30 Years of Software Assurance: What we have learned, and what we haven’t

Ronda R. Henning
rhenning@harris.com

Assurance -- Defined

- Pledge or promise – a declaration that inspires or is intended to inspire confidence.
- Confidence, in your ability or status
- Certainty, freedom from uncertainty
- Making something certain, overcoming doubt
- Insurance against certainty

-- Microsoft Encarta

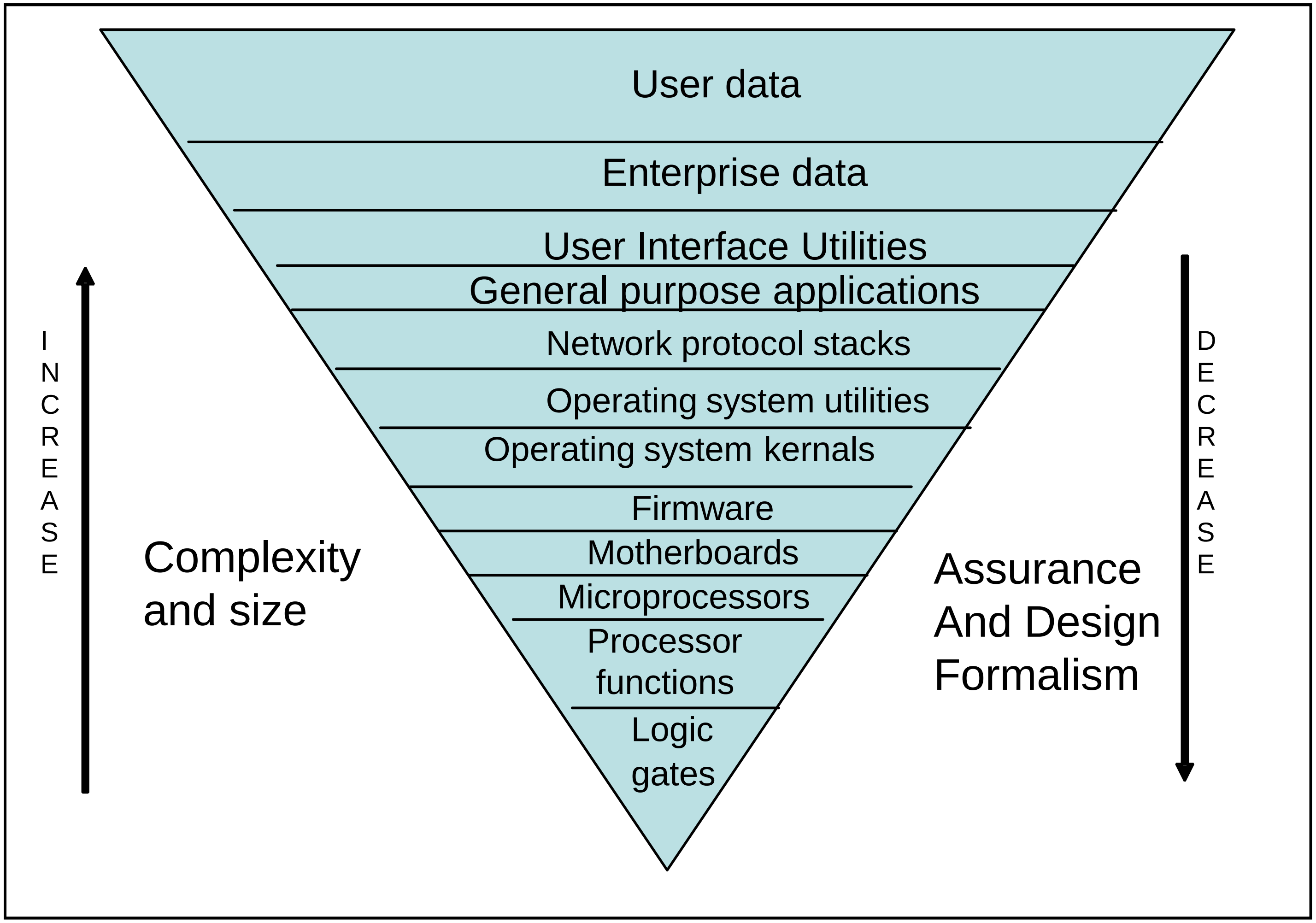
But....

