# Testing – The Anti-Quality Process

**Tom Walton**
**Ottawa Software Quality Association Meeting**
**March 31, 2005**




## Contents

## Introduction

- Tom Walton – Independent Process Improvement Consultant
- 16 years experience in Software Quality Assurance and Process Improvement.
- 33 years total career experience in software development
- http://www.thedreaming.org/~tomw/
  - contains a lot of useful information and links – feel free to explore
- Contact Information
  - (819) 770-7230
  - tom_walton@videotron.ca




## The Nature of Software Quality

## What is Quality – Really?

- “Quality is adherence to requirements.” Philip Crosby, *Quality is Free*
- “A good way to start any inquiry is to define your terms. According to my Little Oxford Dictionary, “Quality” is a noun meaning "degree of excellence". “Excellence” is defined as "surpassing merit", “Merit” as "goodness", and “Goodness” as “Virtue". Bob Green
- Based on these definitions, Quality is difficult to quantify.
- Unless Quality can be quantified, it is not possible to claim that it is getting better or worse.




## What is Quality – Really?

- The work of Juran. Shewhart and Demming was aimed at the reduction of waste, primarily in manufacture.
- My preferred definition is: “Quality is the lack of waste.” This can be restated as “waste free” as in “My organization produces waste free software.”
- Waste is anything that causes unnecessary costs or losses.
- Quality is thus an economic issue.
- Quality is measured in **$** - easy to quantify.

## From the Web Page of MapleWorks Ltd.

"MapleWorks delivers, to its' customers, products of superior quality and reliability while reducing their cost of development and time to market. MapleWorks is able to deliver these benefits by employing a **proprietary methodology**, by understanding the challenges faced by software development companies in the new economy, and by utilizing a development team that has extensive technical expertise coupled with domain experience in delivering quality product.
Further, MapleWorks offers the enterprise the benefit of near-shore rather than offshore development while maintaining cost and speed to market advantages. Advantages include Canada's geographical proximity along with compatible time zones, language, culture, and IP protection to its U.S. client base. With this combination of resources and grants, **MapleWorks can develop software superior to its competition while reducing development costs by over 50%** for its enterprise customers."




## From the Web Page of Light House Technologies

## How do they do that?

- These companies have found ways to reduce waste.
- The primary source of waste in software development is rework.
  - Most rework consists of finding and fixing defects in the integrated product.
- Only enough testing of the integrated system should be done to prove that the developers did a good job.
- All other back-end test effort is waste.
- Back-end testing is the **Anti-Quality** process.




## The Nature of Software Defects

## The Nature of Software Defects

- If you build your system out of perfect parts, you are not guaranteed a perfect system. But if you build your system out of defective parts, you **are guaranteed** a defective system.
- Once the code is written, all of the defects that it will ever contain have been built into the product (neglecting defects inserted by repairs).
- Defects are like mushrooms growing in the woods – the more there are, the easier they are to find.
- The mushroom analogy allows us to develop two important principles.




## Mushrooms

- Consider a wooded park with many branching and joining paths.
- On one summer morning there are N mushrooms in the park.
- A person, who follows a specific route, picks all X mushrooms along his path.
- On a second summer morning, there are 2N mushrooms in the park. By following the same path (and thus expending the same effort), the mushroom picker will find 2X mushrooms.
- Case 1: Fraction found = X/N  Defects remaining = N-X
- Case 2: Fraction found = 2X/2N  Defects remaining = 2N-2X
- For the same effort, the same fraction of the mushrooms will be found in each case.
- For the same effort, the more mushrooms found, the more will be left undiscovered.

## Mushrooms And Defect Estimate

- The Mushroom Analogy can be used to estimate the total number of defects in a system.
- Consider a system containing N defects that has been subjected to 2E units of test effort.
- The first E units of effort produced X defects. The second E units of test effort produced Y defects.
- We can write the equation:

  $X/N = Y/(N-X)$
- Solving for N yields:

  $N = X^2/(X-Y)$

  Subject to the condition that $Y<X$




## The Nature of Software Defects

- Barring delays caused by finding defects, the more defects there are in a system, the more defects a given amount of test effort will find.
- The corollary: The more defects that are found, for a given amount of test effort, the more defects remain in the system.
- Once a software system is buggy, it will always be buggy. If you test it and find a few defects, and fix them, then you will have buggy code with patches.

## The Jelly Bean Model




## The Jellybean Model

- "Trying to find the defects in a software system by testing is a lot like trying to find the blue jellybeans in a big jug of red jellybeans by taking samples." - Presenter at a NASA Space Station Progress Meeting.
- In fact, the mathematics of defect removal by testing are exactly the same as hunting for the blue jellybeans.

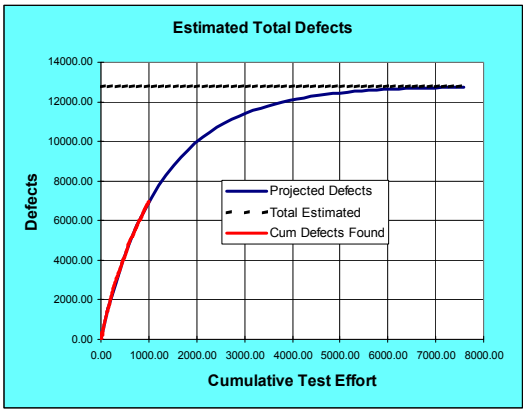
## The Jelly Bean Model

- Consider a large bin containing 450,000 jelly beans. 12,000 are blue (representing defects) and the rest are red. The bin is continuously mixed.
- Our task is to remove as many of the blue jelly beans as we can and replace them with red jelly beans.
- The process is assumed to be 30% efficient – on average, 30% of the blue beans in each sample are replaced and the sample returned to the bin.
- The mathematics are based on the execution time reliability model using the Weibull function. The model is implemented in Excel.




