TSP Secure

Date: December *14*, 2016

William Nichols

© 2015 Carnegie Mellon University

President's Information Technology Advisory Committee (PITAC), 2005

Commonly used software engineering practices permit dangerous errors, such as improper handling of buffer overflows, which enable hundreds of attack programs to compromise millions of computers every year."

This happens mainly because "*commercial software engineering today lacks the scientific underpinnings and rigorous controls needed to produce high-quality, secure products at acceptable cost*."

© 2015 Carnegie Mellon University

U.S. Department of Homeland Security 2006

The most critical difference between secure software and insecure software lies in the nature of the processes and practices used to specify, design, and develop the software . . . correcting potential vulnerabilities as early as possible in the software development lifecycle, mainly through the adoption of security-enhanced process and practices, is far more cost-effective than the currently pervasive approach of developing and releasing frequent patches to operational software.

© 2015 Carnegie Mellon University

Heartbleed

Introduced January 2012 removed April 2014

OpenSSL Open Secure Socket Layer

Unchecked buffer exposed data including passwords

“Heartbleed created a significant challenge for current software assurance tools, and we are not aware of any such tools that were able to discover the Heartbleed vulnerability at the time of announcement” [Kupsch 2014]

At the time, the error was not yet detectable by static analysis tools.

© 2015 Carnegie Mellon University

"goto fail" enabled "man in the middle" attack

Another SSL defect, present from September 2012 to February 2014

A duplicated line of code allowed skipping the final step of the SSL/TLS handshake algorithm

```
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
    goto fail;  /* MISTAKE! THIS LINE SHOULD NOT BE HERE */
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;
err = sslRawVerify(...);
```


© 2015 Carnegie Mellon University

OpenSSL Vul History

OpenSSL CVE Entries

Count of CVE Items Reported

35
30
25
20
15
10
5
0

Reported

Injected

2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016

CVEs
Mean
+1 Std. Dev.
-1 Std. Dev.

© 2015 Carnegie Mellon University

What to do?

DoD mandate to secure software from attack. (section 933 of NDAA and DoD 5000.2)

- DoD Instruction 8500.01 applies Risk Management Framework to the full acquisition life cycle and
- 8510.01 replaces DIACAP with RMF
- This has been implemented by requiring the use of software security assurance (SwA) tools on “all covered systems”.

Why is this hard?

© 2015 Carnegie Mellon University

Security tools and techniques (examples)

Secure Development Practices	Binary/bytecode simple extractor	Host application interface scanner
Warning flags	Focused manual spot check	Web application vulnerabilty scanner
Source code quality analyzer	Manual code review	Web services scanner
Source code weakness analyzer	Inspections	Database scanner
Quality analyzer	Generated code inspection	Fuzz tester
Bytecode weakness analyzer	Configuration checker	Negative testing
Binary weakness analyzer	Permission manifest analyzer	Test coverage analyzer
Inter-application flow analyzer	Host-based vulnerability scanner	Hardening tools/scripts

Take some of these and apply them

© 2015 Carnegie Mellon University

Somewhere in the Software Assurance System Lifecycle

Software Assurance is SE focused on eliminating SW vulnerabilities

Understand SwA Risks | Design Principles | Design Inspection | Secure Coding Practices | Code Inspection | Test Coverage Metrics | Pen Test | Monitor Threats/ Attacks | Respond to Incidents Defects | Upgrade SwA

Requirements Decision

Developmental RFP Decision

Full Rate Production Decision Review

A B C

MDD

Materiel Solution Analysis

AOA

Technology Maturation & Risk Reduction

Engineering & Manufacturing Development

ICD

Production and Deployment

Operation & Sustainment

ASR SRR SFR PDR CDR TRR SVR OTRR IOT&E IOC FOC

Criticality Analysis

Threat Vuln Analysis

SwA Counter-measures

RFP

Eliminate Design Flaws

Sound Arch & Design

Identify Code Flaws

More Secure Code

Identify Test Gaps

Better Test Suites

Vuln Root Cause

Fewer Process Vuln

Effective Threat Response

Cyber Situational Awareness

Cyber Hardened Systems

Software Assurance must span the entire system lifecycle

© 2015 Carnegie Mellon University

Objectives

Provide actionable guidance on how to improve software security assurance throughout the development lifecycle using empirically grounded evidence.

Improves means that the software security assurance must be

- Effective
- Affordable
- Timely

© 2015 Carnegie Mellon University

Composing Software Assurance

However, there "*is inadequate ground truth information to help [DoD] make decisions*", for example the true cost, schedule impact, and effectives of integrating these tools into an SDLC.

Problem:

DoD faces an unfunded Congressional mandate to secure software from attack. This has been implemented by requiring the use of software security assurance (SwA) tools on "all covered systems".

Without empirically validated "ground truth" about cost and effectiveness.

© 2015 Carnegie Mellon University

Solution

- Apply experience and toolkit from quality assurance to security assurance
- Develop a model that predicts the effectiveness and cost of selected SwA tools.
- Use the model and data to help DoD integrate SwA tools into SDLC processes

Tool kit includes

1. A metric framework
2. Quality planning
3. Design
4. Inspections
5. Process composition for additional techniques

© 2015 Carnegie Mellon University

Conjecture that Quality α Security

Quality Assurance:

The planned and systematic activities implemented in a quality system so that quality requirements for a product or service will be fulfilled.