- It’s not a testable definition
  - How to test for intention?
  - How to provide certainty or freedom from doubt?

The timeline of assurance

	1970	1980	1990	2000
Purpose	Specialized Uses	Timesharing/ Early Internet	Computer as Commodity	“Smart devices”
Security Policy	None needed	Userid + Password	MAC and DAC w/labels	RBAC
Mechanisms	Physical Protection	Gold standard = verified code	Pervasive TCSEC “C2 by 1992”	Common Criteria
Philosophy	Common Good	Some were uncommon	Painstaking Evaluation	User Specified strength of countermeasures
Tag Phrase	Woodstock	“Mistakes don’t happen”	Paranoia	Identity Theft

Why Assurance is Hard



User data

Enterprise data

User Interface Utilities

General purpose applications

Network protocol stacks

Operating system utilities

Operating system kernals

Firmware

Motherboards

Microprocessors

Processor functions

Logic gates

INCREASE

Complexity and size

DECREASE

Assurance And Design Formalism

The Reality

- The world changes
  - Requirements for protection change in response to threats
- Not all data is created or protected equally
  - Some is “more sensitive” than others
  - Some is more perishable than others
- When we treat security as static, we become obstacles and not enablers

Assurance Standards

- The Orange Book
  - Linked strength of mechanism with strength of assurance
  - All or nothing concept
- The Common Criteria
  - User defines what functional and assurance objectives are
  - Developer explains how they are met
  - Independent lab verifies the claims

Standards (Continued)

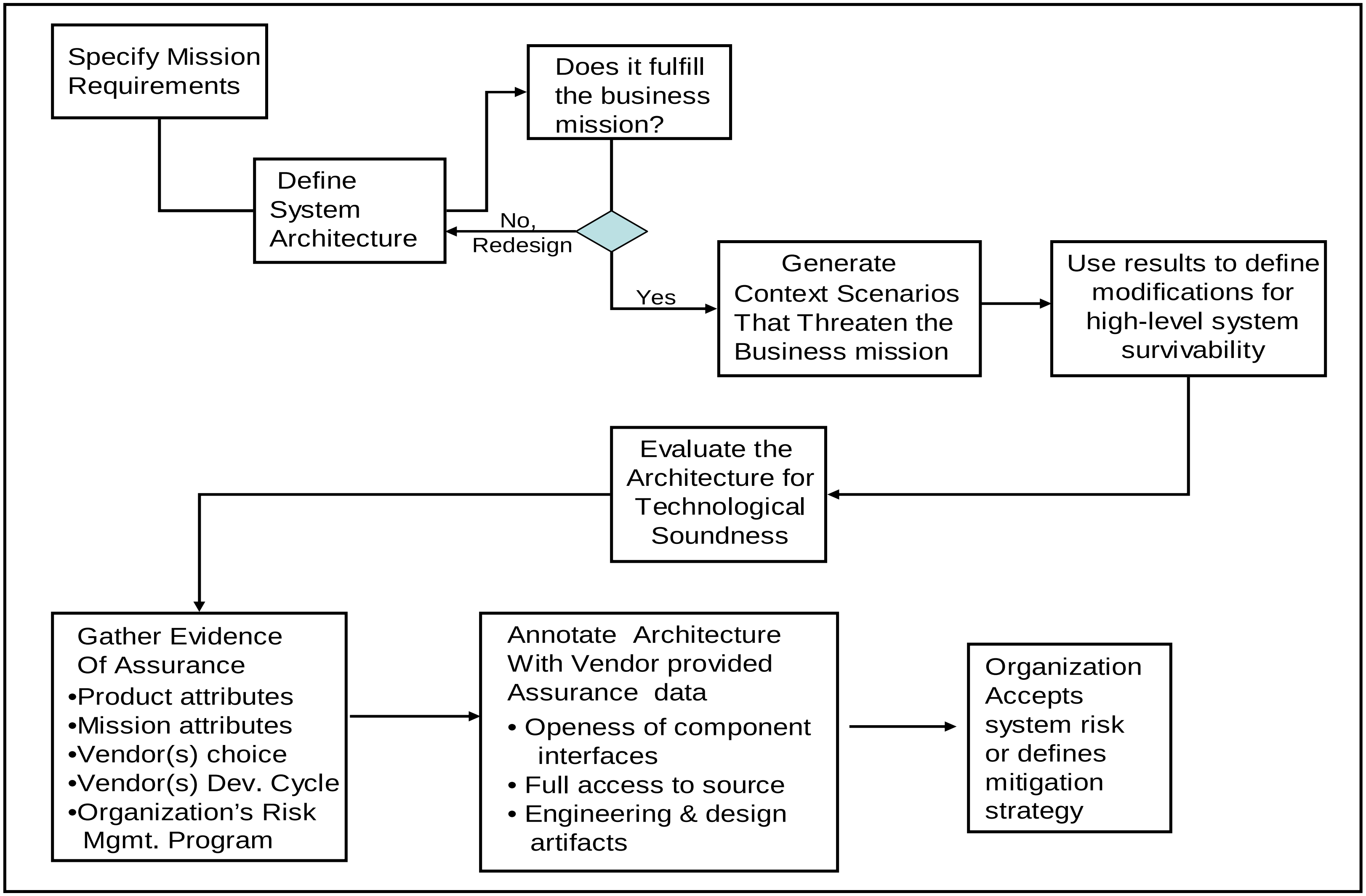
- ISO/IEC 17799
  - Good policies and practices make good neighbors!
- Capability Maturity Model Integrated
  - Process is good, but not specific
- SSE-CMM
  - Process is not only good, but security and assurance bring additional processes to the framework.

