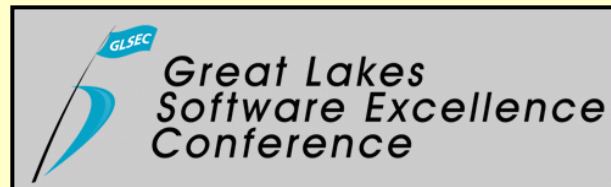


# All Software Is Defective

## *Implications for the Software Industry*



### **Software Quality Consulting Inc.**

---

**Steven R. Rakitin**  
President

- Consulting
- Training
- Auditing

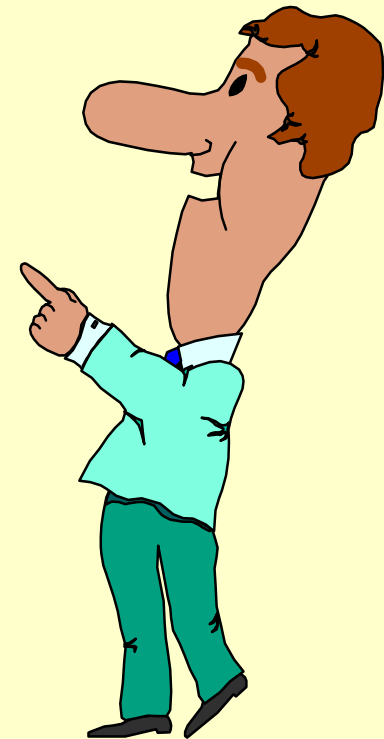
**Phone:** 508.529.4282  
**Fax:** 508.529.7799

[www.swqual.com](http://www.swqual.com)  
[info@swqual.com](mailto:info@swqual.com)

# Overview

---

- **Why All Software is Defective**
- **Implications**
  - For Managers...
  - For Developers...
  - For Testers...
- **Summary**



# Why All Software is Defective

---

- Because **we can't prove that software is defect-free...**
- Because **developers inject on average one defect for every ~8 lines of code...**
- Because **software development is an inherently human process...**



# Because we can't prove that software is defect-free...

---

**“Testing can be used to show the presence of bugs but never their absence.”**



**Prof. Edsger Dijkstra**

# Because we can't prove that software is defect-free...

---

- A trivial program stores names, addresses, phone numbers
  - Names and addresses are 20 alpha characters
  - Phone numbers have 10 digits
  - Names  $26^{20}$  (20 characters, each with 26 possible choices)
  - Addresses  $26^{20}$  (20 characters, each with 26 possible choices)
  - Phone  $10^{10}$  (10 digits, each with 10 possible choices)
- Total combinations =  $26^{20} * 26^{20} * 10^{10} = \sim 10^{66}$
- At **one trillion test cases per second**, it would take about **50,000 years to run all tests!!**

McConnell, S., Code Complete, Microsoft Press, 1993, p. 593

# Because developers inject defects...

---



Dr. Harlan Mills

**“Programs do not acquire bugs as people acquire germs, by hanging around other buggy programs.**

**Programmers must insert them.”**



# Because developers inject defects...

---

- Software is released with many **known defects** and some number of **unknown defects**
  - Watts Humphrey reported Defect Injection Rates for a sample of **810 experienced software engineers**:

Group	Average number of defects injected per KLOC
All	120.8
Upper Quartile	61.9
Upper 10%	28.9
Upper 1%	11.2

Humphrey, W., "The Quality Attitude", *news@sei newsletter*, Number 3, 2004.

---

# Because developers inject defects...

---

- Please try this at work:

**Est. unknown defects = defects injected – defects found**

where: defects injected = size (KLOC) X 120



# Because developers inject defects...

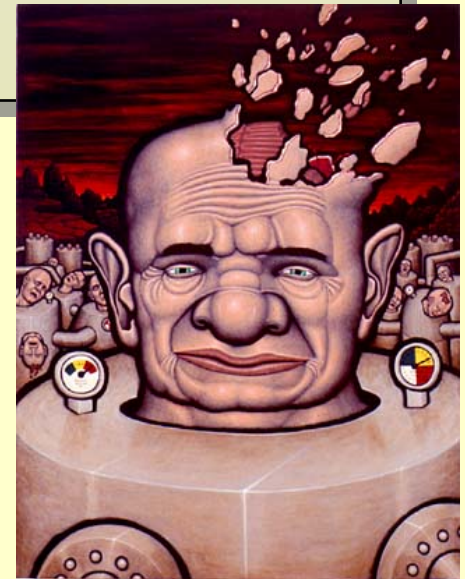
---

- A simple example...
- 1 million LOC = 1,000 KLOCs
  - Avg. defect injection rate of 120 defects/KLOC
  - **120,000** defects injected
  - If we find 95% = **114,000** defects found
- **Unknown defects** = defects injected – defects found
  - = (120,000 – 114,000)
  - = **6,000**

Because software development is an inherently human process...

