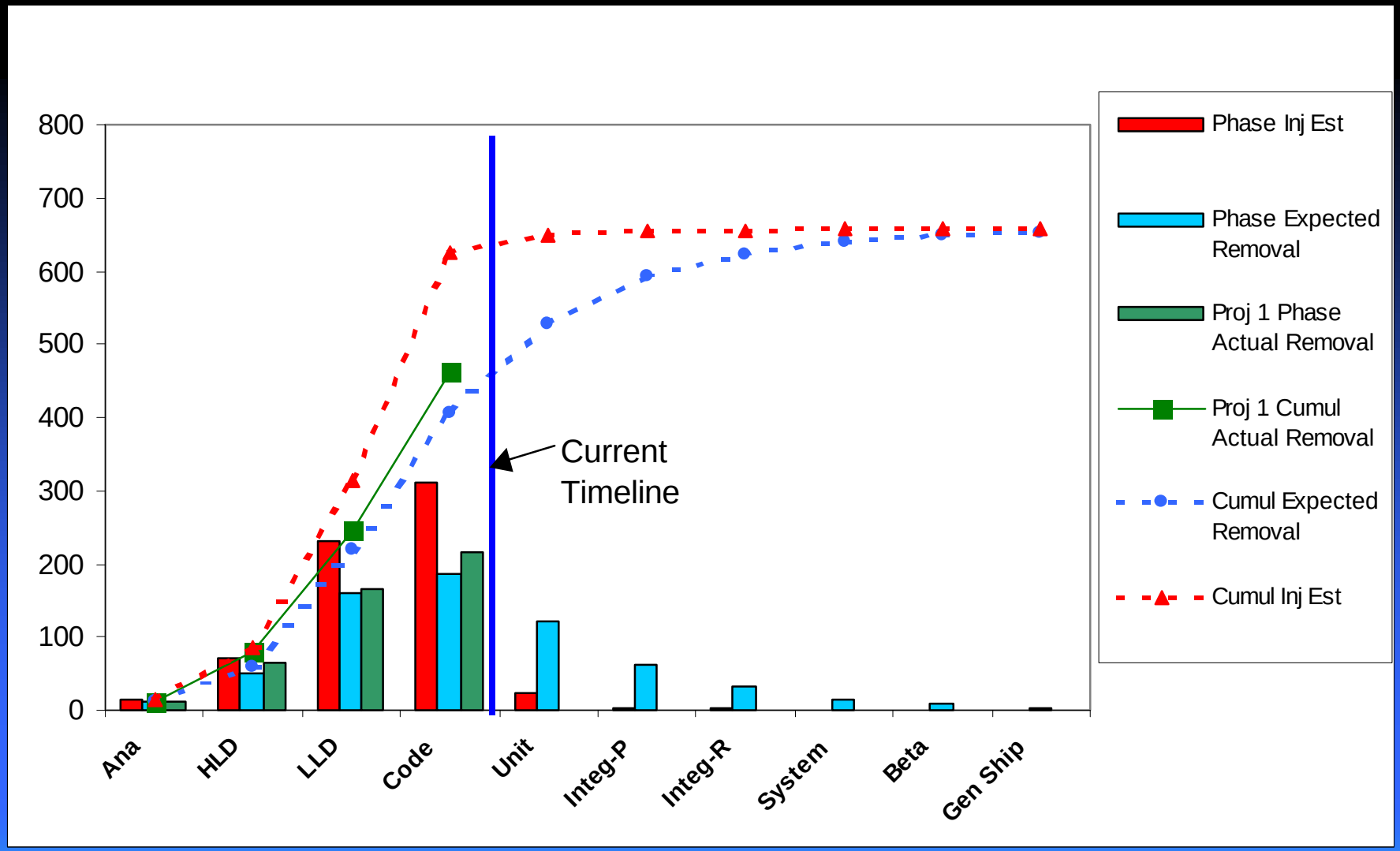
How Well Did You Inspect?



How Good Is the Product?



Establishing Defect Profiles

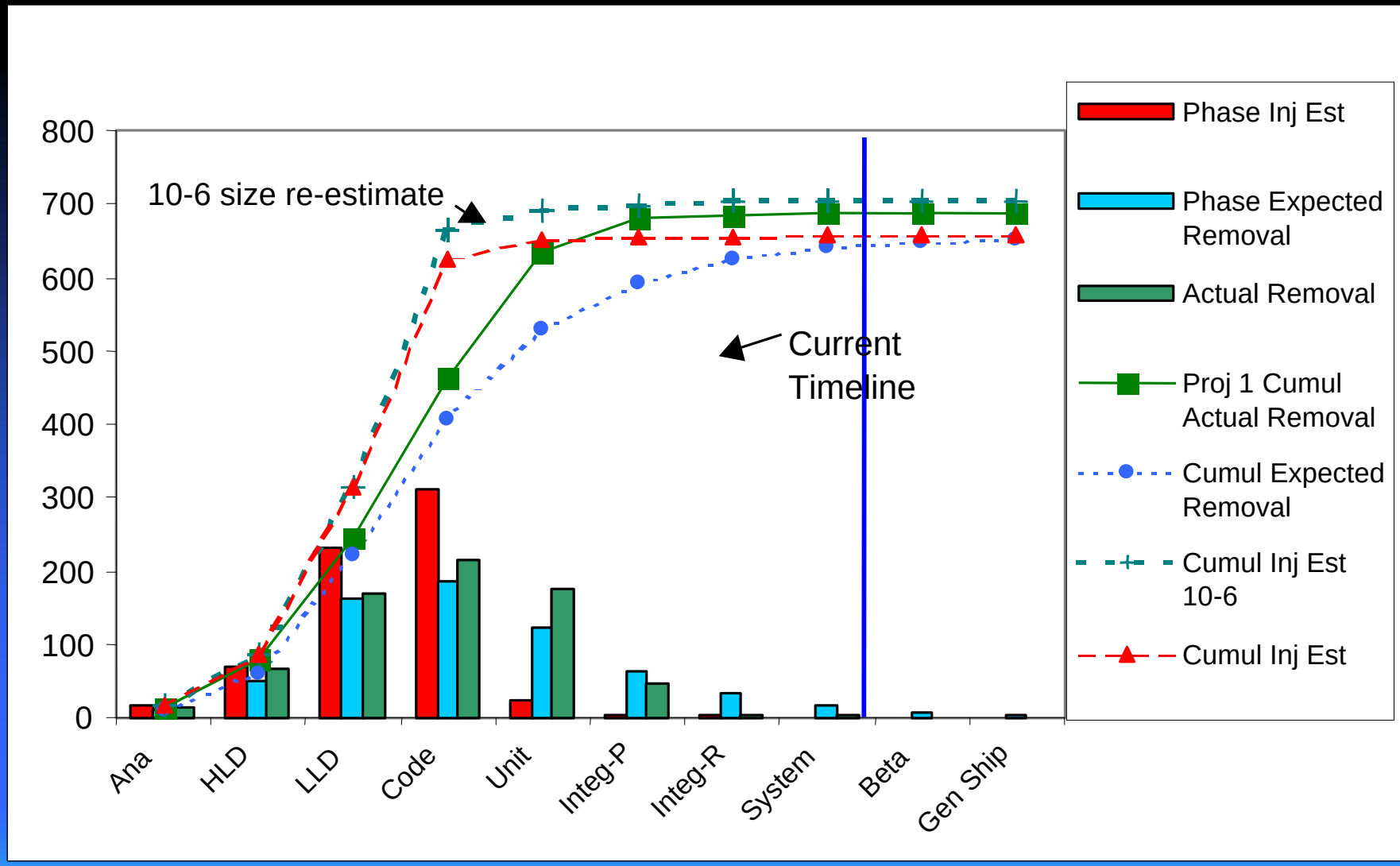


Data in examples has been changed, but is representative of real projects

Evaluating Inspection Performance

- Expected effectiveness was 60%
- Data showed a 70% effectiveness
- Possible causes:
  - Better inspection technique and results
  - Lower defect injection rate
  - Size estimate was low
- Count of actual code showed a 13% increase in code size over design estimate

