

# Improving Your Quality Process

A Practical Example

John Balza  
johnjbalza@comcast.net

# Objectives

To provide an example of how to improve your software quality process

## Agenda:

- Initial Situation
- Defect Analysis Targets Improvements
- Justify the Investment
- Creating a Test Model
- Results

## HP-UX Statistics/Organization

- Approximately 18 Million lines of code total
- New or changed code
  - 1.4 Million in 11.00
  - 3.0 Million in 11.11
  - 6.0 Million in 11.23
- HP-UX Code is developed by ~ 1500 people in 13 labs in 7 geographical locations.
- Each Lab responsible for
  - maintaining existing code
  - developing new functionality
  - quality of their code.
- One lab responsible for end to end quality
  - System testing, solution testing, beta test

# Customer Situation

Results from Interex Engineering Investment Survey, 1999 (HP user group)

- Most important strategic directions for HP in next 5 years
  - 1. Keeping customer costs down
  - 2. Developing higher quality software

# HP-UX Quality Improvement Program

**“10X in 5 Years”**

GOAL: Decrease customer found defects by a factor of 10

## Internal Goals

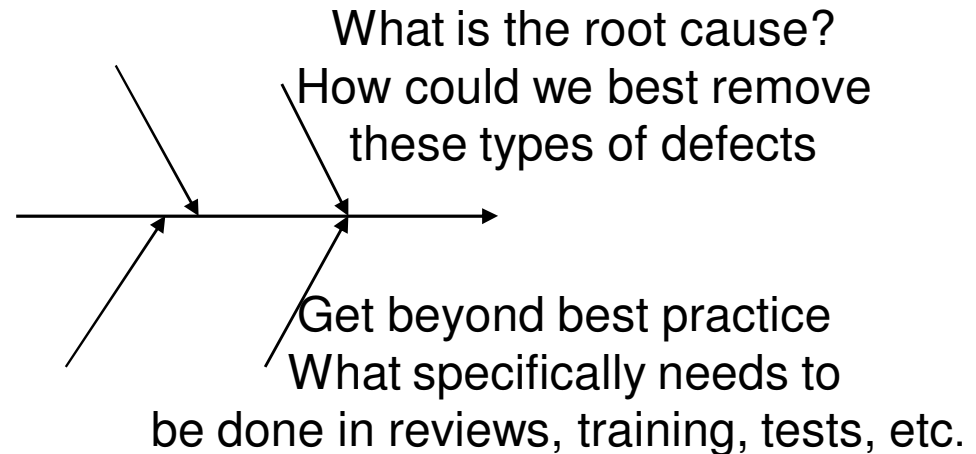
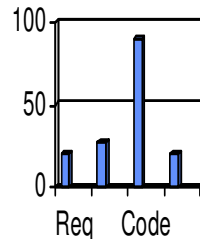
- Drive up the defects prevented or found before check-in
  - Reduce defects found after check-in by 2x every 18 months
- Increase the defect removal rate between check in and customer release
  - Have our system testing better reflect our customers environments

# Defect Analysis

Know what defects are escaping and remove  
the causes

# Required a Defect Analysis

What are the most common types of defects escaping?



Required a defect analysis on all defects that had escaped a development team.

Typical recommendations:

- HP-UX Internals training
- Better communication of dependencies with other teams
- Specialized peer review for 'locking' issues
- Testing improvements

# Testing Themes

From defect analysis we found several common themes:

- Over 60% of our defects were found in the ‘core’ of the operating system (networking, kernel, commands, libraries)
- Over 50% of escaped defects were ‘functional defects’
- Another 10% were ‘configuration defects’ – problems that would occur only in certain configurations, typically high end SPUs

Developers only found a fraction of their defects

- 50% of defects were found in testing by other teams (25% by system test)
- 12% were found by customers