## The Jelly Bean Model – 300% Coverage

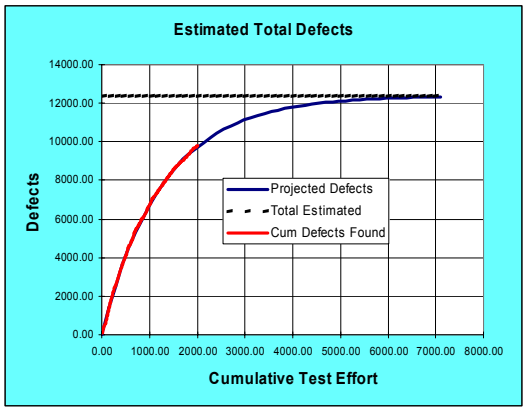
- 500 samples are withdrawn from the bin and an average of 30% of the blue jelly beans are removed from each.
- Average sample size 2700 beans.
- Number of jellybeans processed 1,350,000 (300%)




Percent Removed = 60%
Percent Coverage = 300%

## The Jelly Bean Model – 600% Coverage

- 500 samples are withdrawn from the bin and an average of 30% of the blue jelly beans are removed from each.
- Average sample size 5400 beans.
- Number of jellybeans processed 2,700,000 (600%)

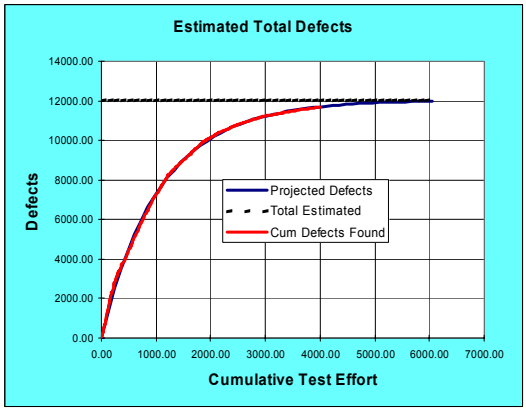



Percent Removed = 82%
Percent Coverage = 600%




## The Jelly Bean Model – 1200% Coverage

- 500 samples are withdrawn from the bin and an average of 30% of the blue jelly beans are removed from each.
- Average sample size 10800 beans.
- Number of jellybeans processed 5,400,000 (1200%)




Percent Removed = 97%
Percent Coverage = 1200%

## The Jellybean Model

- Discovering a significant fraction of the defects (e.g. 90%) in an integrated system by test is not economically feasible.
- In practice, what fraction of the remaining defects can economically be discovered by test?



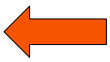

## The Trouble With Testing

**Finding the problems**
**Fixing the problems**
**Reliability**

## The Problems With Testing

- The best system test program can be expected to find about 55% of the remaining defects – Capers Jones.

| Activity | Effectiveness Range |
|---|---|
| Design Review | 25% to 40% |
| Design Inspection | 45% to 65% |
| Code Review | 20% to 35% |
| Code Inspection | 45% to 70% |
| Unit Test | 15% to 50% |
| Functional Test | 20% to 35% |
| Regression Test | 15% to 30% |
| Integration Test | 25% to 55% |
| System Test | 25% to 55% |
| Low Volume Beta Test | 25% to 40% |



Capers Jones

- Few test programs are "the best" and many test programs find as few as 10% of the remaining defects.




## The Problems With Testing – CO$T

- Real data from a real project

**Summary of the Results**

Data Available to ..... 10/27/98
Total PTS reports To Date ..... 5647
Estimated Test Effort to Date ..... 4433.00 (inferred Person Days)

5647/4433 = 1.27 defects/day

| Analysis Estimates | Total Release Defects | Effort to Find 100% * | Effort To Find 50% | Effort for Desired Quality | Fraction Found To Date |
|---|---|---|---|---|---|
| From Quadratic Regression ..... | 12452 | 17132.4 | 5018.0 | 12887.1 | 0.45 |
| From Weibull Regression ..... | 25084 | 97375.0 | 12303.5 | 63697.9 | 0.22 |

| Estimated Phase Distribution | Total Phase PTS Count | Total Real Problems | Critical Problem Count | Major Problem Count | Estimated Test Effort | Remaining Estimated Test Effort |
|---|---|---|---|---|---|---|
| ET PTS Reports ..... | 1133 | 683 | 54 | 261 | 773.0 | 0.0 |
| DV PTS Reports ..... | 2268 | 1388 | 97 | 531 | 1764.0 | 0.0 |
| IST PTS Reports ..... | 1665 | 1122 | 101 | 360 | 1421.0 | 0.0 |
| CI PTS Reports ..... | 573 | 500 | 53 | 153 | 468.0 | 0.0 |
| Estimated Defects Remaining .... | 19446 | 12717 | 1050 | 4494 | | |