---

- Complexity is increasing beyond our ability to “keep it all in our head”
- Many applications lack **“conceptual integrity”**...



Because software development is an inherently human process...

---

**“When coupled with the explosive growth of the Internet and the resulting exposure to hackers, criminals, and terrorists, the need for reliable, dependable, and secure software systems will steadily increase.**

**If experience is any guide, as these systems are used to perform more critical functions, they will get more complex and less reliable.”**



**Watts Humphrey**

---

Humphrey, W., “Defective Software Works”, news@sei newsletter, Number 1, 2004

Because software development is an inherently human process...

---

“I will contend that **conceptual integrity is *the most important consideration in system design.***

It is better to have a system omit anomalous features and improvements, but to **reflect one set of design ideas**, than to have one that contains many good but independent and uncoordinated ideas.”



Fred Brooks

Brooks, F., *The Mythical Man-Month*, Addison-Wesley, 1975

# All Software Is Defective

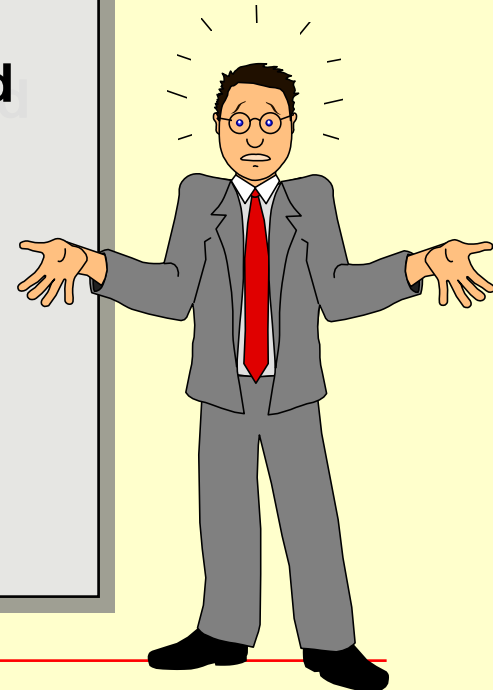
---

- **Implications for Managers...**
- **Implications for Developers...**
- **Implications for Testers...**

# Implications for Managers...

---

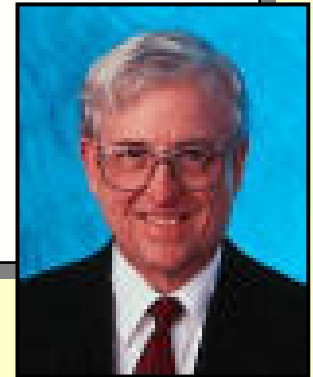
- Requirements written in **English** are inherently **vague** and **ambiguous...**
- We don't **train people** in how to write good requirements
- Accurate schedules can **only** be developed from **complete, concise, unambiguous requirements**
- **Misconception** - spending time writing requirements delays product release...



# Implications for Managers...

---

**“A significant and important aspect of requirements errors is that if they are not prevented or removed, they tend to flow downstream into design, code, and user manuals. Historically, errors which originate in requirements tend to be the most expensive and troublesome to eliminate later.”**