## ROI Justification

Provide management with a solid cost-saving argument using your or industry data

## Defects injected per KNCSS

Phase	Defects Introduced
Requirements	10.1
Design	12.8
Code	17.2
Documentation	4.0
Bad Fixes	5.0
Total	49.1

Taken from Software Quality, Analysis and Guidelines for Success  
Capers Jones, 1997. Pp 138-140 (10,000 FPs size projects)

## Defect removal efficiencies

<u>Removal Steps</u>	Lowest Efficiency	Modal Efficiency	Highest Efficiency
Requirements inspection	25%	35%	50%
Informal design reviews	25%	30%	40%
Formal design inspections	45%	65%	85%
Informal code reviews	20%	30%	45%
Formal code inspections	45%	65%	85%
Code desk check	20%	40%	60%
Unit Testing	15%	30%	55%
New Function Testing	20%	30%	40%
Regression Test	10%	15%	30%
Integration test	25%	35%	45%
Performance Test	15%	20%	30%
System Test	25%	35%	55%
Beta Test <10 sites	20%	30%	40%

Taken from Software Quality, Analysis and Guidelines for Success Capers Jones,

# Defect Detection Costs: hours/defect

Requirements inspection	~18
Informal design reviews	~18
Formal design inspections	~10
Informal code reviews	9
Formal code inspections	15
Code desk check	9
Unit Testing (by developer)	7.5
New Function Testing	12.5
Regression Test	5
Integration test	~15
Performance Test	~15
System Test	15
Beta Test <10 sites	10