How to get there from here

- Failure – deviations from specified behavior
- Fault – failure that doesn't necessarily impact the whole system
- Error– Impacts the operation of the system as a whole, and implies defects prevent correct operation

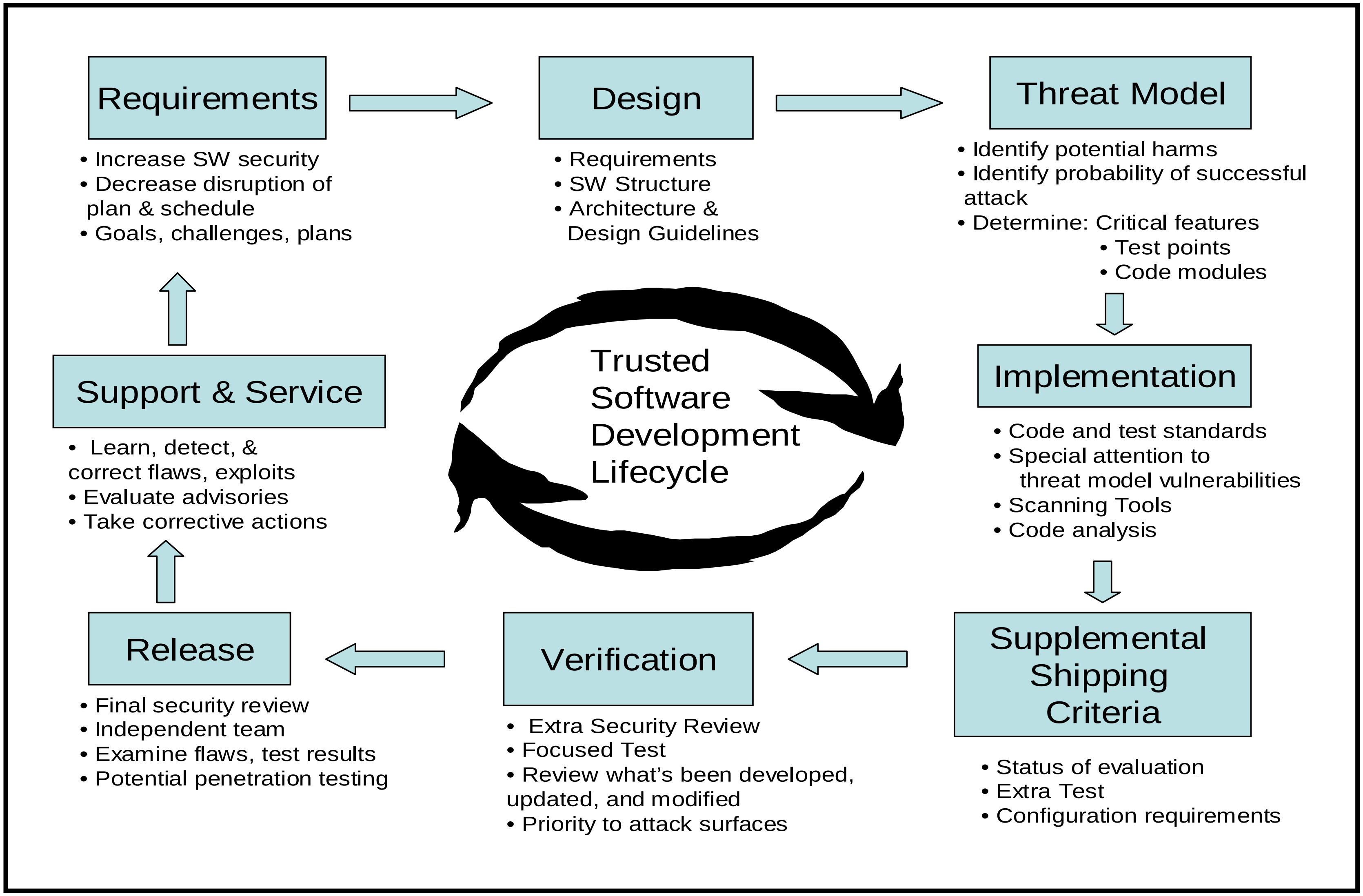
– Dobson and Randell

COTS Risk Assessment Methodology


Specify Mission Requirements
Define System Architecture
Does it fulfill the business mission?
No, Redesign
Yes
Generate Context Scenarios That Threaten the Business mission
Use results to define modifications for high-level system survivability
Evaluate the Architecture for Technological Soundness
Gather Evidence Of Assurance
•Product attributes
•Mission attributes
•Vendor(s) choice
•Vendor(s) Dev. Cycle
•Organization's Risk Mgmt. Program
Annotate Architecture With Vendor provided Assurance data
• Openess of component interfaces
• Full access to source
• Engineering & design artifacts
Organization Accepts system risk or defines mitigation strategy

Source: Software Engineering Institute (V-Rate Methodology)

The Microsoft Way



Requirements
- Increase SW security
- Decrease disruption of plan & schedule
- Goals, challenges, plans

Design
- Requirements
- SW Structure
- Architecture & Design Guidelines

Threat Model
- Identify potential harms
- Identify probability of successful attack
- Determine: Critical features
  - Test points
  - Code modules

Implementation
- Code and test standards
- Special attention to threat model vulnerabilities
- Scanning Tools
- Code analysis

Supplemental Shipping Criteria
- Status of evaluation
- Extra Test
- Configuration requirements

Verification
- Extra Security Review
- Focused Test
- Review what's been developed, updated, and modified
- Priority to attack surfaces