Project Re-estimate



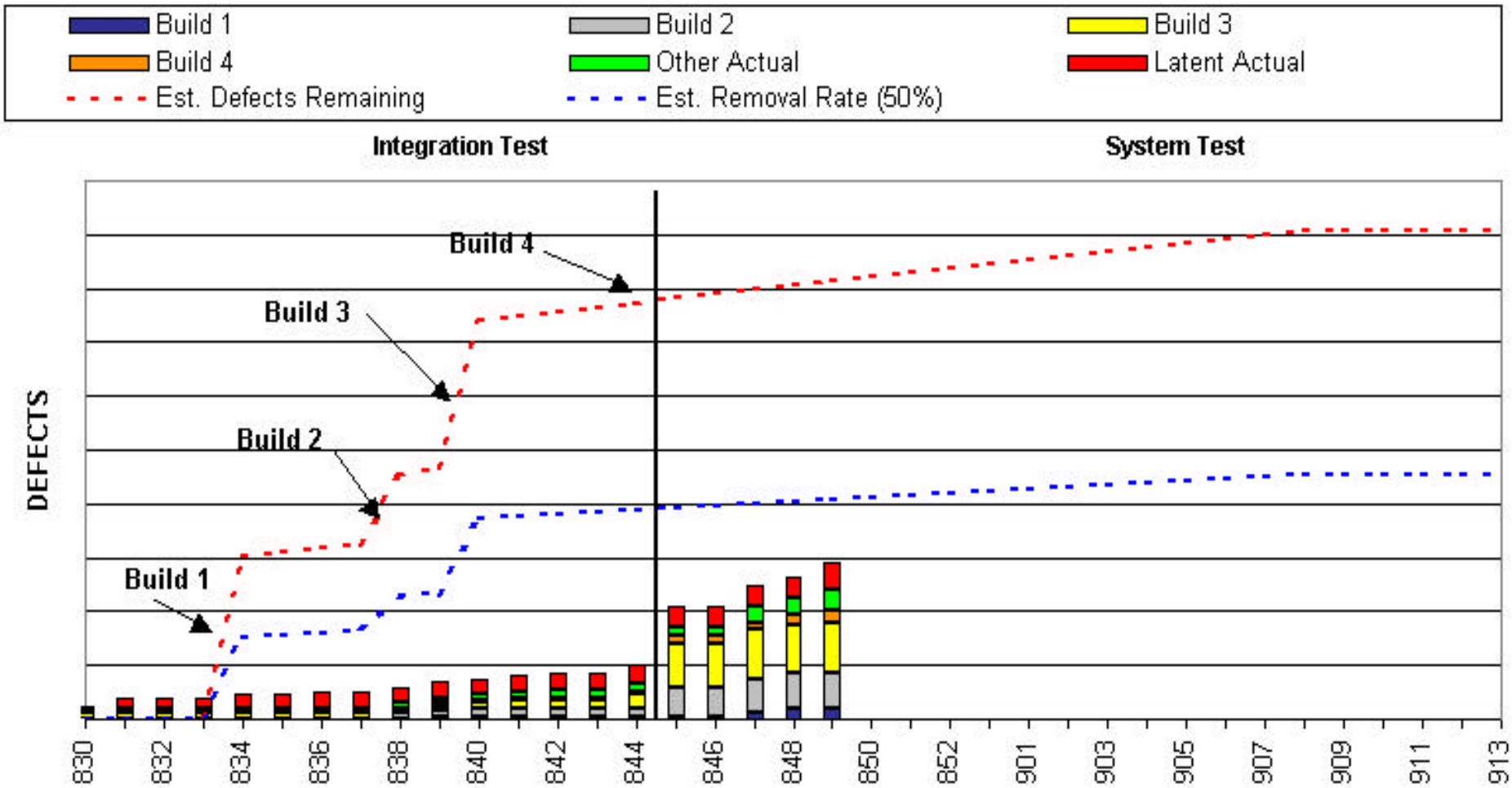
What Changed?

- To "preserve" the original estimate, only the new cumulative total was changed
- A defect injection rate of 20% of the fixes was used for each test phase (Folklore is that this value is as high as 50%!)
  - Measured this at 10%
  - All fixes are inspected

Defect Discovery Rate

- Problem rate is easy to measure
- Defect rate is harder to measure
  - Problem must be closed to count it
  - Defect fixing takes time, so current defect counts don't tell the whole story
- Estimated injection rate is needed for evaluation of test progress

One View of Test Progress



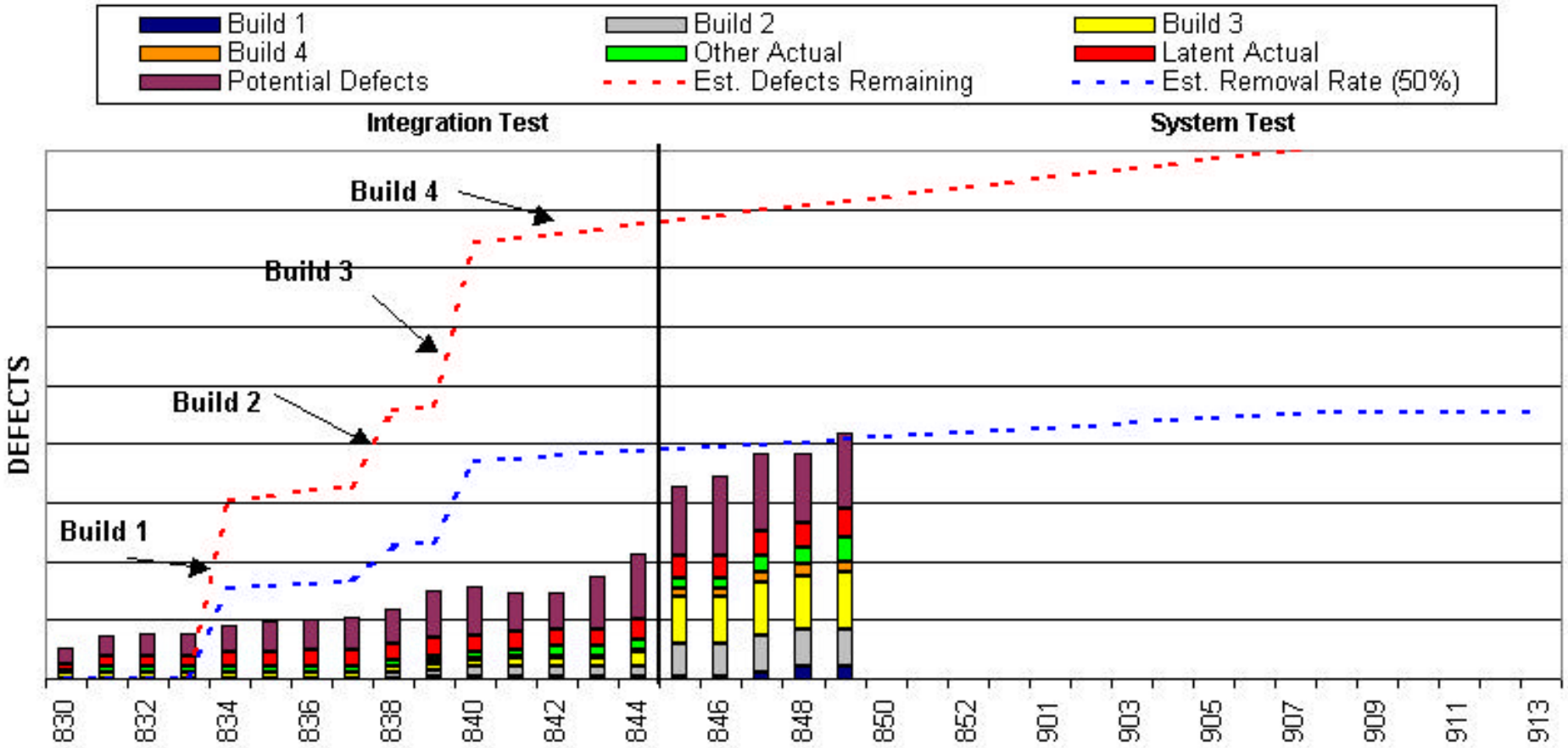
What's Missing?