| Distribution Statistics | Phase Percent Of Total | Phase Percent Real | As a Percent of Real: Percent Critical | As a Percent of Real: Percent Major | Phase Effectiveness (Percent) |
|---|---|---|---|---|---|
| ET PTS Reports ..... | 4.5 | 60.2 | 7.9 | 38.2 | 5.5 |
| DV PTS Reports ..... | 9.0 | 61.1 | 6.9 | 38.2 | 9.4 |
| IST PTS Reports ..... | 6.6 | 67.3 | 9.0 | 32.0 | 7.6 |
| CI PTS Reports ..... | 2.2 | 87.2 | 10.6 | 39.6 | 2.8 |
| Estimated Defects Remaining ... | 77.5 | 65.3 | 8.2 | 35.3 | |

* Weibull effort to find 99.5% of Total Estimated Defects.

- Note that the effectiveness of System Test (DV) was 9.4%.

## The Problems With Testing – CO$T

- There are two major costs of relying on testing
  - Cost of testing
  - Cost of repair
- Using the Jellybean example, removing 55% of the defects means finding 6600 defects (jellybeans).
  - Based on real projects, the rate of problem discovery averages between 0.5 and 2.5 defects/person-day of testing (use 1.25 defects/person-day)
  - Many "defects" discovered in a test program are duplicates, non-reproducible or not real problems. These amount to about 35% of "defects" found. To find 6600 real defects, testing must generate 10,150 PRs.
  - 10,150 PRs requires about 12,960 person days of testing (53 person years). At a loaded cost of $150,000 per person-year, this represents a cost of $7,933,000.
- The expenditure of $7,933,000 adds nothing to the software and the cost comes straight from company's the bottom line.




## The Problems With Testing – CO$T

- Finding a problem does not fix it.
  - About 45% of real problems are serious enough that they **must** be fixed. Thus 2970 defects must be repaired.
  - Average repair effort is between 5 and 10 person-days. (Use 5 days per problem)
  - Repairing 2970 defects thus requires 14,850 person days (61.9 person years)
  - At a loaded cost of $150,000 per person year this is $9,281,000.
- This cost adds nothing to the software and it too comes directly from the company's bottom line.
- Removing 55% of the serious defects remaining at the start of testing has cost $17,214,000. This could be why 55% is an upper limit.

## The Problems with Testing - Reliability

- Reliability varies inversely as the number of defects remaining
  - $TF_m = C/N$
  - Removing 55% of the remaining serious defects improves the mean time to serious failure by 122%.
  - This level of improvement will not be noticeable to the user.
  - A failure once every 6 hours or a failure every 13.3 hours is the same to the users – they will be unhappy.
- Good example is Microsoft Windows 95 – $TF_m$ ~ 6 hours
- **Testing to improve reliability is futile as well as costly.**
- Good Engineering Practice will minimize the amount of testing done on the integrated system.
- The focus must be on producing perfect parts (routines, functions, modules, object methods).




## The Alternative – Verification Planning

## Objectives of Verification Planning

- The focus is on building components that are as close to perfect as possible.
  - Eliminate defects early in the development life cycle
  - Reduce the inter-phase transfer of defects.
- This presentation will focus on these.
- The process described is a planning process. The underlying processes have to be in place to make this work.
  - Metrics (defects, effort)
  - Root Cause Analysis (significant sample)
  - Defect tracing ( how many defects were caused in the child work product by one defect in the parent?)
  - Requirements specification
  - Architectural design based preliminary and detailed design
  - Adequate design detail
  - Configuration Management




## Basic Principles

**- Constant Defect Discovery Rate**
**- Verification Effectiveness**
**- Reference Project**

## Basic Principles - Defect Discovery Rate

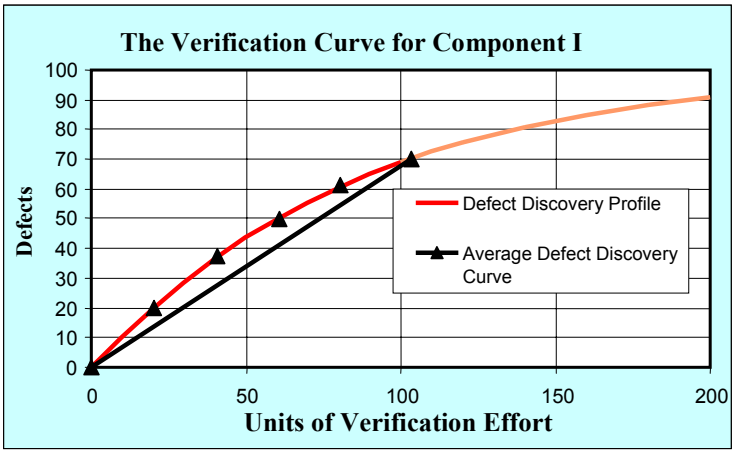
- There are two apparently conflicting principles at work in verification processes.
  - For a given product and a given verification process, the rate of defect discovery is constant.
  - As defects are removed from a work product, defects become less plentiful and the rate of discovery declines.
- How can these be reconciled?




## Basic Principles - Defect Discovery Rate

- Consider a software system that is made up of N component parts. Each part was produced using the same process. Each part has a similar defect density (defects/unit size) and a similar size.
- Consider the application of a verification process to one of the N components.