***Security* Assurance:**

The planned and systematic activities implemented in a *security* system so that *security* requirements for a product or service will be fulfilled.

Quality Control:

The observation techniques and activities used to fulfill requirements for quality

***Security* Control:**

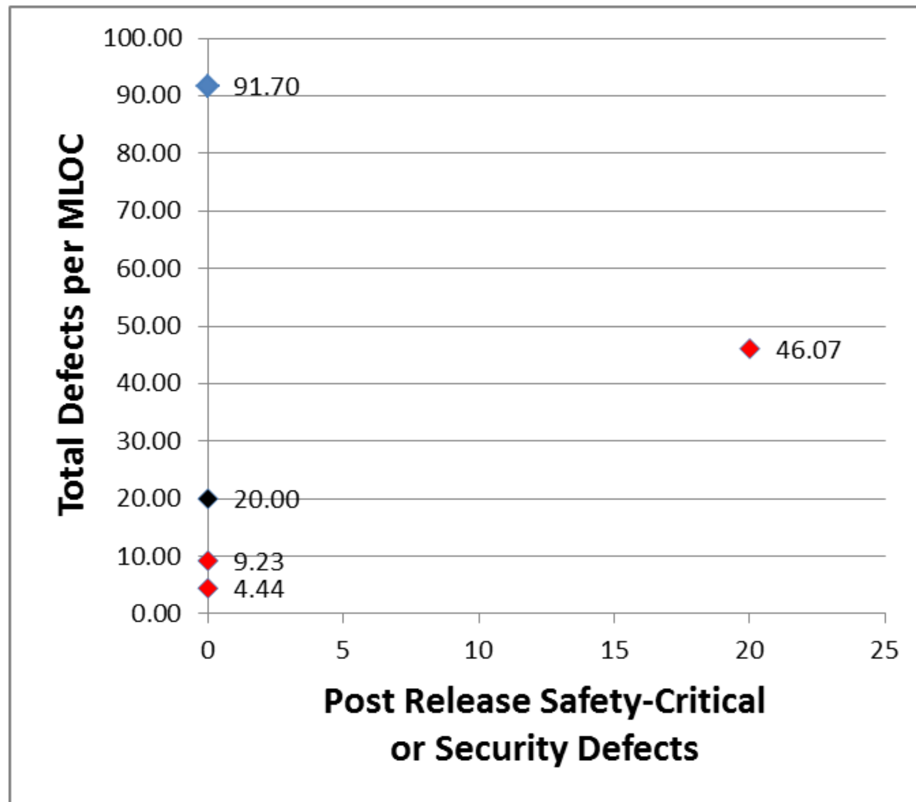
The observation techniques and activities used to fulfill requirements for *security*

We have a lot experience modeling the left column

© 2015 Carnegie Mellon University

Evidence that Quality is a Security issue, and Vice Versa

Rigorous reduction of defects enhances security.



We found high quality software was generally safe and secure

Defective software is NOT secure

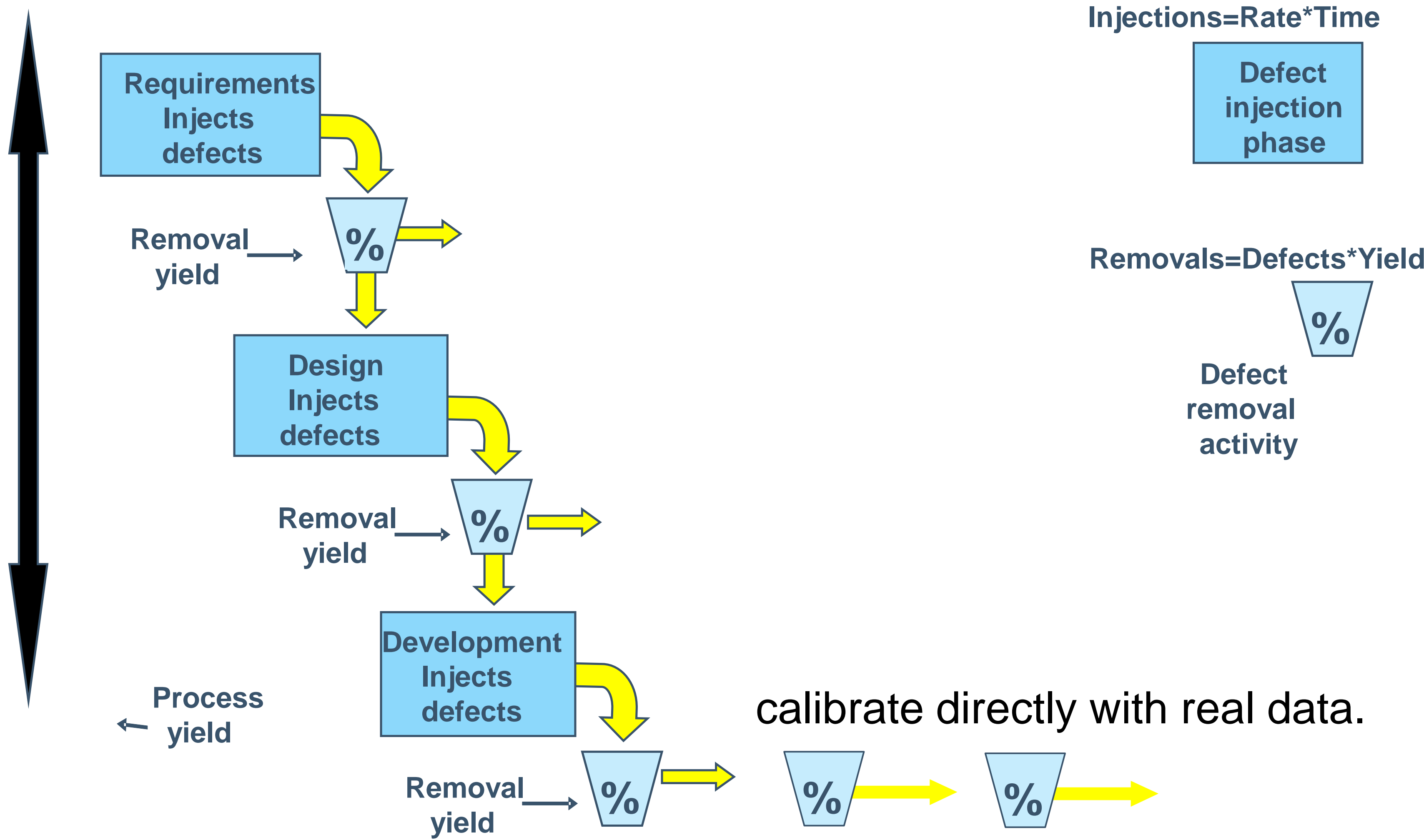
1-5% of defects should be considered to be potential security risks.