- **Problems are not closed immediately**
  - **What is your "average" time to close a problem?**
  - **What's your real time to close a defect?**
- **Backlogs hide the true status of test efforts - current defect counts don't tell the whole story**
- **Do you know your "First-Time-Fix" ratio (Percent of problem reports that are product defects?)**

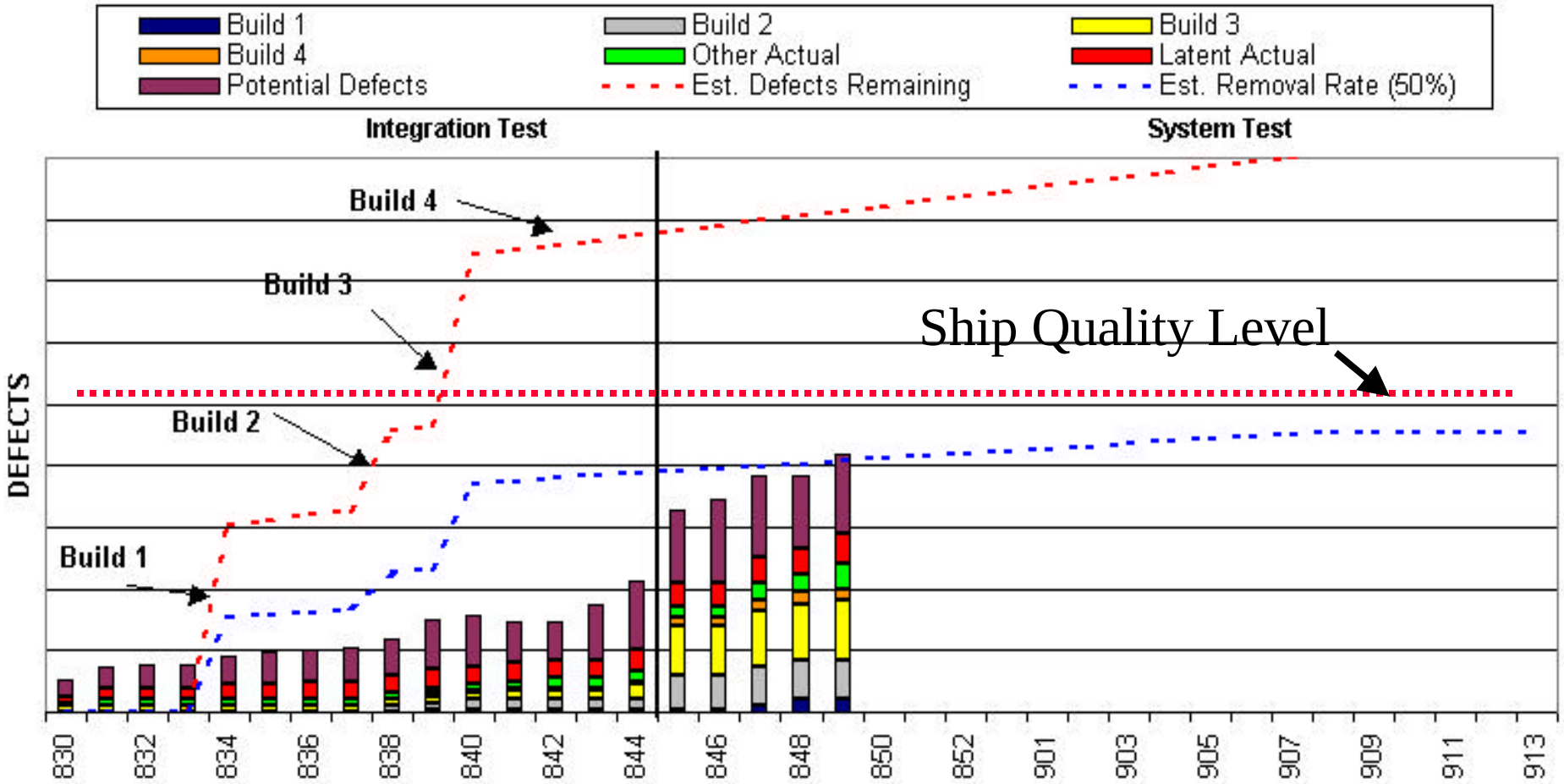
Problem-Defect Resolution

- Until problems reports are closed, the true defect discovery rate is hidden
  - Backlog
  - Duplicate problems
  - Tester errors
- Apply "first-time-fix" ratio to open problem reports ("potential defects") to get a clearer picture of test progress

A More Complete Picture of Test Progress



When Can You Ship?



The Tester's Perspective
What Helps You?

- **What am I missing?**
  - **Defects found later in test**
  - **Post ship defects**
  - **Same defect as in the last release**
- **Tests run and test failures**
  - **Product defects**
  - **Testware defects**
- **Test escapes**

Counting (measuring) all these will help you become a better tester

The Test Manager's Perspective
Answering the Questions

- **When will *YOU* be done?**
  - **How many defects are in the product?**
  - **How many defects have you removed?**
  - **How many defects do you need to find?**
  - **Where do you need to find them?**

The Test Manager's Perspective
Answering the Questions

- When are you finding the defects?
  - Functional defects in System Test?
  - Are you finding any defects in Unit Test?
- Are you finding the right defects?
  - How many problem reports are closed with fixes?
  - How many problem reports are ignored?
  - How many "user error" closures are there?

Developer
What Did I Fail To Detect?

- What leaked into Integration and System Test?
  - Should I have found it in Unit Test?
  - Should I have found it by Inspection?
  - What is the cost tradeoff?
- How can I do better the next time?
  - Defect prevention
  - "Awareness"

Summary

- **Defect measurement helps you understand the dynamics of development and test**
- **Defect measurement provides data for cost tradeoffs**
- **Defect measurement is necessary to predict product quality**

Defect measurement can provide a competitive edge

Defect Management in Development and Test

Edward F. Weller
Fellow, Software Process
Bull HN Information Systems
13430 N. Black Canyon
Phoenix, AZ 85029

Tele: (602) 862-4563
Fax: (602) 862-4288
e-Mail: **e.weller@bull.com**

Bio:
Edward F. Weller is a Fellow at Bull HN Information Systems in Phoenix, AZ, where he is responsible for the software processes used by the GCOS8 operating systems group.

He received the IEEE Software "Best Article of the Year" award for his September 1993, "Lessons From Three Years of Inspection Data", and was awarded the Best Track Presentation at the 1994 Applications of Software Measurement conference for "Using Metrics to Manage Software Projects". He is a member of the SEI's Software Measurements Steering Committee, the Embry Riddle University Computing and Mathematics Industrial Advisory Board, and was the first Co-Chair of the Software Inspection and Review Organization, a special interest group promoting the use of inspection process.

Ed has been the Program Chair for ASM 96 and ASM 99, and will again be Program Chair for ASM 2000. He is also affiliated with Software Technology Transition, providing consulting in the area of Inspections, metrics, and the Capability Maturity Model®. He has 30 years experience in hardware, test, software, and systems engineering of large scale hardware and software projects and is a Senior Member of IEEE.

Mr. Weller received his BSE in Electrical Engineering from the University of Michigan, and MSEE from the Florida Institute of Technology.

®CMM, and Capability Maturity Model are registered in the U.S. Patent and Trademark Office

Defect Management in Development and Test

A simple survey I have been conducting at conferences since 1994 demonstrates few organizations use defect data to manage their product development. I have asked the series of questions below, with the typical responses following the questions:

Do you test your product?
No one admits they do not.
Do you have a defect tracking system?
Most organizations do this in 1999 (considerably better than 1994).
Do you capture the cost to find and fix defects?
About ½ do this.
Do you capture source of defect?
About ½ of the remainder do this (Down to about 1/8th at this point).
Do you use this data to plan your next project?
Typically only 2-3% do all of the above.