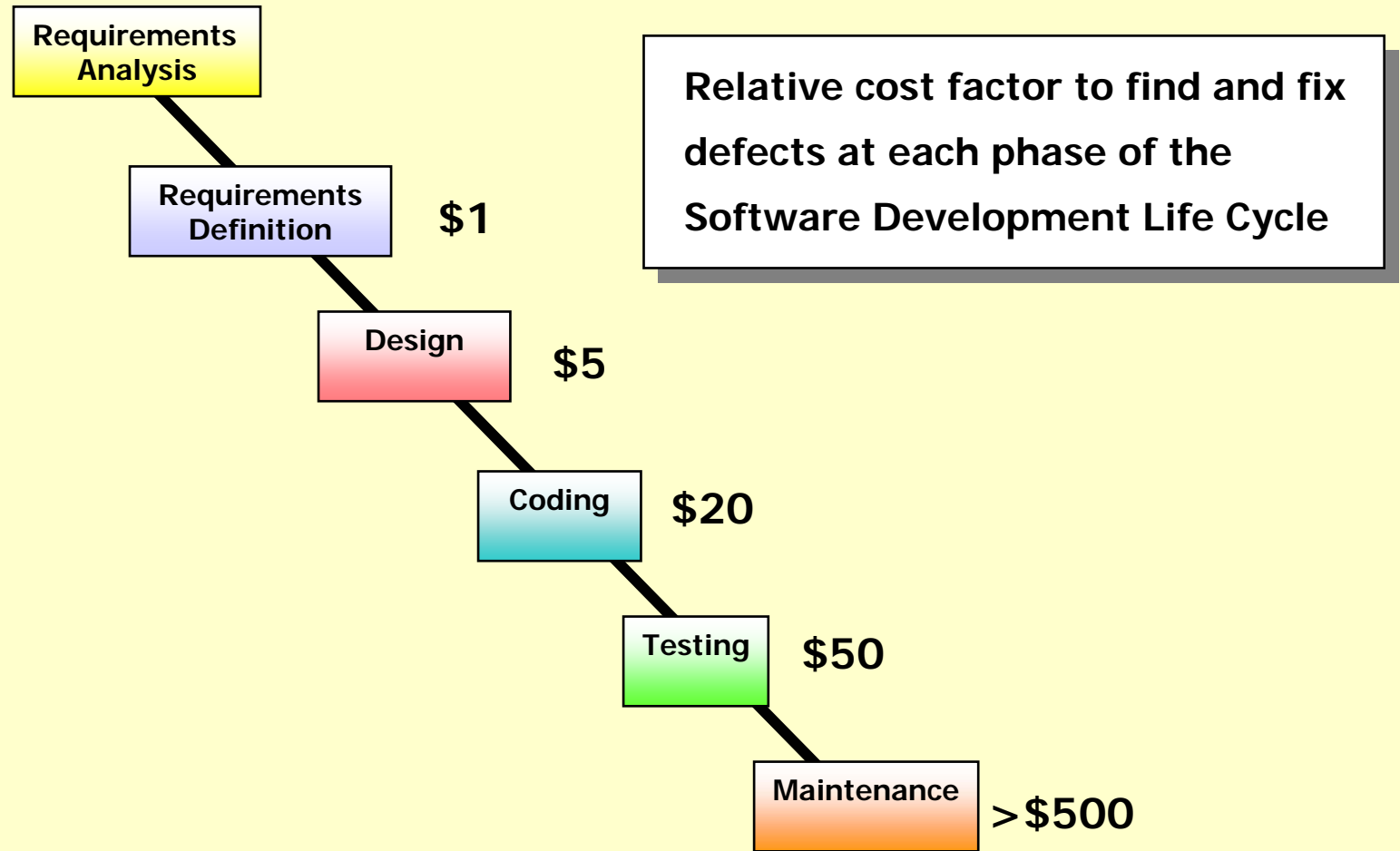
**Capers Jones**

---

Jones, C., Software Quality: Analysis and Guidelines for Success, International Thomson Computer Press, 1997.

# Implications for Managers...

---



Boehm, B., Software Engineering Economics, Prentice-Hall, 1981

---

# Implications for Managers...

---

- **Take away message for Managers:**
  - **Getting it right the first time is most efficient way to work**
  - **Provide appropriate training so teams can do quality work**
  - **If you want teams to do quality work, hold them accountable and reward them for meeting quality goals**
  - **If managers don't take responsibility for managing quality, no one else will**