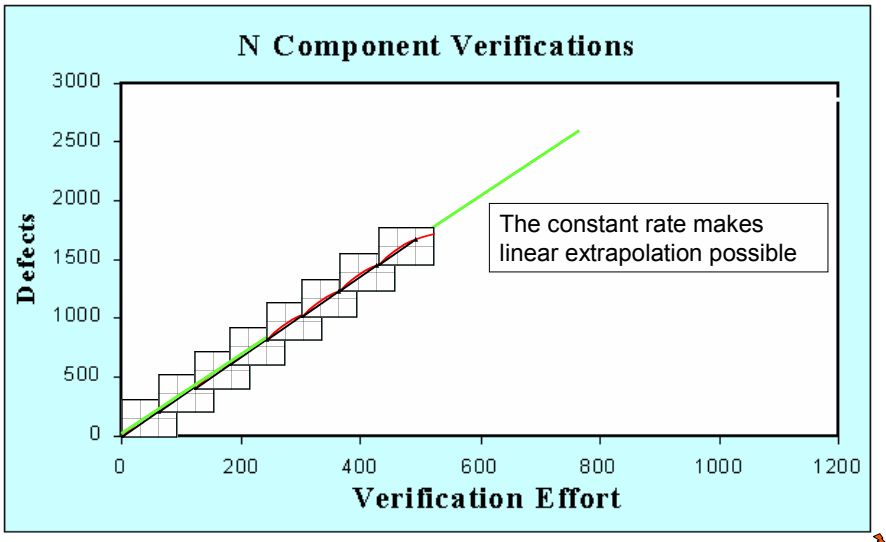
## The Average Defect Discovery Rate




- For each component, there is an average rate of defect discovery over the course of the verification process.

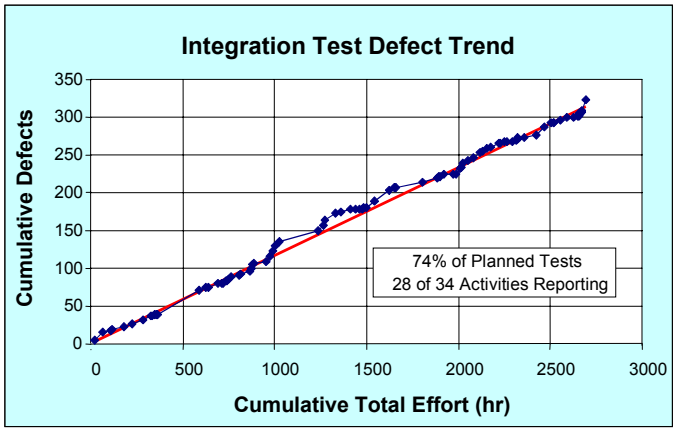



## Combining Results of N Components



## Real Example

- In reality the size and defect density vary about mean values.
- The result is a noisy straight line.







## Basic Principles - Verification Effectiveness

- Verification Effectiveness (VE) is the ratio of the number of defects detected by a verification process to the number of defects presented.

| Activity | Effectiveness Range |
|---|---|
| Design Review | 25% to 40% |
| Design Inspection | 45% to 65% |
| Code Review | 20% to 35% |
| Code Inspection | 45% to 70% |
| Unit Test | 15% to 50% |
| Functional Test | 20% to 35% |
| Regression Test | 15% to 30% |
| Integration Test | 25% to 55% |
| System Test | 25% to 55% |
| Low Volume Beta Test | 25% to 40% |

Capers Jones

- Verification Effectiveness (VE) is the controlling factor in the planning process. All other factors are estimates or historical data.

## Basic Principles -The Reference Project

- The reference project is a completed project for which sufficient data has been collected for use in planning.
- The reference project should be as similar to the new development project as possible.
- The planning process involves “scaling” the reference project parameters to the new project.
- The basic defect estimating function used is

  Defects = S*d + N*F

  Where S = work product size

  d = defect density due to new errors made creating the WP

  N = defects in parent when it was used

  F = defect multiplier (bug breeding factor)




## Reference Project and the Planned Project

| | | Historical Data | | | The Plan | | |
|---|---|---|---|---|---|---|---|
| Work Product | Process | Starting Defects | VE | Defects Left | Starting Defects | VE (Plan) | Defects Left |
| Requirements | Inspection | 590 | 0.4356 | 333 | 590 | 0.7 | 177 |
| Design | Inspection | 1330 | 0.3436 | 873 | 1145 | 0.6 | 458 |
| Code | Inspection | 3712 | 0.0685 | 3458 | 3006 | 0.6 | 1202 |
| Code | Unit Test | 3458 | 0.0628 | 3241 | 1202 | 0.45 | 661 |
| Code | Integration Test | 3241 | 0.1449 | 2772 | 661 | 0.3 | 463 |
| Code | System Test | 2772 | 0.1264 | 2421 | 463 | 0.2 | 370 |

- The reference project is analyzed to obtain the estimating parameters.
- The estimating parameters are used to produce a Verification Plan.

| Work Product | Size | d | F |
|---|---|---|---|
| SW Requirements | 914 pages | 0.6455 | N/A |
| SW Design | 2342 pages | 0.3975 | 1.2 |
| Program Code | 400 KLOC | 0.00557 | 1.7 |

- The objective is to reduce the delivered defects by 85%.