What is amazing to me is that of those who have collected all the data, only about 1/4th of them actually use the data to plan their next projects. The troublesome aspect of the lack of measurement is that organizations do not understand where as much as one-half their development resources are being spent, and what they might do to improve their development productivity.

Specifically, defect data from test data can be used to answer the age-old questions "When will you be finished with test?" and "How long will it take to fix ***that*** bug?" I've always maintained the development team should answer both these questions, since the length of test is a function of the number of defects in the product (at least until the defect discovery rate becomes relatively low). How long it takes to fix a problem can be predicted based on past data, but this is one of those "on average" answers that never seems to satisfy the person asking the question unless they understand the problem, in which case they probably wouldn't have asked the question!

Predicting Test Finish Dates
In order to predict the end of test, you need the following data:
- Number of defects
- Defect discovery rate
- Defect closure rate
- Effort to close a defect
- Elapsed time to close a defect

To predict the number of defects, the product size and defect injection rate per unit of size are needed. Defect discovery and closure rates should be for defects, not problem reports. As we shall see later, the ratio of defects to problem reports is an important metric. The effort and elapsed time to close defects and problem reports can be used to evaluate the effort needed to complete test. If resources

are fixed (and when do we ever have unlimited resource?), the effort and elapsed time relationship can be checked against the schedule.

Defect injection predictions require some knowledge of past history and a comparison or value judgement of how well that history can be used to predict today's project. For an organization that counts defects with reasonable[1] accuracy, the following table has worked for me as a rough starting point[2]:

Phase	Requirements	Analysis	HLD	LLD	Code
Defects/KLOC	1	3	6	20	30

Two to three iterations through this process can provide enough experience and data to make these predictions fairly accurate. Don't be confused into thinking this analysis requires you to follow a waterfall model. Think about the type of work being done, and the defect removal related to that work. Your profile will also develop its own distinctive shape as you tailor it to your product, project, and personnel capability. "Defects/KLOC" is the measured defect density against the measured size when the project ships the product. Until you have actual size data, the number is based on the size estimate. I use KLOC in this example, but other size measures can be used.

Phase and cumulative injection rates are shown in Figure 1.

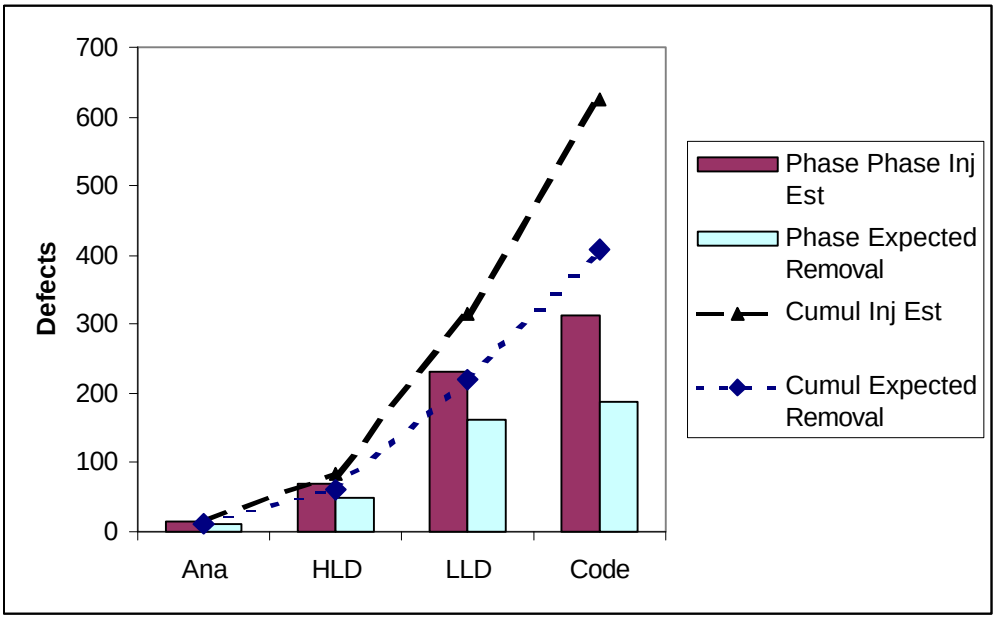


Figure 1 – Initial Defect Injection and Removal Estimate

[1] "Reasonable accuracy" means they know what a major defect is in an inspection, and they collect unit test data

[2] These numbers have been reported anywhere from 4-5 to 200 defects per KLOC so some care is needed when selecting values for your organization.

We are looking for the number of defects in the product as test starts. We have an estimate if defects injected, so it would be useful to estimate the development stage removal rate. This example assumes a review or inspection process is used during development.

In Figure 1, the gap between the two dashed lines is an estimate of the number of defects in the product when test starts. A more complete analysis would include ranges for defect injection rates and removal rates. I show this analysis to the project teams, but usually leave the ranges off the charts used in project reviews to avoid clutter. The $64 question is obviously how accurate are the injection and removal estimates. This is where a detailed analysis of the inspection data using SPC proved useful.

Inspection Data Analysis

On these two projects the first opportunity to apply SPC was during code inspections. When analyzing data, I generally look at all the data to see if any patterns are present. On one project, the work was divided into two parts; the creation of a product feature, and the revision of existing code. A histogram of preparation rates in lines of code per hour is shown in Figure 2.

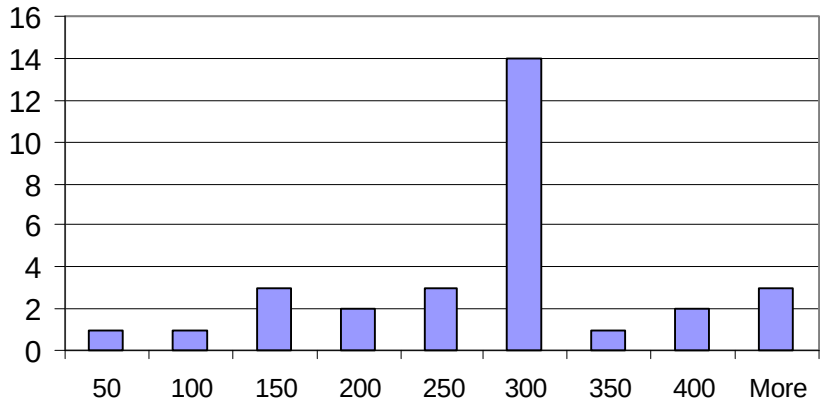


Figure 2 – Preparation Rates for 30 inspections

This group of inspections seems to fall into the 250-350 range[3] with a few possible outliers. Three of the rates below 150 lines per hour were for small amounts of code, and of the 3 above 400 lines per hour, two were due to small size and the other was a very large chunk of code. I did investigate the high preparation rate inspections and found only the one with large size and high rate to be a problem. This inspection occurred near the end of the coding phase, when familiarity with the product and time pressure typically causes higher preparation rates. I also compared the preparation rate distribution to the inspection rate, shown in Figure 3.

[3] This project counted source and comment lines of code – Source LOC rates were 40% lower.

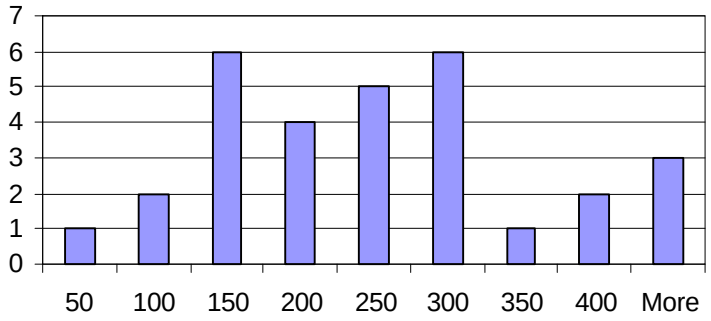


Figure 3 – Inspection Rates for 30 inspections

There is an appearance of a bi-modal distribution in Figure 3. Since the data was from inspection of "new" code (the new modules developed specifically for this product) as well as changes to the existing base, I divided the preparation rates into two classes, with the results of Figure 4.