(Woody et al, 2014)

How many more? Estimate from defect density.

© 2015 Carnegie Mellon University

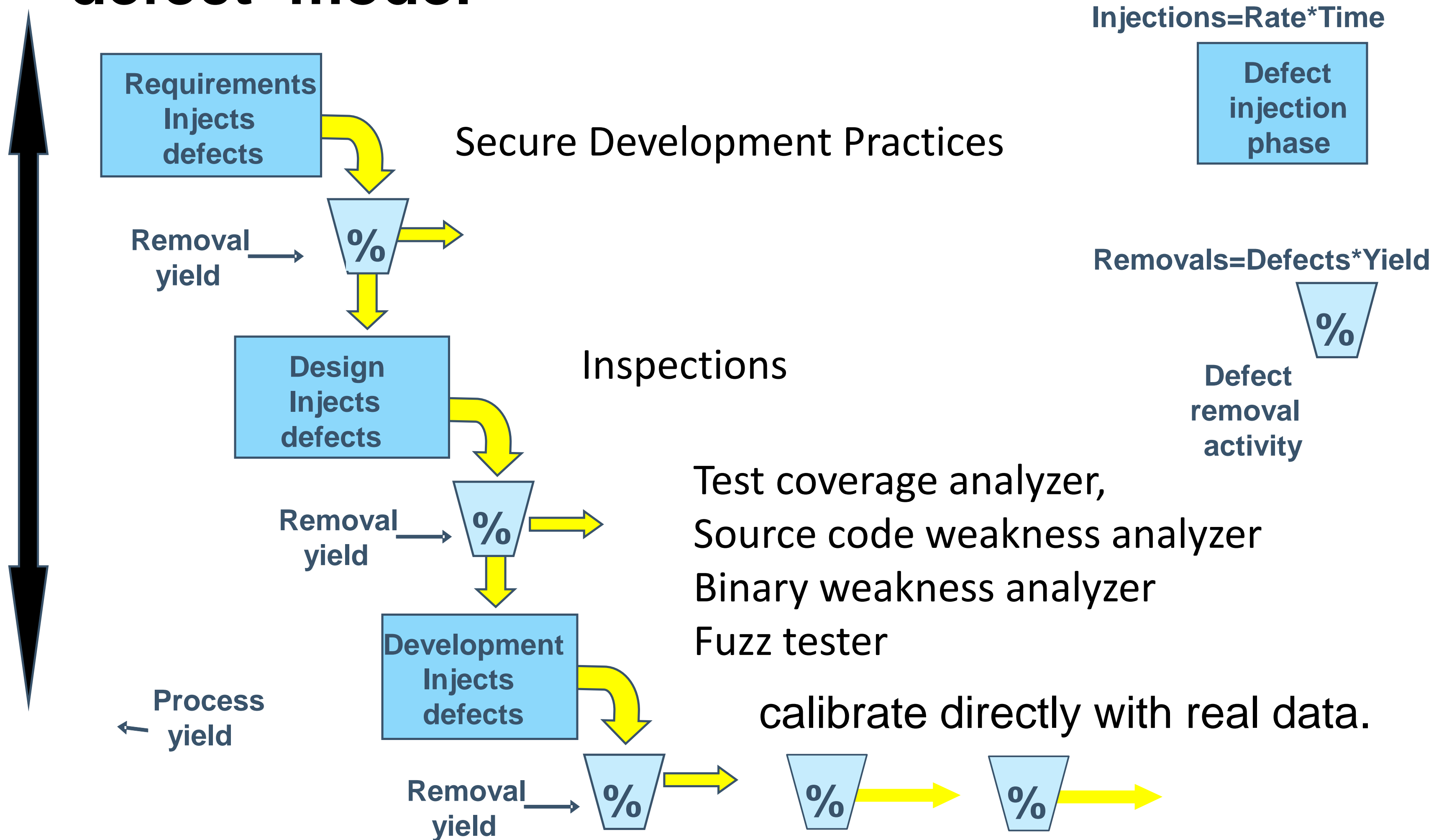
Defect Injection-Filter Model



Similar to C. Jones "Tank and filter." Simplifies assumptions found in Boehm/Chulani COQUALMO.