Data from our peer  
review DB, Jan 1, 1998 to Nov 1  
1999

---

Data from Capers  
Jones, Applied  
Software Management  
1991

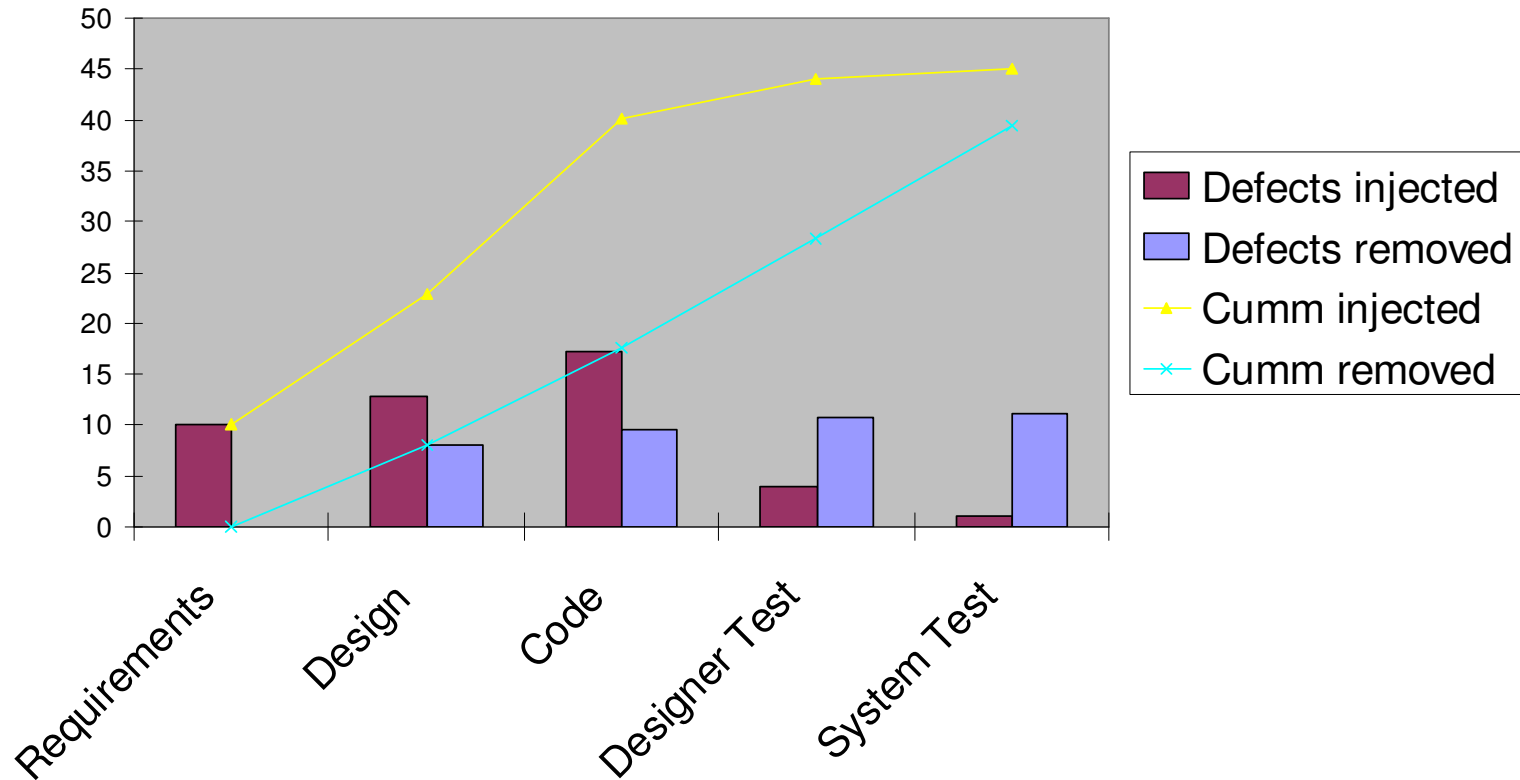
# Defect Fix Costs: hours/defects

Inspections/reviews	.5	Internal Data
Code desk check	~.5	
Unit Testing (by developer)	2.5	Industry data
New Function Testing	5.0	
Regression Test	12	Internal Data
Integration test	12	
Performance Test	20	Various internal estimates
System Test	20	vary from 20-40 hours
Beta Test <10 sites	~20	
Customer reported defect (fix and patch once)	40	