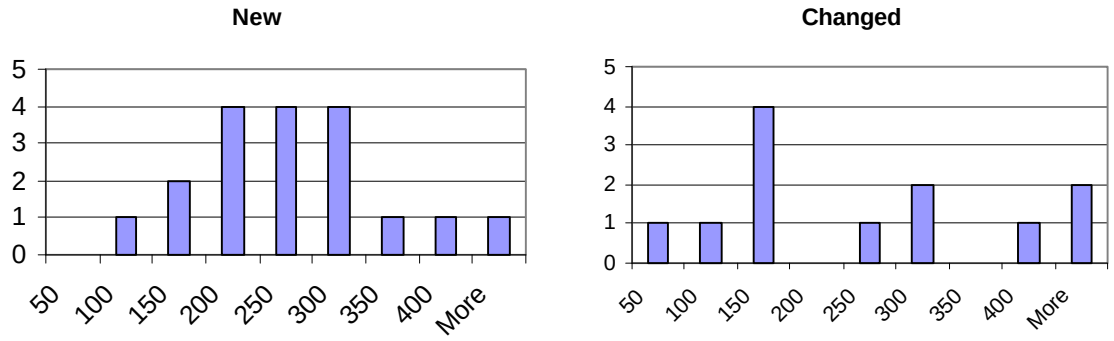


Figure 4 – New vs Changed Inspection Rates

I expect new code inspections to be "better behaved' than changes to existing code. (Many inspections of modified code are small in size, causing preparation and inspection rates to have a larger variance. Knowledge of the changed (old) code inspected may also have a wider variance than the new code). The separate views in Figure 4 are typical of much of the inspection data I investigate. The new code approximates a normal distribution as closely as you may see with real data.

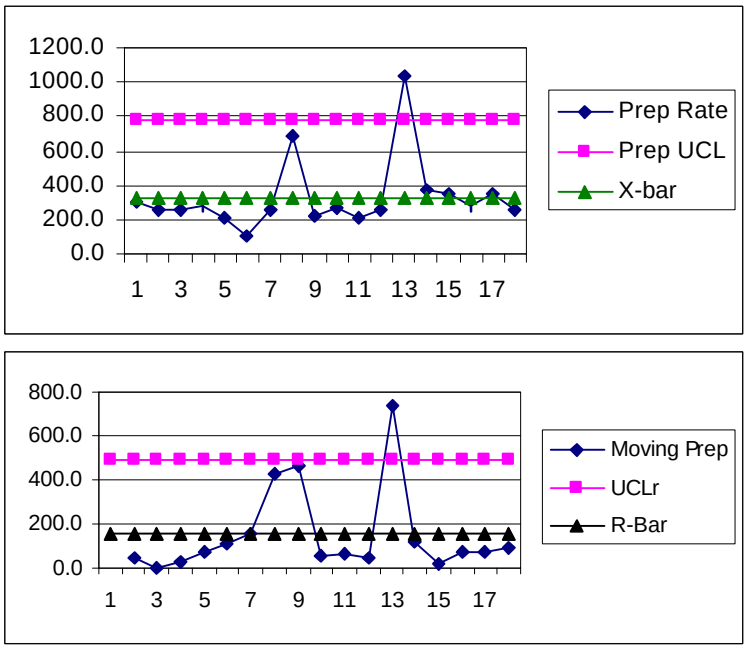


Figure 5 – XmR Chart for Preparation Rate of New Code[4]

There are several indications this inspection process was not in control – Inspections 13 caused an out of control point for both the Preparation rate and Moving Range chart. Also, the first 7 points on the Preparation rate chart are below the mean[5]. When inspection 13 was investigated, the high preparation rate was due to one of the three inspectors not preparing for the inspection. When Inspection 13 was removed from the dataset, inspection 8 then fell outside the Upper Control Limit. Inspection 8 had a high preparation rate due to sections of "cut & paste" code. With Inspections 8 and 13 treated as "special causes of variation", the recalculated XmR charts in Figure 6 show the process is a controlled process. There are several points to consider when performing this analysis;

- The variation in rates may be due to either a process violation or a work product that is unusual (in this case - no preparation by one inspector and a work product with repeated code)
- Is there truly a special cause of variation? Look at the remaining data critically to see if poor preparation is a problem, even if the data is within the control limits. When the decision is made to remove a data point from the analysis, it really has to be a "special cause". Figure 6 shows the remaining data to be well behaved, suggesting the removal of the 2 data points is justified.
- If the inspection or preparation rates are out of control, is the product a possible cause of the out of limit condition (lack of time might not be a cause of high preparation or inspection rates). A poorly written

[4] A.Burr and M.Owen, *Statistical Methods for Software Quality*, International Thompson Computer Press, pp114-128 explain the derivation of XmR charts

[5] 8 or 9 points on the same side of the mean is another test for a controlled process

document, or ‘cut and paste” code could cause the preparation rate to be outside the limits.

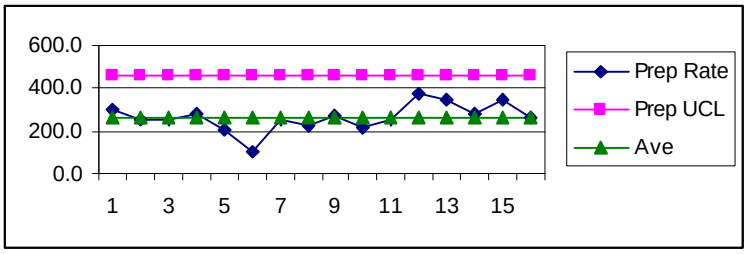


Figure 6 – Preparation Rate with Outliers Removed

The next step is to look at the control charts for defect density. When the inspection process is under control, I view the defect density as an indicator of the product quality, rather than a process indicator. All inspections (both new and changed code) were included in Figure 7. Although not shown, the data for the new code inspections were “better behaved” than that for inspections of changes, since the sample sizes were more uniform and larger for the new code.

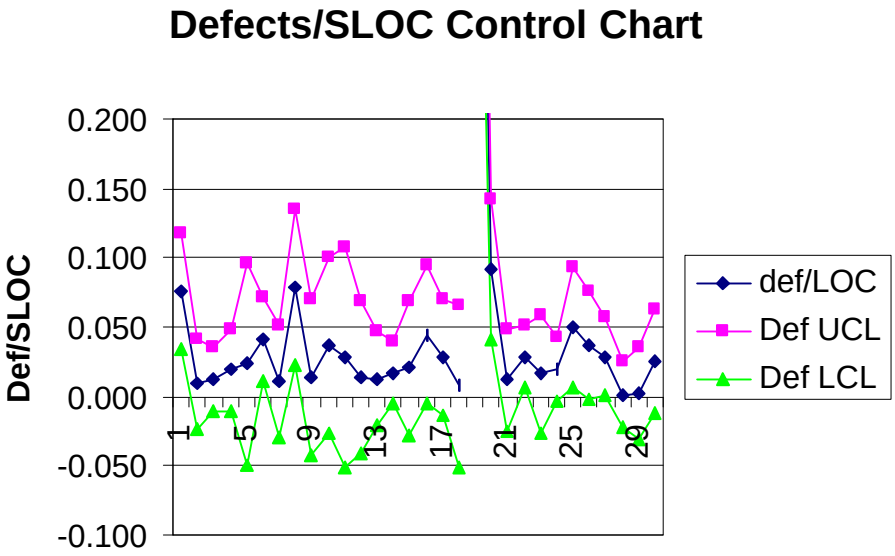


Figure 7 – Defect Density Control Chart[6]

In preparing this chart, I used a u-chart since the sample size (area or opportunity, in this case lines of source code) varied considerably. The U-chart equations are:

U-bar = $\Sigma u_i / \Sigma a_i$, or the total number of defects divided by the total size.