## The Planning Model

**Effort Adjustment Factors**



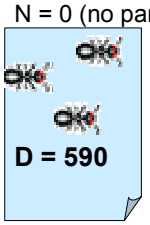

## The Planning Model

- The planning model is based on reference project discovery effort data with adjustments for the changes in the number of defects to be discovered.
- There are two effort adjustment factors used:
  - $\mathbf{A_d}$ (effort adjustment for changes in the defect density)
  - $\mathbf{A_p}$ (effort adjustment to reflect changes in planned defect removal)
- Both adjustments are based on a reference defect discovery profile (Weibull Function).
- The reference discovery profile is used to calculate Effort Adjustment Factors for each verification process in the project.
- The reference discovery profile is used for scaling purposes only.

## Defect Insertion and Removal - Requirements

**Requirements**

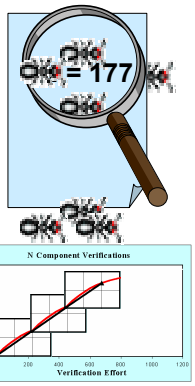
S = 914 pg
d = 0.6455 def/pg
F = 0 (no parent)
N = 0 (no parent)




**Analyst**

**Inspection**

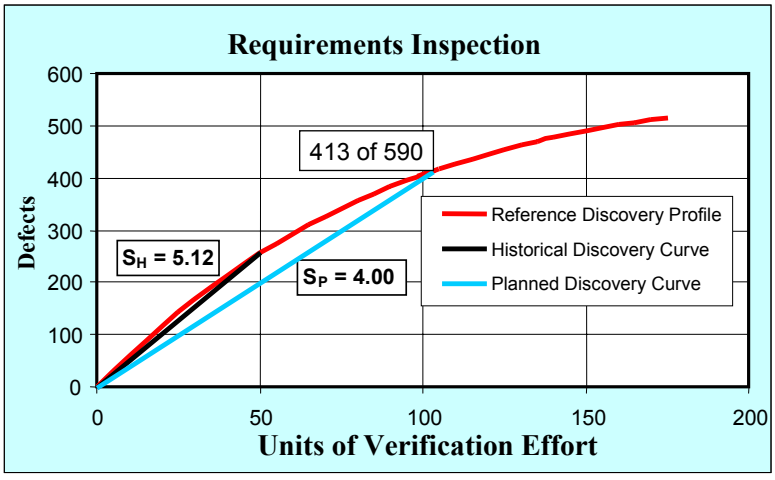
VE = 0.7




413 Defects

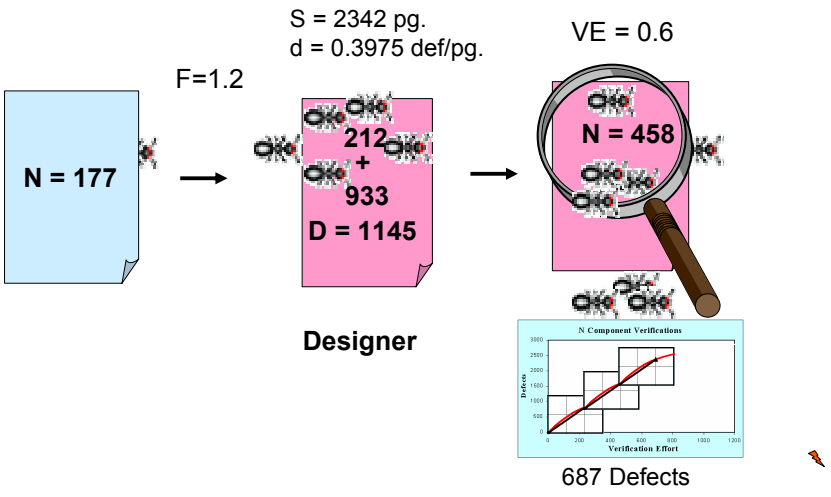



## Requirements Inspection Effort Adjustment




**Effort Correction $A_P$ = 5.12/4.00 = 1.28**

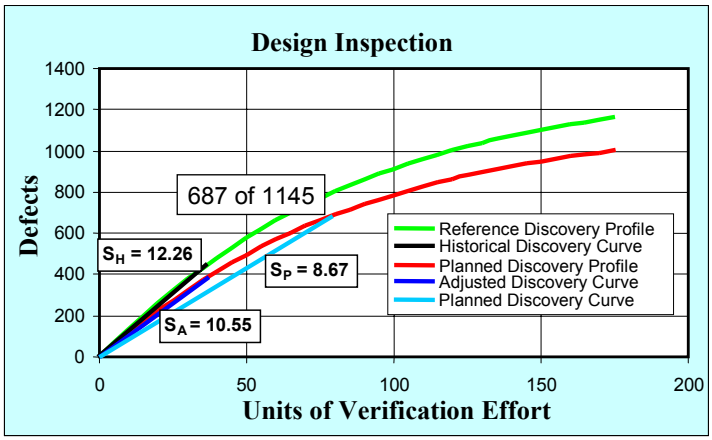
## Defect Insertion and Removal - Design