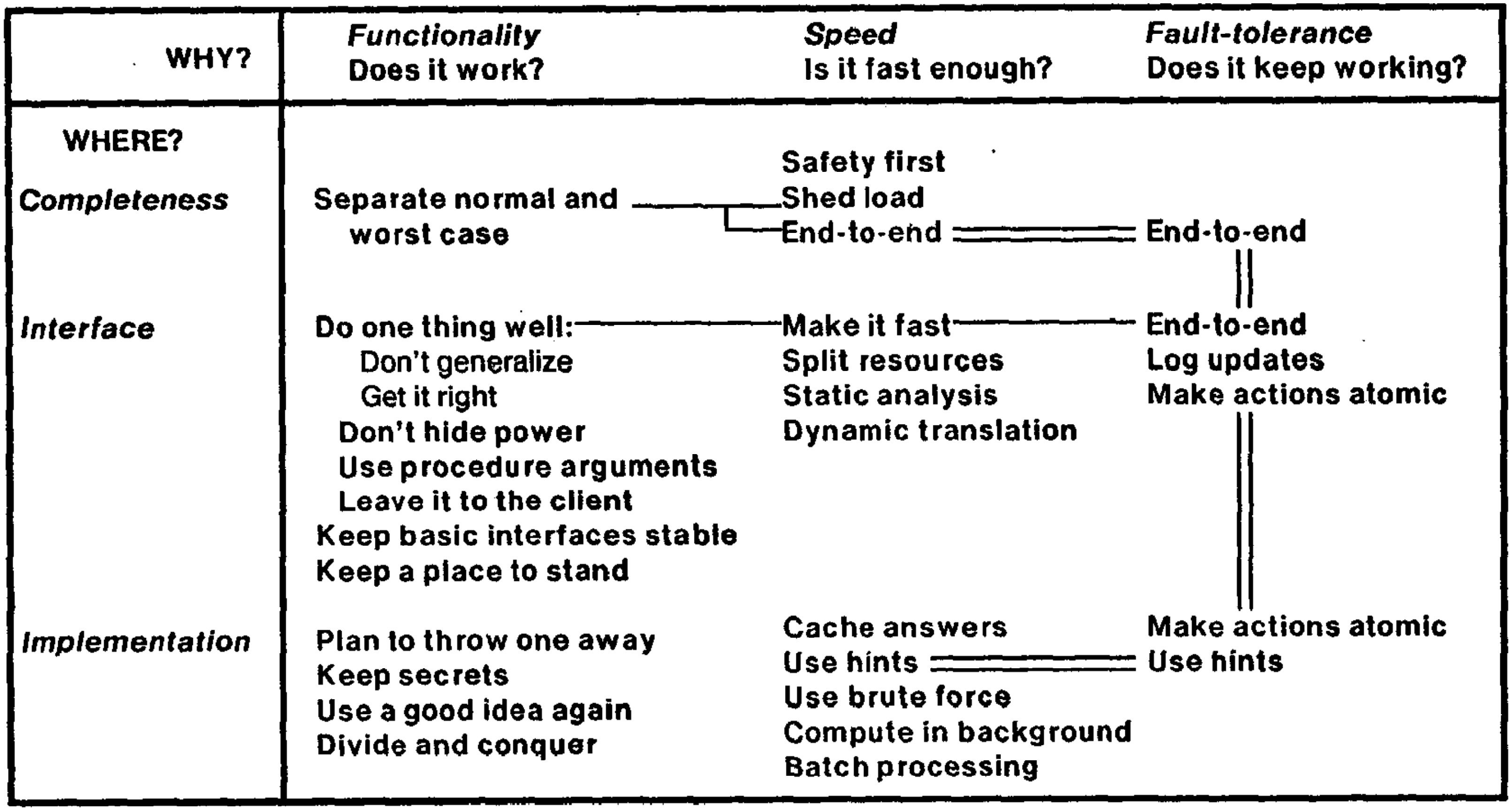
Release
- Final security review
- Independent team
- Examine flaws, test results
- Potential penetration testing

Support & Service
- Learn, detect, & correct flaws, exploits
- Evaluate advisories
- Take corrective actions

Trusted Software Development Lifecycle

Source: Lipner

Hints for Computer System Design



WHY?	*Functionality* Does it work?	*Speed* Is it fast enough?	*Fault-tolerance* Does it keep working?
WHERE?			
Completeness	Separate normal and worst case	Safety first Shed load End-to-end	End-to-end
Interface	Do one thing well: Don't generalize Get it right Don't hide power Use procedure arguments Leave it to the client Keep basic interfaces stable Keep a place to stand	Make it fast Split resources Static analysis Dynamic translation	End-to-end Log updates Make actions atomic
Implementation	Plan to throw one away Keep secrets Use a good idea again Divide and conquer	Cache answers Use hints Use brute force Compute in background Batch processing	Make actions atomic Use hints

Figure 1: Summary of the slogans

Everything we ever needed, we learned in the early 1970's -- Lampson

Conclusions

- What have we learned:
  - Countermeasures are better
  - Defense in Depth helps
  - Process Improvement Initiatives institutionalize improvement

What we haven't learned

- Discipline
  - Computer science and system design is still an art
  - Engineers that understand integration and allocation of assurance are hard to find
  - We substitute testing for early error detection – and pay the penalty.

In Summary

- Those who do not learn from the mistakes of the past are doomed to repeat them.

- Forums such as this capture our attempts to learn about assurance, and to learn how to implement it more effectively.