© 2015 Carnegie Mellon University

Add security specific activities to the defect model



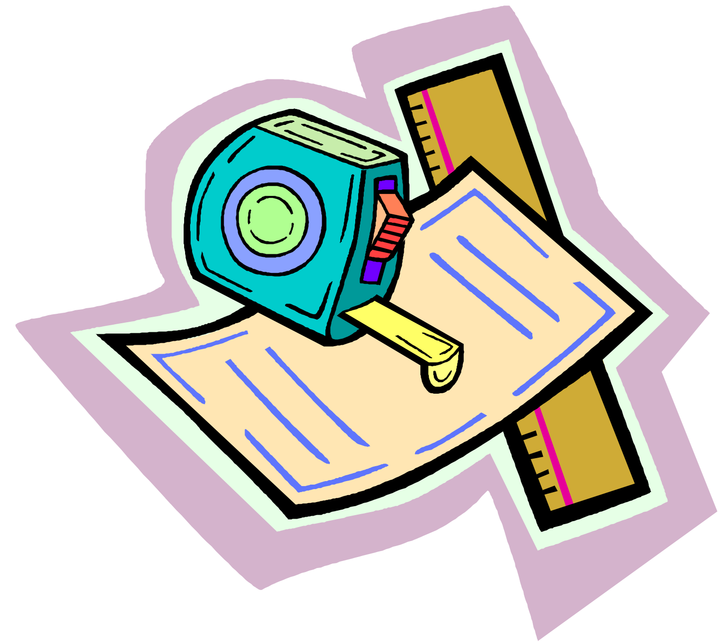
Similar to C. Jones "Tank and filter." Simplifies assumptions found in Boehm/Chulani COQUALMO.

© 2015 Carnegie Mellon University

Quality Process Measures

The TSP uses quality measures for planning and tracking.

1. Defect injection rates [Def/hr/ and removal yields [% removed]
2. Defect density (defects found and present at various stages and size)
3. Review/inspection rates [LOC/hr]



© 2015 Carnegie Mellon University

Metrics

Defect Density (throughout lifetime)

Vulnerability Density (found at each stage)

Phase Injection Rate [defects/hr] (derived)

Phase Effort Distribution (effort-hr)

Phase Removal Yield [% removed] (effectiveness)

Defect “Find and Fix” time [hr/defect] (what was found)

Defect Type (categorize what was wrong)

Defect Injection/Removal Phases

Zero Defect Test time [hr] (cost if no defects present)

Product Size [LOC] [FP] (for normalization and comparisons)

Development Rate (construction phase) [LOC/hr]

Review/Inspection Rate [LOC/hr] (cost of human appraisal)

© 2015 Carnegie Mellon University

Parameters

Phase Injection Rate [defects/hr]

Phase Effort Distribution [%] total time

Size [LOC]

Production Rate (construction phase) [LOC/hr]

Phase Removal Yield [% removed]

Zero Defect Test time [hr]

Phase “Find and Fix” time [hr/defect]

Review/Inspection Rate [LOC/hr]

© 2015 Carnegie Mellon University

Defect Find and Fix Effort

Defect-removal Phase	Time in Minutes
Design Review	5
Design Inspect.	22
Code Review	2
Code Inspect.	25
Unit Test	32
System Test	1405

Y-axis: Time in Minutes (1, 10, 100, 1000, 10000)

X-axis: Defect-removal Phase

© 2015 Carnegie Mellon University

Make the Theoretical Concrete

Do you achieve your goals?

- How much functionality do you want to deliver?
- What are the non-functional targets? (performance, security…)
- What is your desired schedule?
- How many defects do you expect the user to find?

Build the model.