## Design Inspection Effort Adjustment



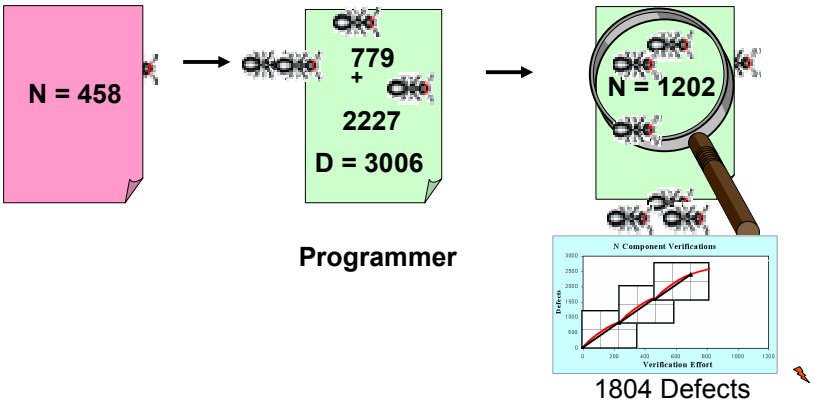

**Effort Correction $A_d$ = (12.26/10.55) = 1.16**

**Effort Correction $A_P$ = (10.55/8.67) = 1.21**

## Defect Insertion and Removal - Code

**Design** **Code** **Inspection**

F=1.7 S = 400 KLOCs. d = 5.57 def/KLOC VE = 0.6

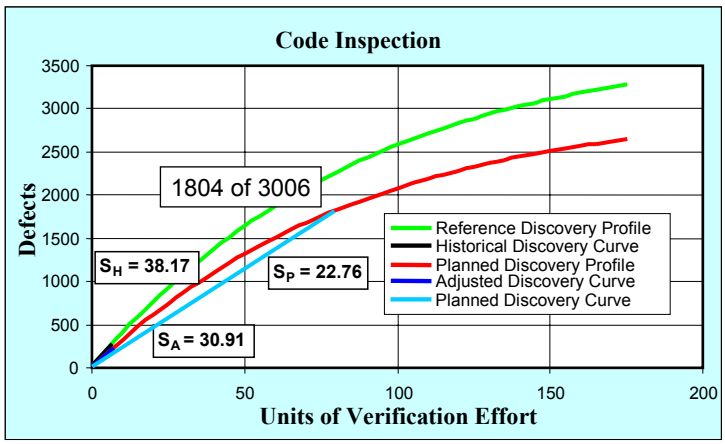



**Programmer**

1804 Defects




## Code Inspection Effort Adjustment




**Effort Correction $A_d$ = (38.17/30.91) = 1.23**

**Effort Correction $A_P$ = (30.91/22.76) = 1.36**

## Unit Test

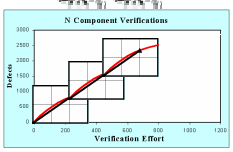
**Inspected Code**

**Unit Test**

VE = 0.45

**N = 1202**

**N = 661**

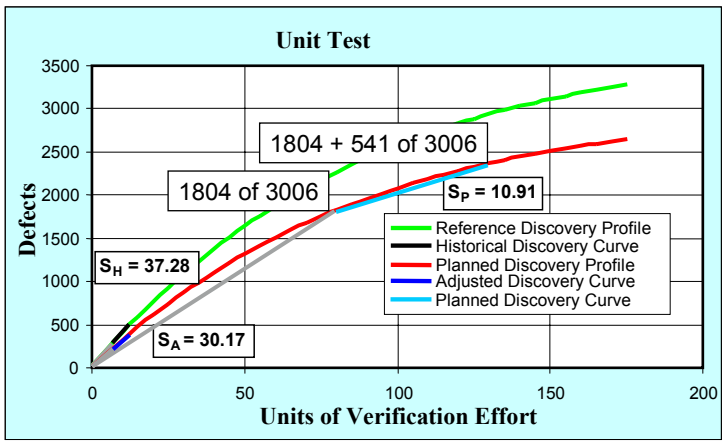



541 Defects





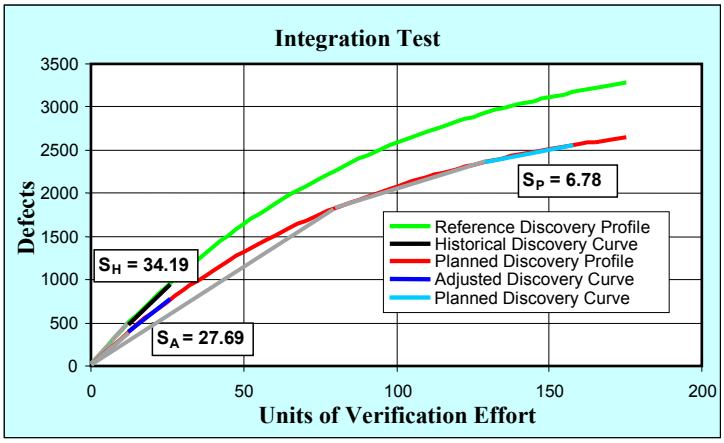
## Unit Test Effort Adjustment




**Effort Correction $A_d$ = (37.28/30.17) = 1.23**

**Effort Correction $A_P$ = (30.17/10.91) = 2.77**

## Integration Test Effort Adjustment

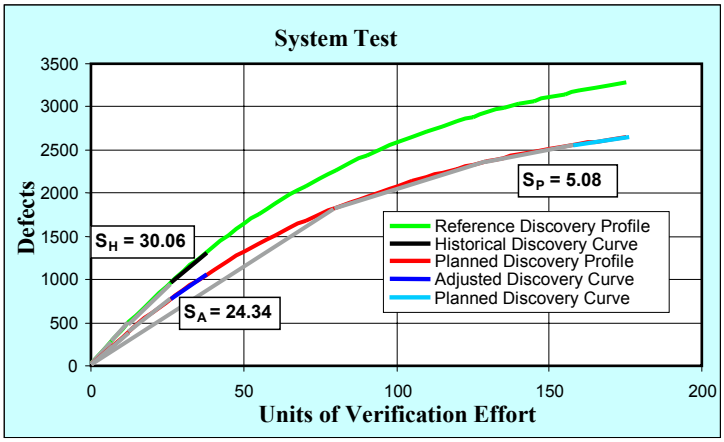



**Effort Correction $A_d$ = (34.19/27.69) = 1.23**

**Effort Correction $A_P$ = (27.69/6.78) = 4.08**




## System Test Effort Adjustment




**Effort Correction $A_d$ = (30.06/24.34) = 1.23**

**Effort Correction $A_P$ = (24.34/5.08) = 4.79**

# Planned Verification Effort by Process

- Effort Calculation
- Comparison with Reference Project