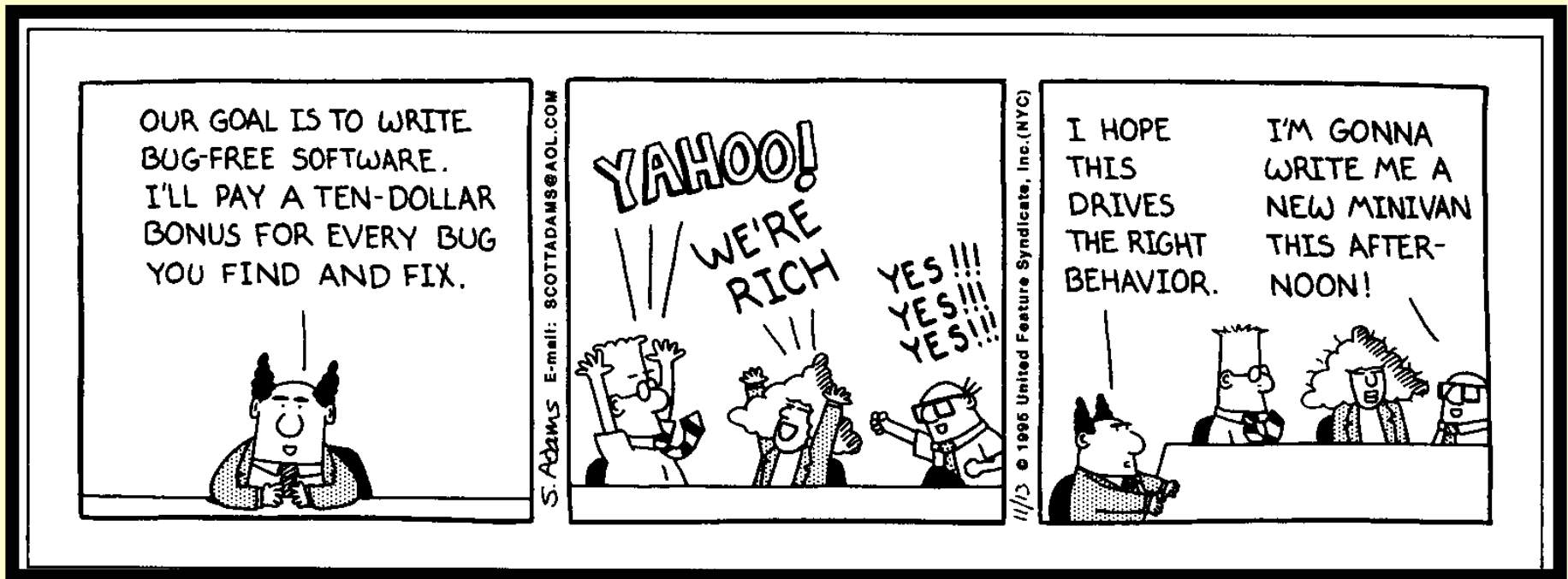
---

Humphrey, W., *Winning with Software – An Executive Strategy*, Addison-Wesley, 2002

# Implications for Managers...

---

- What **NOT** to do...



# Implications for Developers...

---

- “The current common view that it is impossible to produce defect free software is an excuse for not making the effort to do **quality work.**”
- To do **quality work**, developers need to:
  - Insist on starting with **clearly written requirements**
  - **Perform peer reviews** of requirements, designs, code
  - **Apply conceptual integrity** to system design
  - **Take pride in your work!**

---

Humphrey, W., “The Quality Attitude”, *news@sei newsletter*, Number 3, 2004.

# Implications for Developers...

---

- Insist on **clearly written requirements**

**“The following process allows the user to set the recommended plan parameters from the New Plan worksheet screen (via a new Recommended Plan screen) and then when the user opens the worksheet (see Step 3 below) with the existing Open Worksheet function the system will automatically call a new Recommended Plan Process and wait for it to finish and then display the worksheet to the user.”**

**1 sentence – 66 words!**



# Implications for Developers...

---

- Insist on **clearly written requirements**

“When running the calculation rule, if a numerical operation is to be performed such as multiplication or division, the system shall only run the rule if the value to be used is in numerical form.”

1 sentence - 32 words

**IF** numerical operations performed when running calculation rules

**THEN** run calculation rule ONLY IF value to be used is numeric

**ELSE ??**

**ENDIF**

# Implications for Developers...

---

Practice	Airlie Council (1)	SEI CMM (2) (Level)	Best in Class Companies (3)	Best Practices Clearinghouse (4)
Risk Management	✓			✓
Requirements Definition	✓	✓ (2)	✓	✓
<b>Peer Reviews and Inspections</b>	<b>✓</b>	<b>✓ (3)</b>	<b>✓</b>	<b>✓</b>
Binary Gates at Inch-pebble Level	✓	✓ (2)		
Software Quality Assurance		✓ (2)	✓	✓
Defect Tracking vs. Quality Targets	✓	✓ (4)	✓	
Configuration Management	✓	✓ (2)		
People-aware Management	✓			

(1) Yourdon, E., Rise & Resurrection of the American Programmer, Prentice-Hall, 1998

(2) Paulk, M. C., et. al., The Capability Maturity Model: Guidelines for Improving the Software Process, Addison-Wesley, 1995

(3) Jones, C., Software Quality: Analysis and Guidelines for Success, International Thomson Computer Press, 1997.