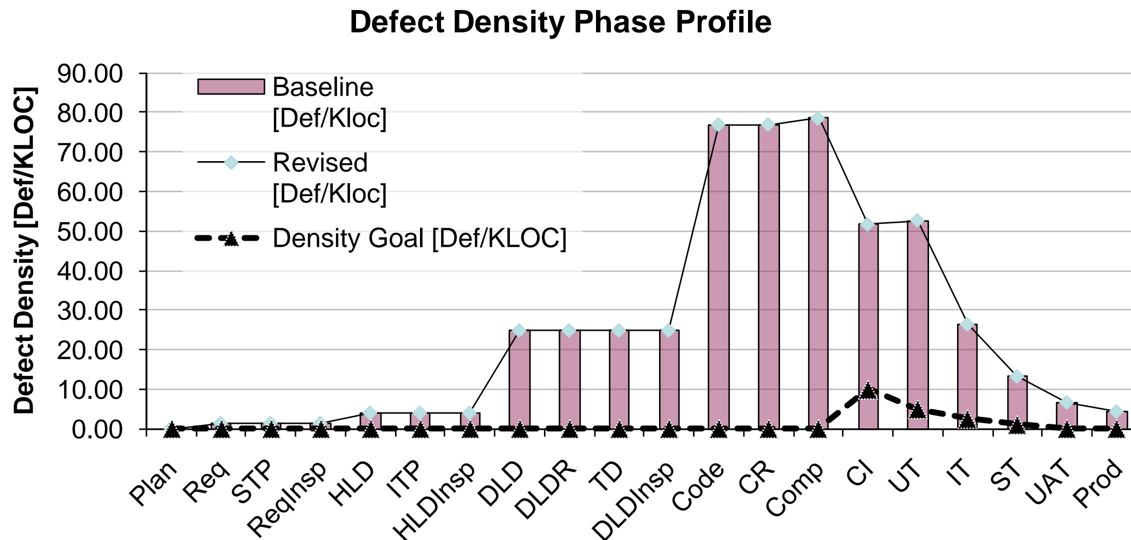
Use real data.

Visualize the result.

© 2015 Carnegie Mellon University

Control Panel

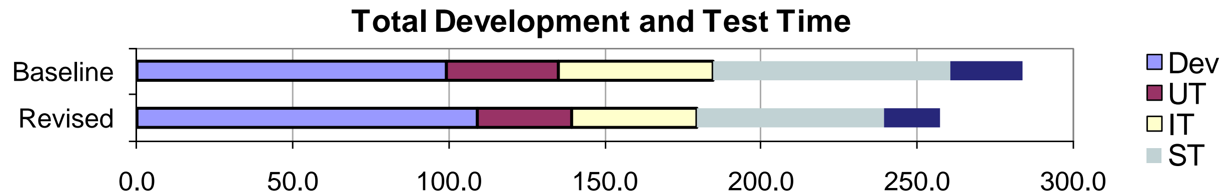
	Rate [LOC/hr]	Yield (per insp)	Yield (total)	# Insp	Effort [hr]
Design Review	200	50.0%	**0.0%**	0	0.0
Design Inspection	200	50.0%	**0.0%**	0	0.0
Code Review	200	50.0%	**0.0%**	0	0.0
Code Inspection	200	50.0%	**0.0%**	0	0.0

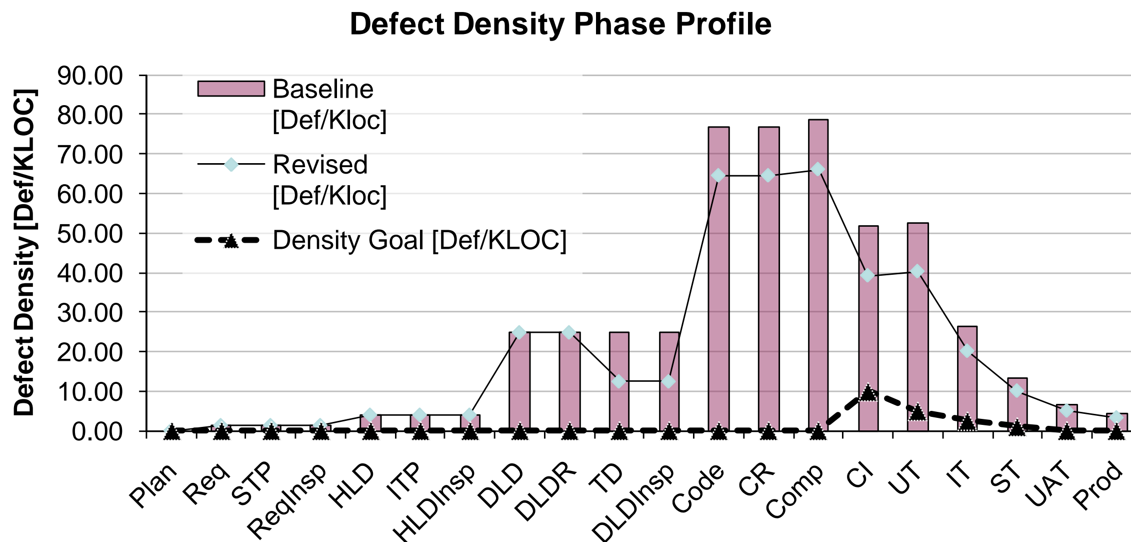


Compare your performance to a baseline.