UCL_u = U-bar + 3 * $\sqrt{U-bar} / a_i$

UCL_L = U-bar - 3 * $\sqrt{U-bar} / a_i$

where a_i is the sample size.

[6] The lower control limits cannot be less than zero, although for convenience the LCL was plotted on this chart as calculated. Once you verify the data is above the LCL, for possible values of LCL, it is probably better to delete this line from the chart.

What have we learned about this product and its contribution to the system release? With two exceptions, the inspection process seems to be well controlled. The two outliers were investigated (as were other inspection meetings) to understand how well the inspection process was performed. The defect data was within control limits.

The first project in the release contributed the following data:

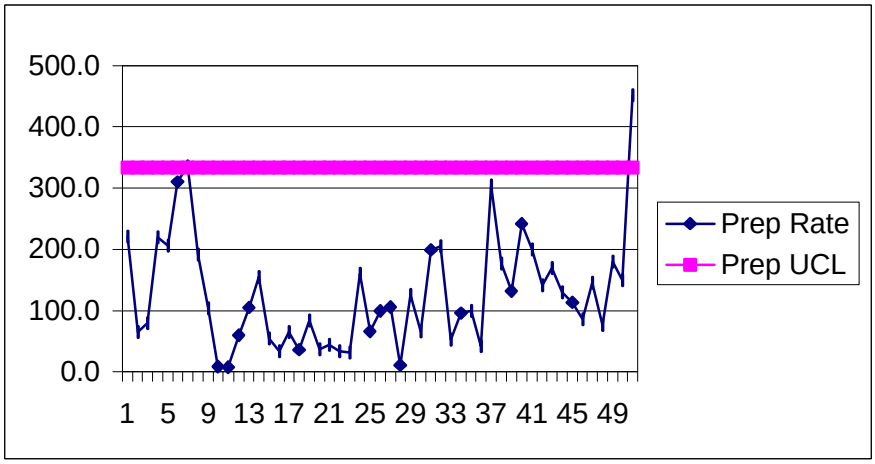


Figure 8 – Preparation Rate – First Project

Other than the ubiquitous last inspection, this series of 51 inspections had only one meeting here the preparation rate was out of control. Defect data was all within control.

This data was periodically discussed with the project teams at their weekly team meetings for several reasons. First, it sent a message that the data was being used to make decisions on the project. Second, keeping the estimates and data in front of the team kept them aware of the progress toward the quality targets. Third, it was a deliberate attempt to avoid the “metrics are going into a black hole” problem that causes metrics programs to fail.

We now had two sets of data showing inspections were reasonably well performed. Inspection data was then used to refine the prediction for the number of defects remaining to be found for these products in the release. Based on the differences in inspection process data from the second project, the inspection effectiveness estimates for the two parts of the project were changed; higher for the new code, and lower for the changed code. In the absence of data, this would seem a likely thing to do, but basing the change on data rather than assumption adds credibility to the prediction process[7], as well as a baseline for future predictions.

[7] Credibility is important if you ever get into the situation where a slip in the release date is being discussed – sometimes having data to help with the decision process will help win the discussion.

The defect depletion chart with defects removed is shown in Figure 9. Note that more defects were found than estimated, however we now had an actual count of size, and replotted the estimate as shown in Figure 10.

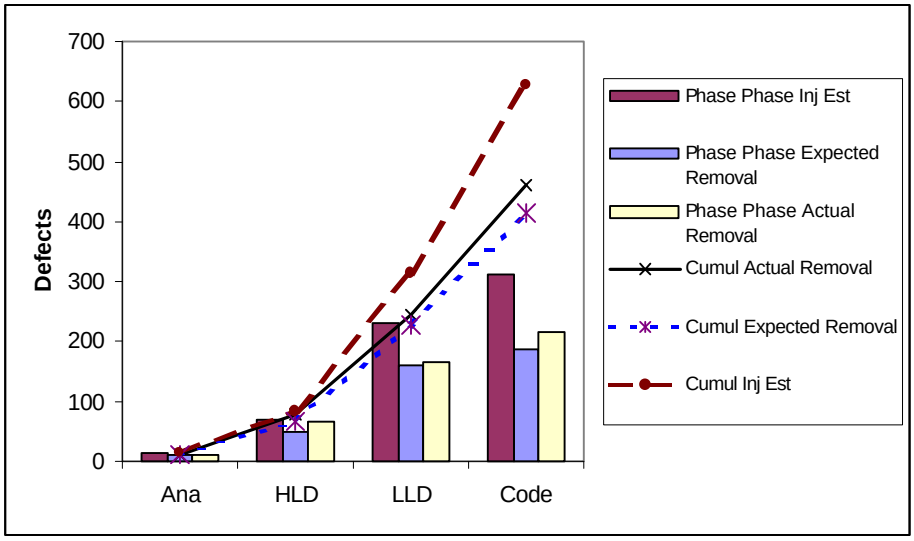


Figure 9 – Defect Depletion at End of Code Inspection

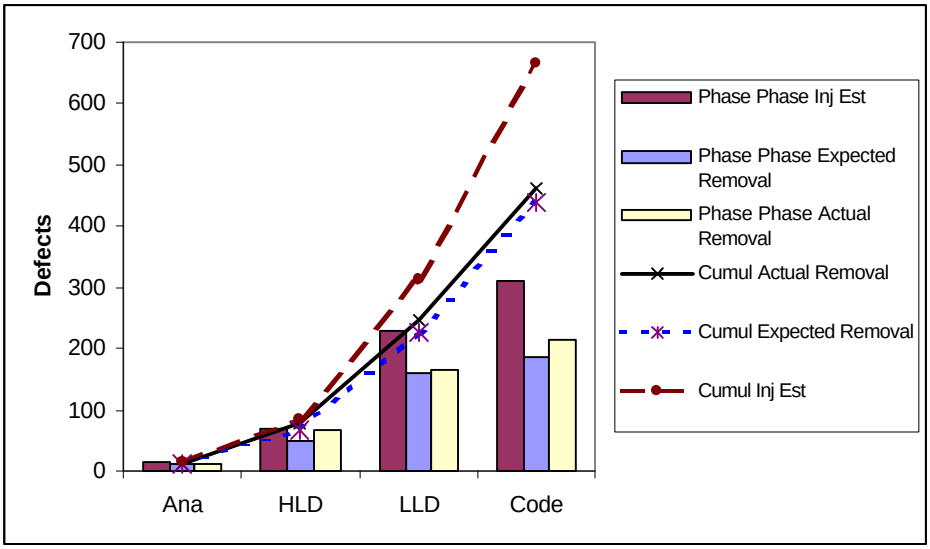


Figure 10 – Replotted Defect Depletion with New Size Estimate

Defect Prediction

At this point the defect predictions can be verified. Originally based on an estimated injection rate and size, you can apply the estimate of inspection effectiveness to the size and injection estimates to check their reasonableness. In the table below, I constructed three sets of data to illustrate how to verify initial estimates with inspection data. The calculated and estimated effectiveness for product 1 agree within 10%. Note that this “small” difference for product 1 (only 10%) translates into a 100 defect difference when entering test, which can have a significant impact on test elapsed time and total effort estimates. This is where

the inspection process data is important in determining which estimate to believe – size, defect injection rate, or inspection removal effectiveness. If the inspection process was under control, with rates that were reasonable and with the right people in the inspections, then look to the size and injection estimates for correction. If on the other hand, the data suggests the inspection process was sloppy (out of control), then the effectiveness should be lowered. The amount to lower would be based on an understanding of the inspection process

For product 1, if the inspection process were in control, I would lower the defect or size estimate to reach a revised calculated effectiveness of 60%. This would lower the total defects to 833. At this stage, you should have an actual code size, so the injection rate is probably the first candidate for change. Obviously, product 3 has one or more very bad estimates. Either the injection rate is off by 2-3 to one, the size is much smaller, or the inspections did not remove as many defects as expected. Since at this point in the project there is an actual code size, check to see if 1) all the code was inspected, 2) the size agrees with initial estimates, 3) that the injection rate estimate was reasonable, or 4) the code was poorly inspected. Product 2 seems to be on the money[8].