# Our Typical Defect Removal

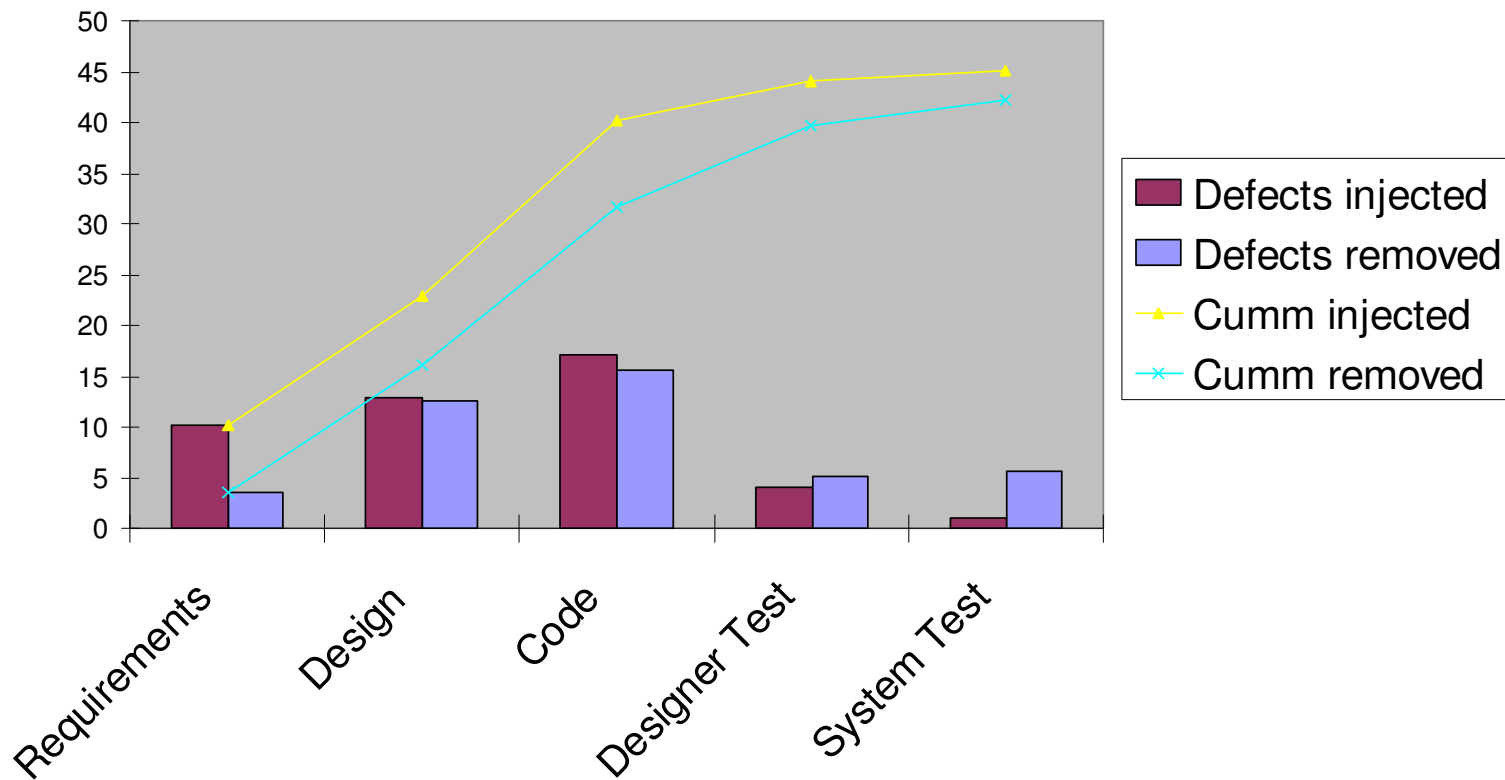
with design walk through, code desk check, tests for new functionality and regression system tests

Today's typical practice  
Customer sees 5.7 defects/KNCSS  
Total cost 619 hours/KNCSS



# Improvement With Inspections

Add in rigorous inspections  
Customer sees 2.8 defects/KNCSS  
Total Cost 356 hours/KNCSS