© 2015 Carnegie Mellon University

Control Panel	Rate [LOC/hr]	Yield (per insp)	Yield (total)	# Insp	Effort [hr]
Design Review	200	50.0%	**0.0%**	1	0.0
Design Inspection	200	50.0%	**0.0%**	0	0.0
Code Review	200	50.0%	**0.0%**	0	0.0
Code Inspection	200	50.0%	**0.0%**	0	0.0

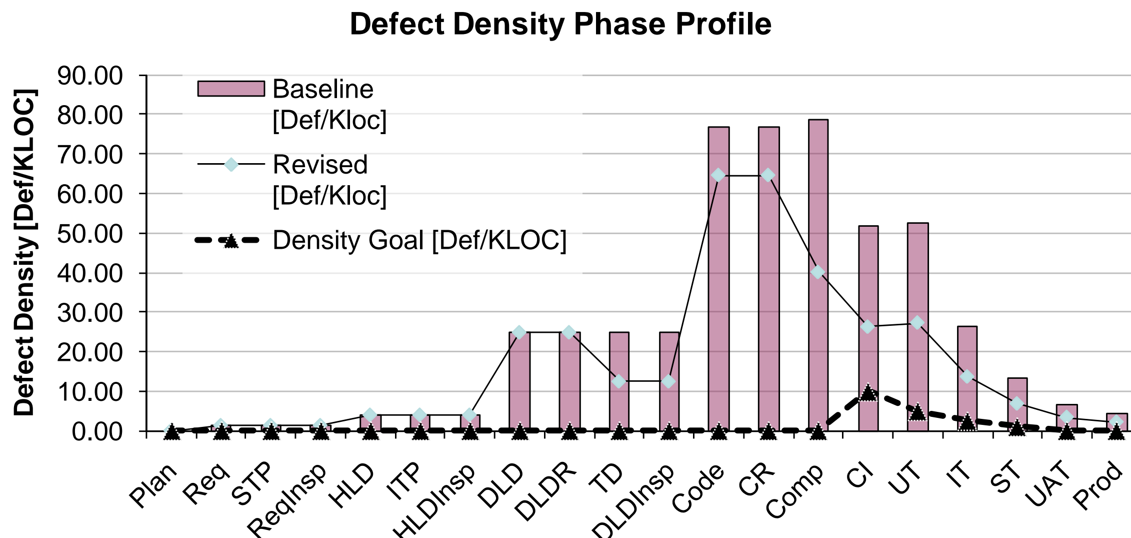




Perform a personal design review.

© 2015 Carnegie Mellon University

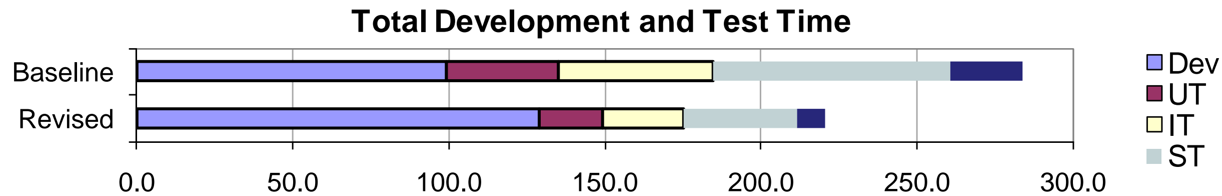
Control Panel	Rate [LOC/hr]	Yield (per insp)	Yield (total)	# Insp	Effort [hr]
Design Review	200	50.0%	**0.0%**	1	0.0
Design Inspection	200	50.0%	**0.0%**	0	0.0
Code Review	200	50.0%	**0.0%**	1	0.0
Code Inspection	200	50.0%	**0.0%**	0	0.0

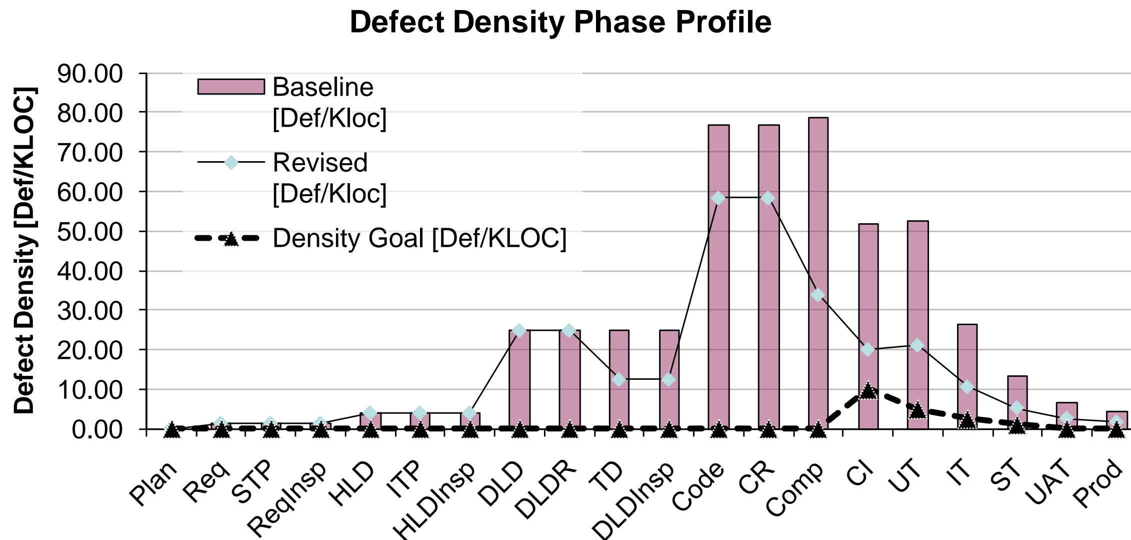


Include a peer design review.