Product	Estimated Size	Estimated Injection Rate	Defects Removed by Inspection[9]	Calculated Effectiveness	Original Estimated Effectiveness
1	20 kloc	50/kloc	500	50%	60%
2	40 kloc	40/kloc	1000	62.5%	60%
3	15 kloc	40/kloc	100	16%	50%

Adjusting the Estimates

Returning to the two development projects, in the case of project one, the adjustment caused by the size re-estimate was about 13% more defects – not a large number, but significant as you enter the later stages of test. It may appear that the adjustment was to make the estimates look better, however the reason for making the pre-test data fit as well as possible is to get a better estimate of the defects remaining in the product. The reason for the changes to the estimates should be captured for later use in the project lessons learned, and as input to following project estimates.

There is no hard and fast rule that I can offer when re-estimating. I look to the data that has the most “substance”, that is, who provided it, the track record for accuracy of previous estimates, what’s most likely to be suspect, and “estimator’s instinct” as well as the data charts. Inspection data is the first thing I evaluate,

[8] R.A. Radice, Getting to Level 4 in the CMM, a tutorial delivered at the Software Technology Conference May 1, 1997, Salt Lake City, Utah. The case study in this tutorial relating defect injection rates, defect removal rates, and size estimates was first presented to the author in 1991.

[9] As in all cases where inspection defects are discussed, “defect” refers to a major defect, one that could show up in test or product use if not removed.

using both the statistical methods mentioned above, and polls of the inspection team members to see how they think the inspections have been conducted.

Unit and Integration Test

Both projects kept accurate records of defects found during Unit and Integration test. Both projects developed test objective matrices and developed test plans and specifications, so we had some expectation to remove defects more effectively than the 30-50% “norm” often quoted in the industry.

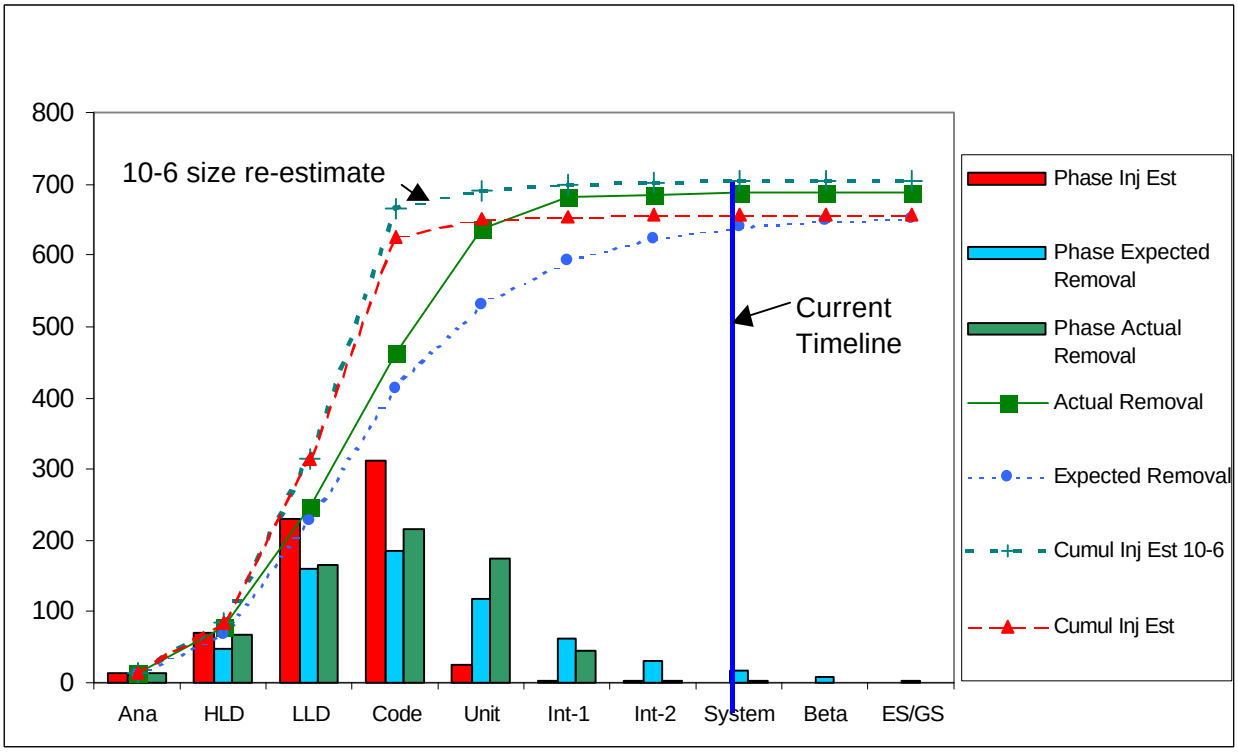


Figure 11 – Project One Defect Depletion

Figure 11 shows project one, as it was about to enter System Test (this chart is used in our monthly project review as well as the weekly team meetings). It shows the re-estimate for the number of defects injected, however I did not replot the expected removal to reduce clutter on the chart. In the project reviews the focus is on the gap between estimated injected and actual removed. More complete charts with re-estimates of phase and cumulative data are used in discussions with the project manager. Note the defect removal in Unit Test was higher than estimated and that subsequently in the two phases of Integration Test a small number of defects were removed. Without accurate defect removal data from Unit test these low numbers would be of more concern with respect to product quality. The Current Timeline is indicated to show the furthest stage where the project defect removal is happening.

Evaluating Test Progress

One difficulty we all have in measuring test progress is determining how many defects have actually been found. If a significant backlog builds up, and average

response time lengthens, the actual number of defects removed does not give a good picture of test progress.

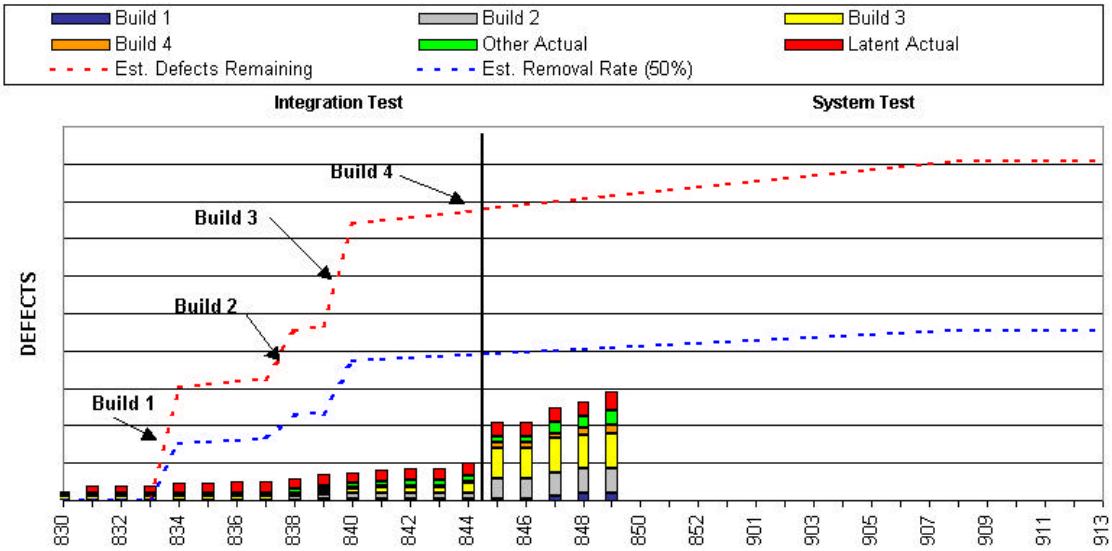


Figure 12 - One View of Test Progress

Figure 12 shows one way of displaying defect removal during integration and system test. As builds add more functionality to the release, the total estimated defects in the product, and estimated removed, increases. The slope continuously increases due to bad fixes (new defects are introduced) as well as continuing corrections from previous releases. At about week 909 new corrections are deferred until a later release to prevent unacceptable churn.