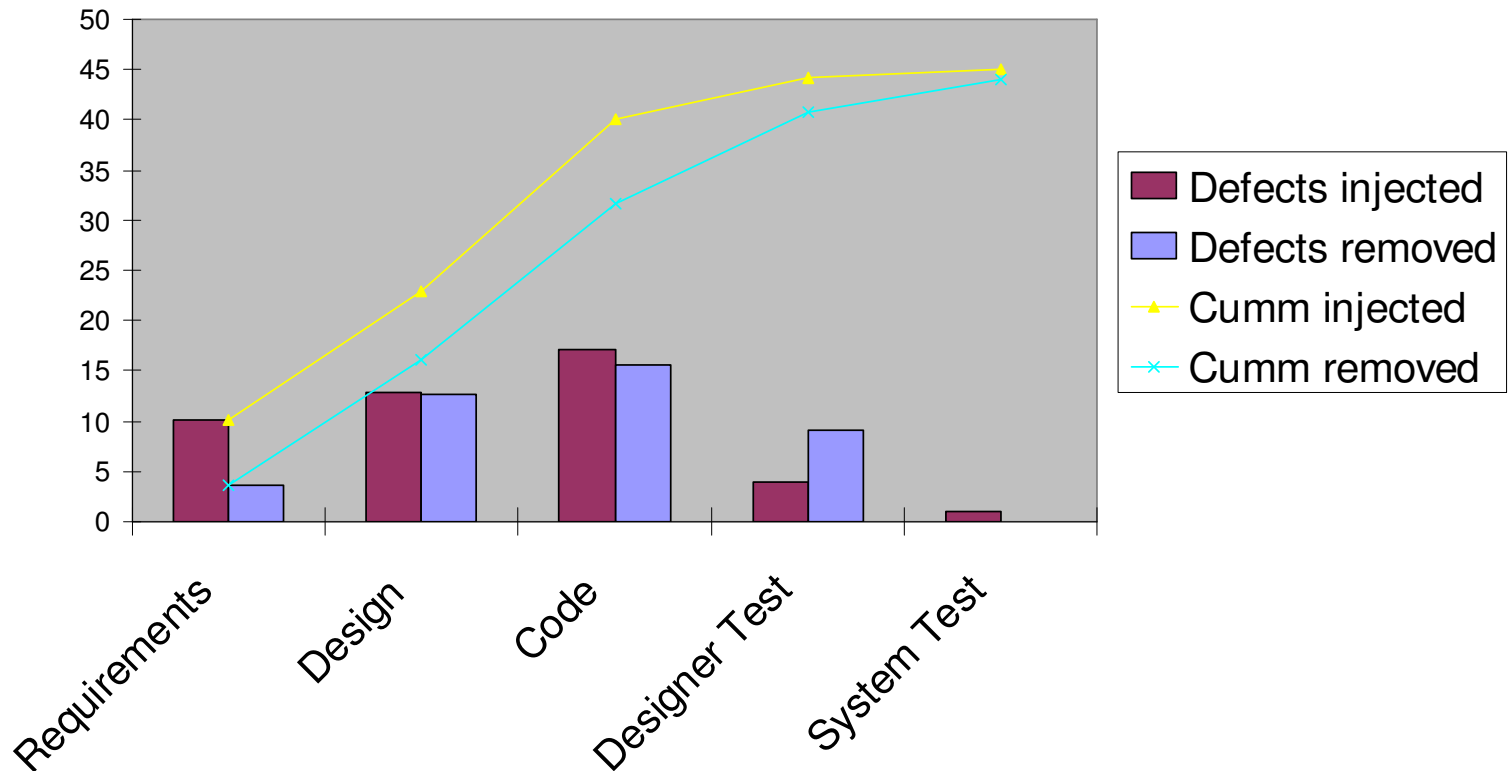
2X improvement with 42% less labor

## Inspections/Peer Reviews

- Retrained every engineer on the Inspection Process
- Set-up a database to record the peer review results
- Required inspections of all design documents and some form of peer review on all code
- Started development of checklists for the peer reviews
- Many labs established a Plan-Do-Check-Act cycle with quarterly reviews by management

# With Inspections & More tests

Inspections, Unit, Subsystem Integration, Beta Test  
Customer sees 1.0 defect/KNCSS  
Total Cost: 304 hours/KNCSS



Another 3X quality improvement with 15% less labor

## Changed Internal Goals

Combination of defect analysis and this ROI model resulted in management endorsement of new internal goals

- Development team finds 90% of their own defects (70% through peer review, 20% through testing)
- Partner/System Test finds 90% of remaining defects
- Customer only finds 1%