© 2015 Carnegie Mellon University

Control Panel	Rate [LOC/hr]	Yield (per insp)	Yield (total)	# Insp	Effort [hr]
Design Review	200	50.0%	**0.0%**	1	0.0
Design Inspection	200	50.0%	**0.0%**	1	0.0
Code Review	200	50.0%	**0.0%**	1	0.0
Code Inspection	200	50.0%	**0.0%**	0	0.0

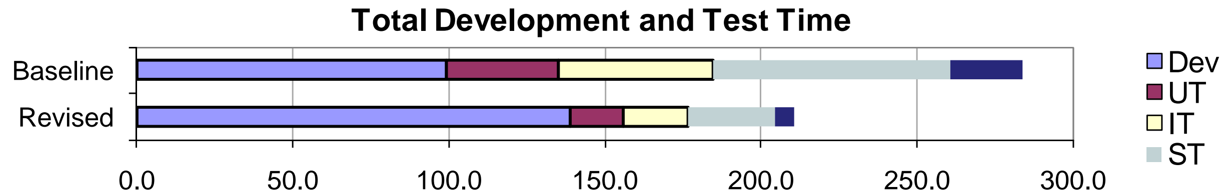


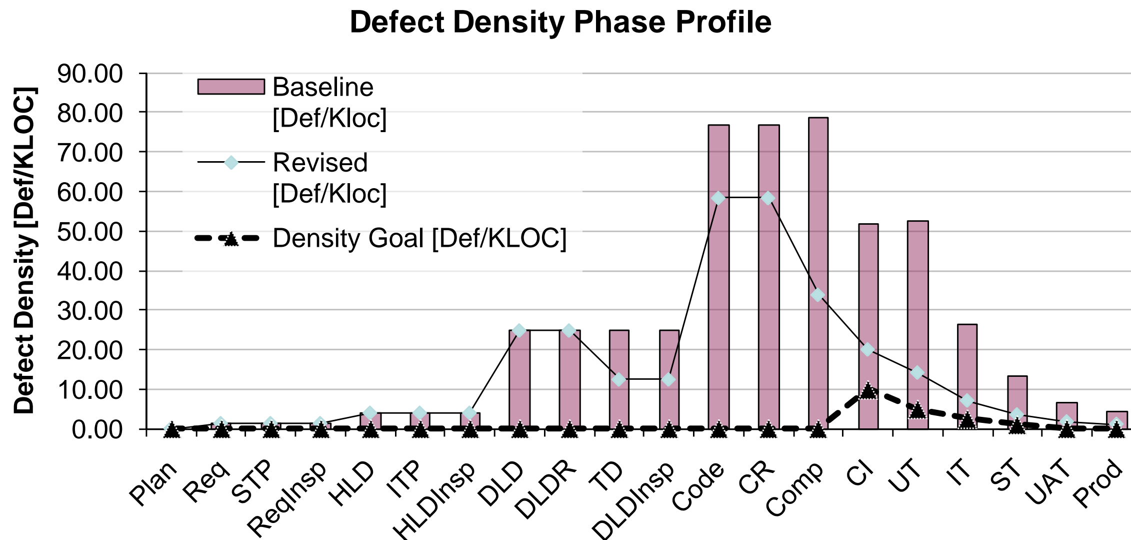


Have a peer inspect the code.

© 2015 Carnegie Mellon University

Control Panel	Rate [LOC/hr]	Yield (per insp)	Yield (total)	# Insp	Effort [hr]
Design Review	200	50.0%	**0.0%**	1	0.0
Design Inspection	200	50.0%	**0.0%**	1	0.0
Code Review	200	50.0%	**0.0%**	1	0.0
Code Inspection	200	50.0%	**0.0%**	1	0.0