A second view of test progress factors in the backlog. To do this, you have to know the percentage of problem reports that actually turn into defects that are fixed. The “First-Time-Fix Ratio” can be used to provide a snapshot of the total defects plus potential defects in test, as shown in Figure 13.

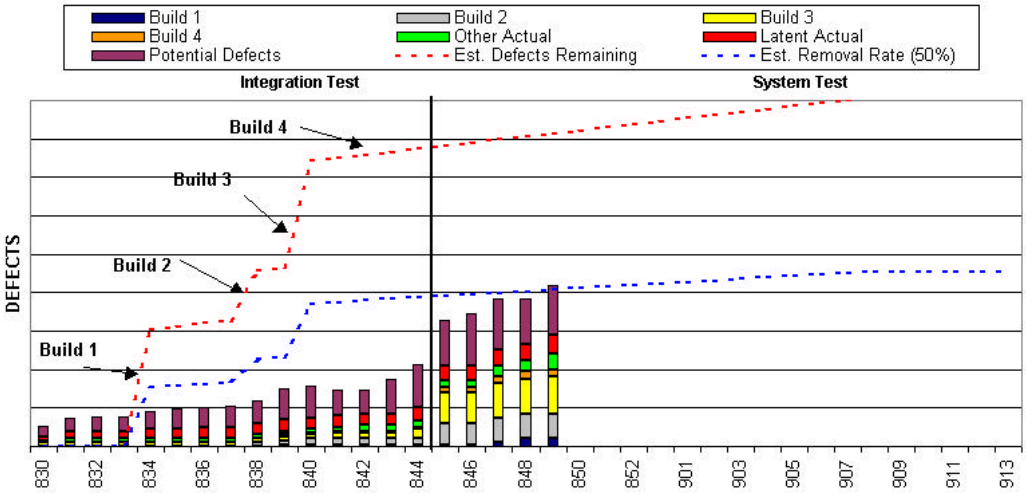


Figure 13 – A More Complete View of Test Progress

One effect observed with this analysis is that the first-time-fix-ratio tends to lower percentages as you get closer to the ship date, so be careful to watch the problem to defect ratio for changes.

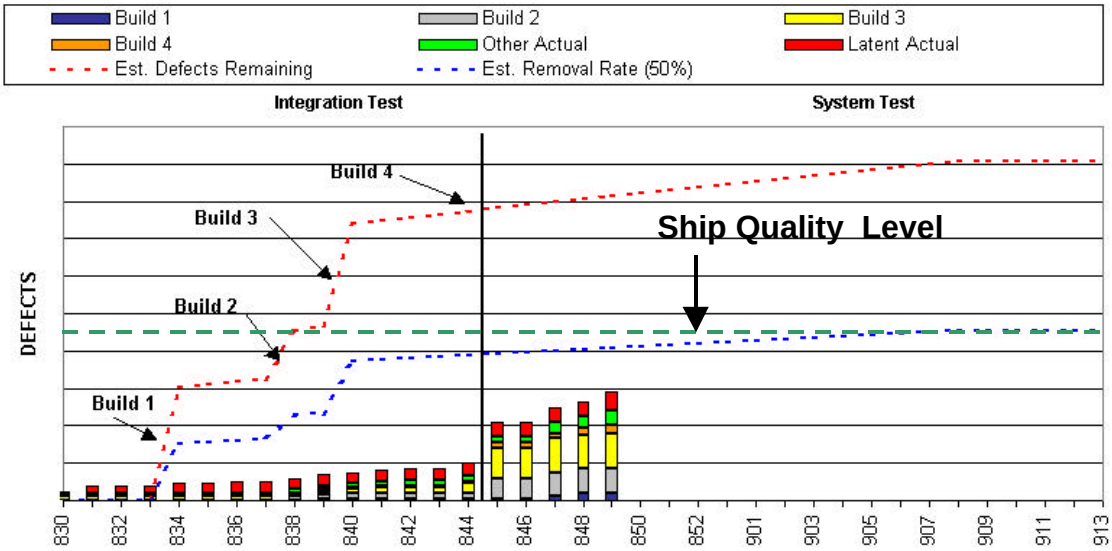


Figure 14 – When Can You Ship?

With the data and analysis represented by Figures 1-13, it is possible to establish ship criteria based on defects found and predicted post ship defects[10]. Historical data on the effectiveness of your test stages is necessary. The ”Quality Level” in Figure 14 was arbitrarily set at 505, but you would want to set this based on past history (what you can expect to do) and what your customers expect (or require).

Conclusions

You should ask two questions about any metric or analysis technique:

- Is it *useful*? Does it provide information that helps make decisions?
- Is it *useable*? Can we reasonably collect the data and do the analysis?

What we have shown in this article is that defect measurement and SPC can be used, with minimal additional cost, to evaluate process data and product quality. The analysis can provide useful information to project managers, release managers, and development teams. The calculations are relatively easy to do, and spreadsheets make the work easy (useable). The additional cost included:

- Analysis of inspection data
- Collection of unit test data
- Analysis of integration and system test data

Our inspection data is in a database, which allows extraction of the data necessary for the inspection SPC charts by a single query. This data is inserted

[10] Note post ship failure rates and defect densities are two different measures. Fenton and Pfleeger treat this in *Software Metrics*, PWS Publishing Company, 1997, pp 344-348

into a spreadsheet template which plots preparation rate, inspection rate, and defect density U-charts. This process takes less than 5 minutes. Unit test data collection is not free, but the cost savings in later problem analysis and tracking offset this cost. I estimate that on a per project basis the initial data analysis cost is less than 1-2 hours per week. Additional cost incurred as part of specific investigation initiated by the data analysis is not included.

Our experience in this release has been that the analysis has been useful, and the net gain has been positive. We have focused on the process aspect of the release as well as the normal problem resolution. The process information has identified several “do-betters” for the next release.

As always, the numbers should be used as a guide, not as absolute rules. They should cause you to ask questions. Knowing where to look or what to look at is the beginning. The answers should accurately explain or resolve the process or product quality issues.

Further Reading

In preparing this article I relied on three sources of information:

1. *Statistical Methods for Software Quality*, Burr and Owen
2. *Understanding Statistical Process Control*, Wheeler and Chambers
3. *Advanced Topics in Statistical Process Control,* Wheeler, SPC Press, 1995
4. *Practical Software Measurement: Measuring for Process Management and Improvement*, Florac, Park, and Carleton, CMU/SEI-97-HB-003

The length of this article precluded exploring many of the details and reasoning behind the calculations, for those interested in exploring this topic in more depth, these references are suggested.

EDWARD F. WELLER

Edward F. Weller is a Fellow at Bull HN Information Systems in Phoenix, Arizona, where he is responsible for the software processes used by the GCOS8 operating systems group.

He received the IEEE Software "Best Article of the Year" award for his September 1993, "Lessons from Three Years of Inspection Data," and was awarded the Best Track Presentation at the 1994 Applications of Software Measurement conference for "Using Metrics to Manage Software Projects." He is a member of the SEI's Software Measurements Steering Committee, the Embry Riddle University Computing and Mathematics Industrial Advisory Board, and was the first co-chair of the Software Inspection and Review Organization, a special interest group promoting the use of the inspection process.

Ed has been the program chair for ASM 96 and ASM 99, and will again be program chair for ASM 2000. He is also affiliated with Software Technology Transition, providing consulting in the areas of inspections, metrics, and the Capability Maturity Model. He has 30 years' experience in hardware, test, software, and systems engineering of large-scale hardware and software projects and is a senior member of IEEE.

Mr. Weller received his B.S.E. in electrical engineering from the University of Michigan, and M.S.E.E. from the Florida Institute of Technology.