## Planned Verification Effort Calculation

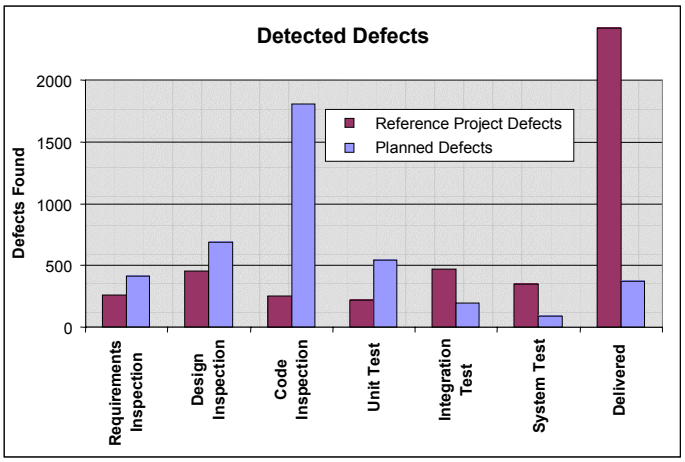
- The unit defect discovery effort data from the reference project is adjusted using the adjustment factors.
- The total effort is calculated using the adjusted unit defect discovery effort values and the planned number of defects.

| Process | Reference Project Unit Effort (Hr/Def) | Planned Removal Effort Ajustment $A_p$ | Defect Density Effort Adjustment $A_d$ | Overall Effort Ajustment Factor A | Planned Defects | Estimated Effort (hr) |
|---|---|---|---|---|---|---|
| Requirements Inspection | 1.50 | 1.278 | 1.000 | 1.278 | 413 | 792 |
| Design Inspection | 3.10 | 1.214 | 1.162 | 1.411 | 687 | 3004 |
| Code Inspection | 4.80 | 1.358 | 1.234 | 1.676 | 1804 | 14511 |
| Unit Test | 7.28 | 2.768 | 1.234 | 3.416 | 541 | 13453 |
| Integration Test | 10.64 | 4.084 | 1.234 | 5.040 | 198 | 10617 |
| System Test | 17.50 | 4.790 | 1.234 | 5.911 | 93 | 9620 |
| **Total Estimated Effort (hr)** | | $4,050,000 | 27 person years | | | 51997 |

- Total Reference Project effort was 15735 Hrs (8.19 man yrs–$1,335,000).

## Change from Reference Project - Defects

- But more importantly, far fewer defects will be delivered.

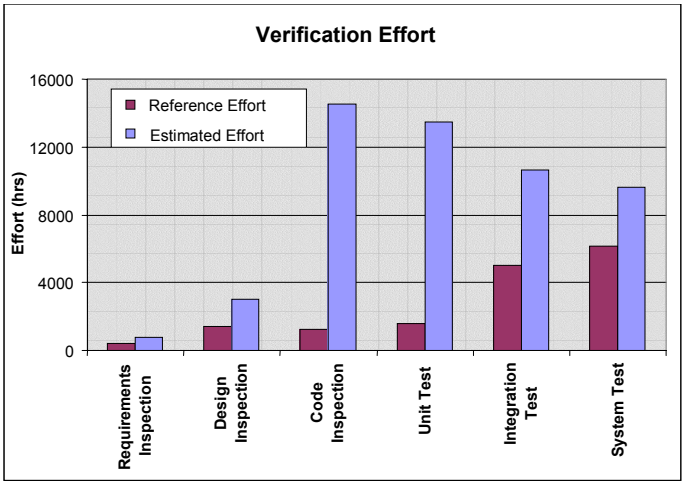






## Change from Reference Project - Effort

- System test and Integration test effort can be reduced.