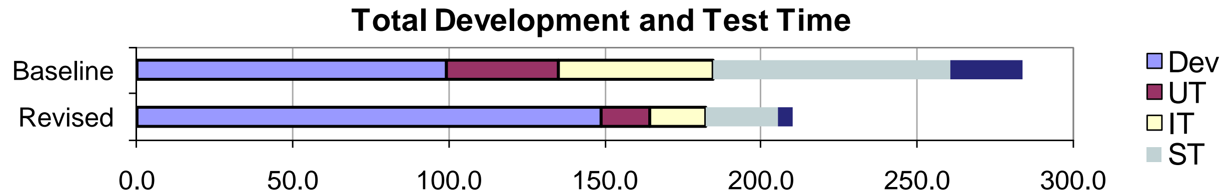


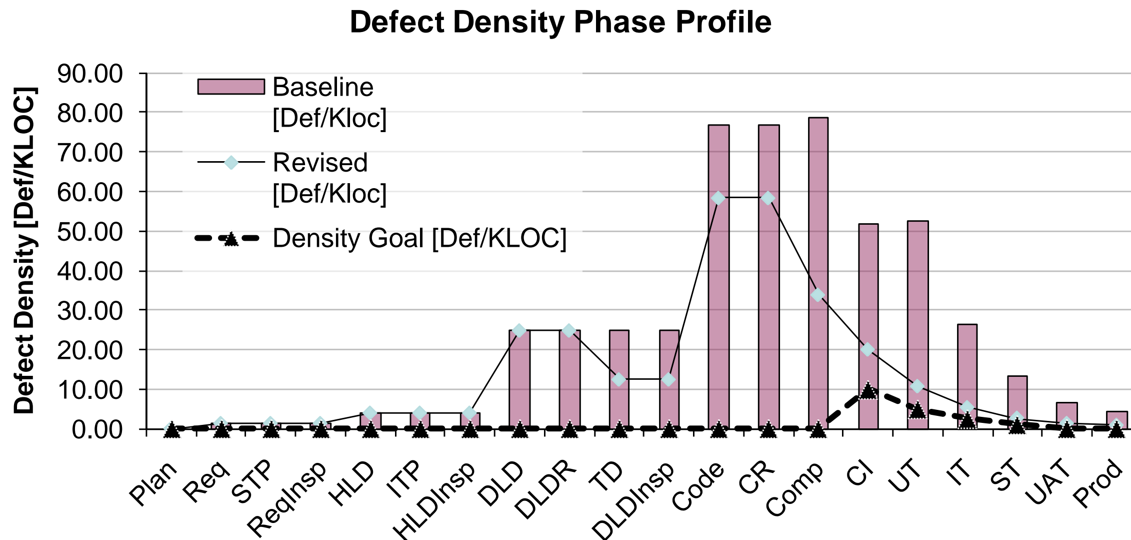


At some point we cross the “quality is free” point.

© 2015 Carnegie Mellon University

Control Panel	Rate [LOC/hr]	Yield (per insp)	Yield (total)	# Insp	Effort [hr]
Design Review	200	50.0%	**0.0%**	1	0.0
Design Inspection	200	50.0%	**0.0%**	1	0.0
Code Review	200	50.0%	**0.0%**	1	0.0
Code Inspection	200	50.0%	**0.0%**	2	0.0

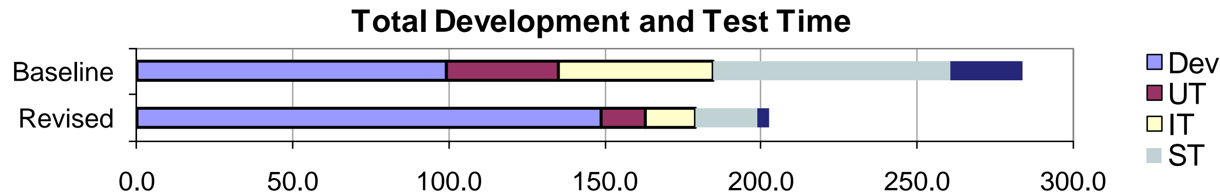


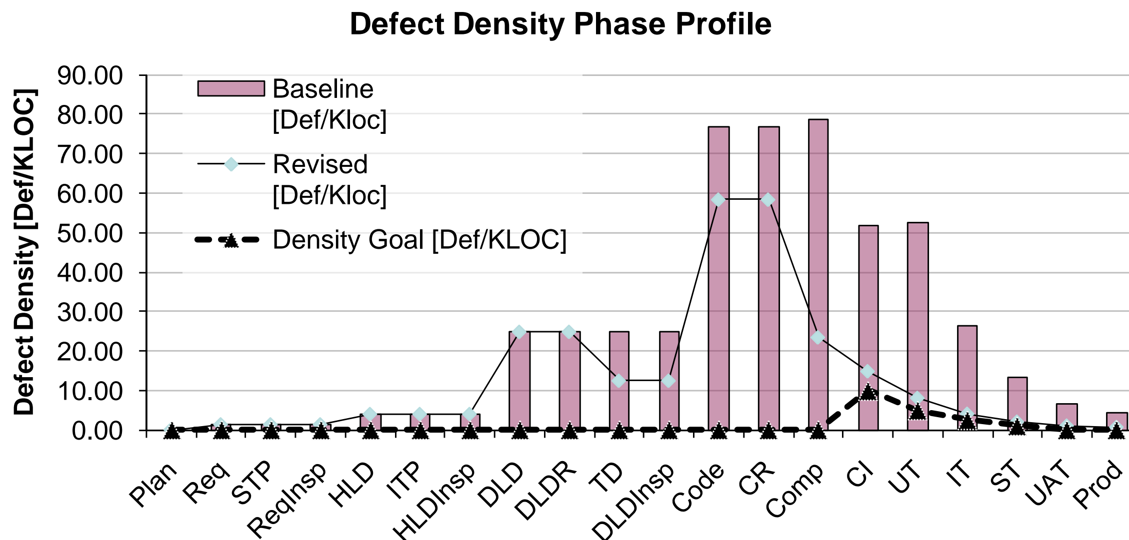


Here’s where you reach the “quality is free” point!

© 2015 Carnegie Mellon University

Control Panel	Rate [LOC/hr]	Yield (per insp)	Yield (total)	# Insp	Effort [hr]
Design Review	200	50.0%	**0.0%**	1	0.0
Design Inspection	200	50.0%	**0.0%**	1	0.0
Code Review	200	70.0%	**0.0%**	1	0.0
Code Inspection	200	70.0%	**0.0%**	2	0.0





The "quality is free" point depends on your personal parameters.

© 2015 Carnegie Mellon University

Questions

If we know that these techniques work, why don’t more people use them?

Who do you know that has issues with developing secure software

How can you help others adopt

© 2015 Carnegie Mellon University