## Creating a Test Model

Create a process model: the current model  
and target model

# Original Test Model

Tests of subsystem

White Box  
Test

Subsystem  
Test

Tests of entire product

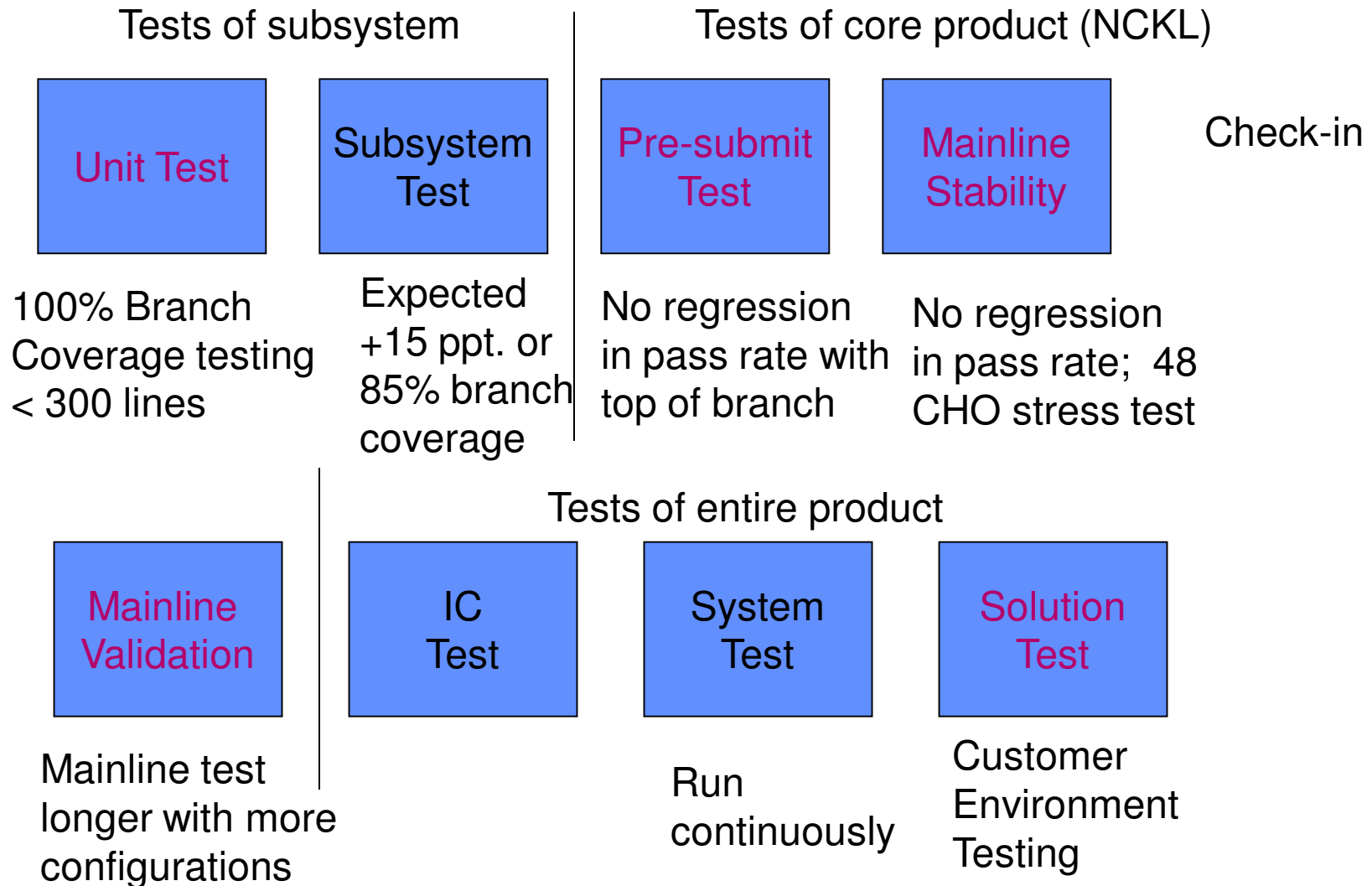
IC  
Test

System  
Test

Solution  
Test

No criteria	>95% pass on Functional test (usually on 1 SPU)	>95% pass on functional test on a couple configurations  Run stress tests on small configuration	>95-99% pass on functional test (more varied and larger config)  96 continuous hours of operation on stress tests	0 defects running with key layered applications
-------------	---	--	---	---

# Target Model



## Example: Subsystem Test

Definition	Testing conducted to verify the complete functionality of a subsystem.
Objective	Expose defects in the subsystem, particularly interactions between components. Test against the External Specification.
Strategy	Test the entire subsystem for correct functional and internal behavior. These tests should be automated to form a permanent regression test to ensure compatibility.
Metric	Functional coverage (have all external interfaces been tested?) and branch coverage (condition/decision coverage in C-Cover terms) with automated tests of new and legacy functionality.
Entry criteria	Test plan and test specifications developed, reviewed and in the configuration management system.
Exit criteria	100% tests passed and 100% functional coverage, 85% branch coverage metrics with acceptable exceptions. Document in the defect tracking systems those items that need to be corrected later. Test plan and test specifications updated code coverage
Roles and Responsibilities	Automated test creation and analysis owned by subsystem development team.