(4) Best Practices Clearinghouse at <http://bpch.dau.mil/provider/mainpage.asp>

# Implications for Developers...

---

- Design systems with **conceptual integrity**

“...those software systems that have excited passionate fans are those that are the products of **one or a few designing minds** [,...].

Conceptual integrity in turn dictates that the design must proceed from **one mind**, or from **a very small number of agreeing resonant minds.**”



Vesalius 1542

Brooks, F., [The Mythical Man-Month](#), Addison-Wesley, 1975

# Implications for Developers...

---

What can happen when conceptual integrity is ignored...



# Implications for Developers...

---

- **Take Away Message for Developers...**

- **Poorly written requirements is root cause of many defects**
- **Insist on clearly written requirements**
  - **Is every requirement implementable?**
  - **Avoid inherently vague words...**
    - could, would, should, might, may, some, sometimes, often, usually, always, most, mostly, all, none, etc.
- **Peer review Requirements, Designs and selected code**
- **Apply conceptual integrity to system design**
- **Maintain good working relationships with testers...**
  - **Take a tester to lunch...**

# Implications for Testers...

---

- Of all defects, those reported by **customers** are most important...
- To find these defects, testers need **domain knowledge, story telling skills**, as well as good testing skills...
- Overall test strategy should include **combination of testing approaches...**
- **Design for Testability** means writing requirements in a way that **improves testability...**

# Implications for Testers...

---

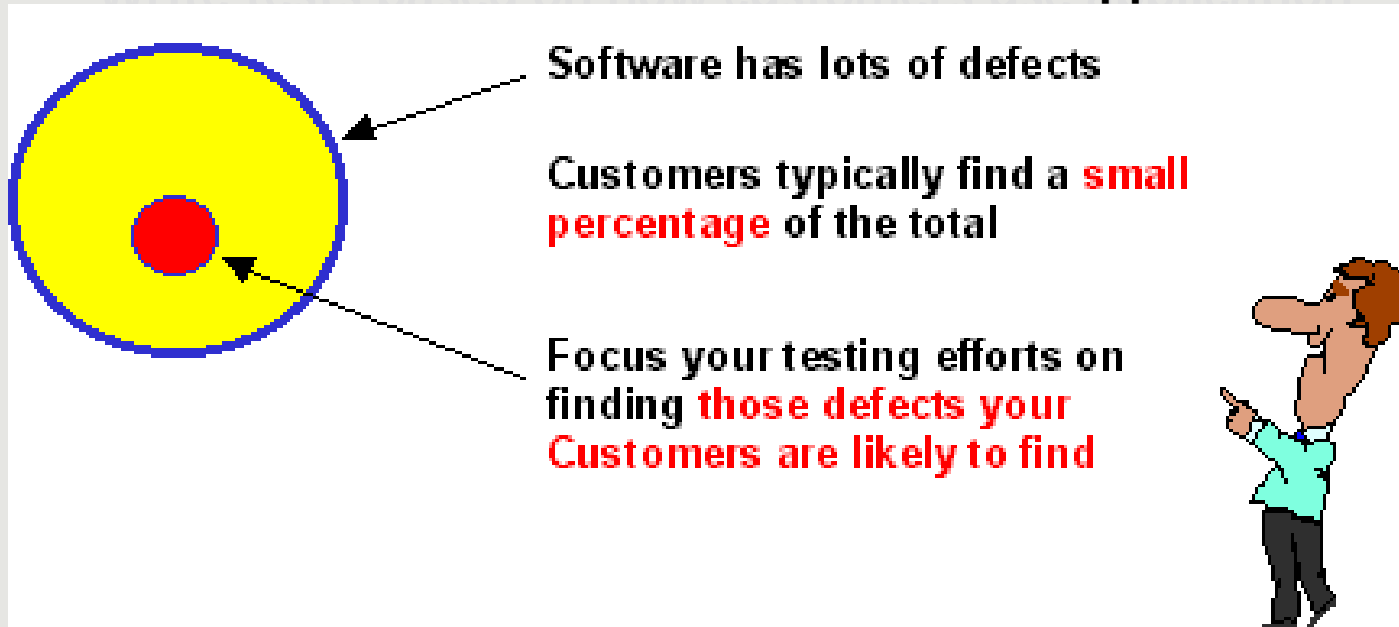
- **Overall test strategy** should include combination of:
  - **Scenario-based testing**
    - Act Like a Customer Testing™
    - Storytelling
  - **Requirements-based testing**
    - Using variety of **test types** such as positive, negative, boundary, stress, startup, installation, configuration, API, GUI, etc...
  - **Exploratory testing**
    - Using testers who are good at thinking outside the box
  - **Automated tests**
    - In areas that yield highest ROI...