## Using the Planning Data

- Where size is known
- Where size is not known



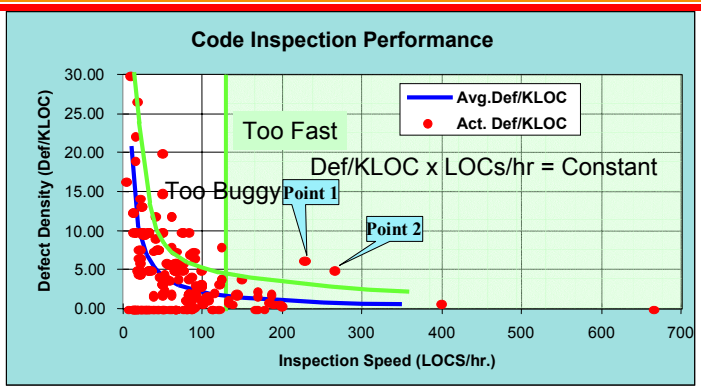

## Using the Planning Data - Size is Known

- The average verification speed is known for Inspections and Unit Test.

| Process | Deliverable Size | Estimated Effort (hr) | Verification Speed |
|---|---|---|---|
| **Requirements Inspection** | 914 pg | 792 | 1.1540 |
| **Design Inspection** | 2341 pg | 3004 | 0.7793 |
| **Code Inspection** | 400 KLOC | 14511 | 0.0276 |
| **Unit Test** | 400 KLOC | 13453 | 0.0297 |

- This data can be used to plan individual inspections and tests.
- E.g. Unit test of 300 lines of code requires 300/(1000 *0.0297) = 10.1 hours of testing.

## Using the Planning Data - Size is Known




- The acceptance criteria can be applied at the component level.

| Measure | Average | Point 1 | Point 2 |
|---|---|---|---|
| Size (LOCS) | | 800 | 800 |
| Inspection Rate(Locs/Hr) | 78 | 229 | 267 |
| Defect Rate(Def/Hr) | 0.206 | 1.43 | 1.33 |
| Defect Density (Def/KLOC) | 3.65 | 6 | 5 |




## Using the Planning Data - Size is Not Known

- When size is not known, as in Integration Test and System Test, enough test cases are required to consume the planned effort.
- The effort required to execute the tests with no failures (Zero Defect Test Program) sets the lower limit.
- If enough test cases cannot be produced, then the planned effort must be revised.

## Discussion

- The sample plan represents a simplified process – only one design phase.
- The target number of delivered defects was arbitrary – 85% reduction in delivered defects. 90% => 10 times improvement in reliability. 99% => 100 times improvement.
- The plan represents a first pass at planning. It can be revised to reduce the cost. For example, increasing code inspection and unit test effort would allow reduced integration and system test effort.
- The system test effort can be reduced to the level required to establish the slope of the defect-effort curve. This tells the test team how well the software was developed.



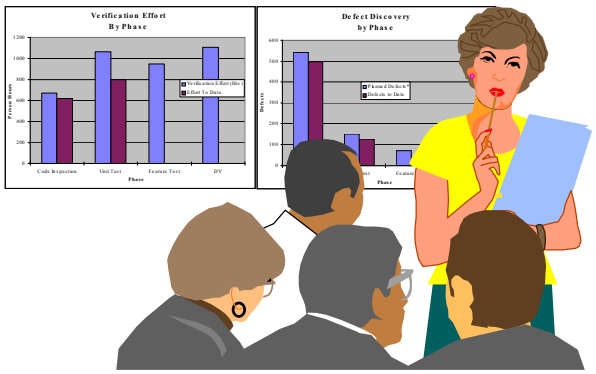

## Final Points

## Final Points

- All design staff must receive training in the planning process or they will not support the plans.
- An effective Configuration Management system is a major aid in ensuring the plan execution.
- The plan must be maintained and revised as real data arrives and the project scope changes.
- Rework effort estimates can be made and included in the development plan.
- Reduction in delivered defects will result in reduced field maintenance and happier customers.
- The data collected to track the plan and support future projects is valuable for project post mortems.
- The planning model can be used to resist schedule compression.




## Final Points




Make the results Visible!

## Conclusions – Quality and Testing

- The developers who built the software system, not the back end test team, are responsible for the quality of the delivered product.
- Management is responsible for providing the tools, training and process to allow the developers to do their job well.
- Back end testing is not an effective way of improving reliability – attempting to do this is both futile and costly.
- Testing is the Anti-Quality Process




## Next Stop – India!

accenture
Technology Solutions

Home / Work / Environment / Skills / Search Jobs

Digital Forum / accenture.com / Contact Us / Sitemap

Print | E-mail

**SEPG/SQA Professionals**

Accenture Technology Solutions is a global company of technology specialists who build, deploy and maintain technology solutions for Accenture clients. As part of our team in India, you'll be working with the latest software and leading-edge development tools - giving you the opportunity to build specialist skills and expertise on the job. We need people who thrive on technology challenges. People who can help turn innovative ideas into effective results.

We are looking for candidates with experience in implementing CMM / CMMi in a Level 4 or Level 5 Company. You should have experience in SEPG, Quality Assurance and audits, collation and analysis of metrics. Preference would be given to applicants with CSQA certification. Candidates with a mix of delivery and quality systems experience shall also apply.

**Your competition is improving!**
**What about you?**

- Hyderabad

Individuals will be considered in the countries in which they currently have long-term work authorization.

# Questions?