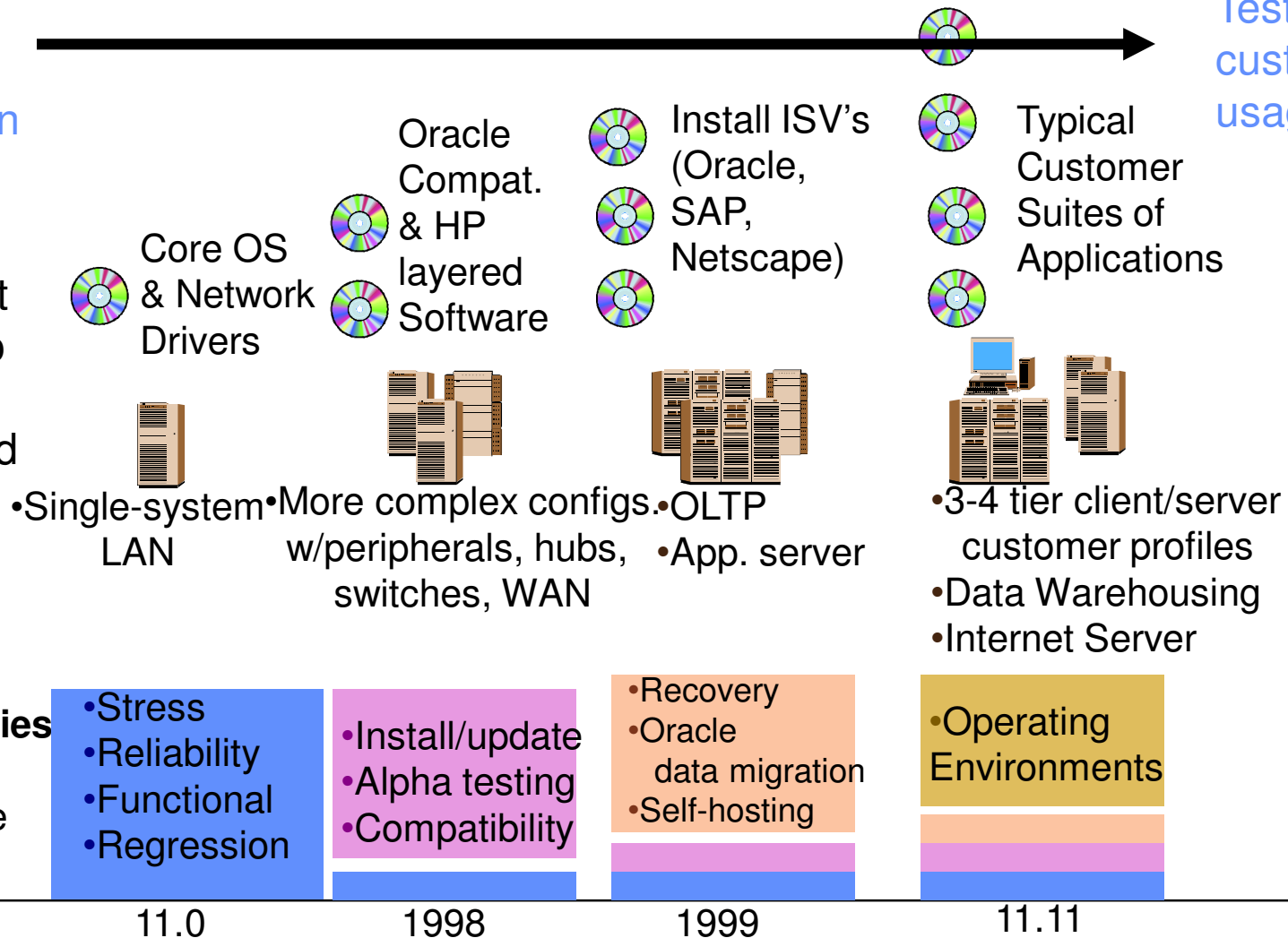
# Solutions Focused Testing

Test of generic system specification

Test of customer usage

**Software Objective:**  
Expand test coverage to include ISU/ISV and customer apps.

**Test Methodologies Objective:**  
Add/enhance tests

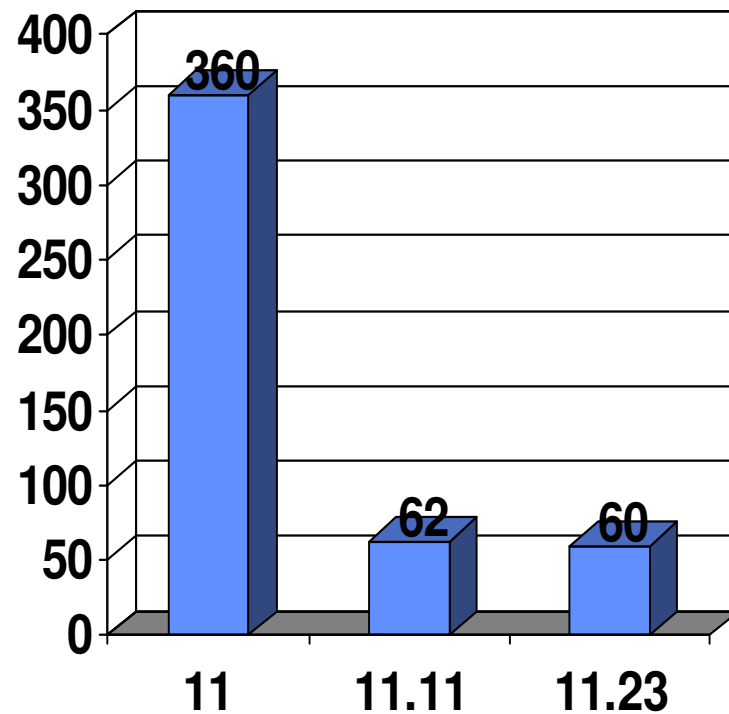


# Results

# Customer Defects

- 6X reduction in customer defects from 11.00 release to 11.11.
- At the same time, each release was twice as complex as the last.

Customer Defects in first year



## Internal Goals

- Today, developers find over 90% of their own defects, most of this through inspections and peer reviews.
- On the 2<sup>nd</sup> release we met our 1% customer found goal.



## Productivity Increase

Because our quality increased throughout our lifecycle, our productivity has also increased:

- 30% more lines of code/engineer year
- Reduction in current product engineering from 20% of staff to 13%
- Backend testing time has been reduced by 25%.

# Lessons

- Start with implementing a formal inspections process.
- Use defect analysis to pinpoint focus areas for inspections and what testing needs to be improved
- Use your own or industry data to show the ROI – because customer defects are so expensive to fix, you can afford quite a lot of testing and still save money.
- Build a test strategy where each stage of testing is well defined in terms of purpose, objectives, and metrics
- Improving quality actually improves productivity
- You reach true success when each team in the organization is asking how they could have found a defect earlier and making those process changes.

# Questions?

- Parallel Talk about how we used measures to track quality as easily as schedule plus other papers and talks at <http://sites.google.com/site/johnjbalza/Home>

Email: [johnjbalza@comcast.net](mailto:johnjbalza@comcast.net)