---

Act Like a Customer Testing is a trademark of Software Quality Consulting, Inc.

# Implications for Testers...

- Act Like a Customer Testing™
  - Testers need **domain knowledge** to be effective
  - Write tests based on how customers use application



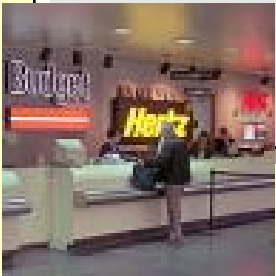
Act Like a Customer Testing is a trademark of Software Quality Consulting, Inc.

# Implications for Testers...

---

- **A storytelling example:**

A customer rents a car for a three-day business trip. Midway through the rental, he extends the rental for another week. This, by the way, gives him enough rental points to reach Preferred status. Several days later, he calls to report that the car has been stolen. He insists that the Preferred benefit of on-site replacement applies, even though he was not a Preferred customer at the start of the rental. The company agrees and a new car is delivered to him. Two days after that, he calls to report that the “stolen” car has been found. It turns out he’d forgotten where he’d parked it. He wants one of the cars picked up and the appropriate transaction closed. Oh, and one other thing, the way he discovered the missing car was by backing into it with its replacement, so they’re both damaged.

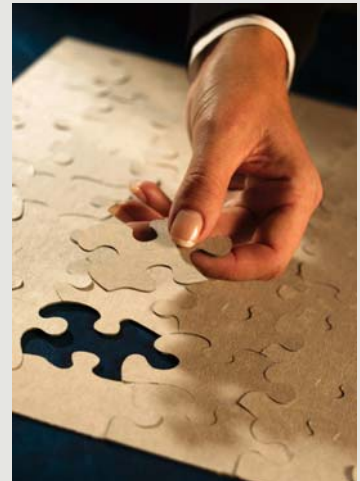


Buwalda, H., “Soap Opera Testing”, *Better Software*, February 2004.

# Implications for Testers...

---

- **Requirements-based testing**
  - Testing against requirements is always appropriate but never sufficient by itself
- **Exploratory Testing**
  - Bach describes ET as akin to doing a jig saw puzzle...
  - Techniques and approaches used to do a jig saw puzzle often change as puzzle begins to take shape
  - “ The puzzle changes the puzzling.”



# Design for Testability

---

- **Example:**

User enters a new password. Application confirms if new password (NP) meets the following rules.

1. If old password (OP) is correct, NP and Confirm NP are same, pass configuration edit, and has not been used during prior two changes, confirm change with a successfully changed password message. An "OK" button brings user to Home page. Error Message = Password successfully changed.
2. If OP is correct and NP and Confirm NP match but do not conform to configuration settings, a message describing error is displayed. OP, NP and Confirm NP are blank after user selects "OK" on error message. "OK" button brings user to Change Password screen. Error Message = Password you entered does not conform to format specified by your system administrator. Enter a valid password.
3. If OP is valid and NP and Confirm NP do not match, a message describing error is displayed. OP, NP and Confirm NP are blank after user selects "OK" on error message. "OK" button brings user to Change Password screen. Error Message = NP and Confirm NP entries do not match. Try again.
4. If OP is correct and NP and Confirm NP match and pass configuration but has been used prior by user during previous two changes a message is displayed. OP, NP and Confirm NP are blank after user selects "OK" on error message. "Ok" button brings user to Change Password screen. Error Message – Password has been used too recently. Try again.
5. If NP and Confirm NP match and pass configuration but OP is invalid, a message is displayed. OP, NP and Confirm NP are blank after user selects "OK" on error message. "Ok" button brings user to Change Password screen. Error Message – OP entered is invalid. Try again.

# Design for Testability

---

OP Confirmed	NP and Confirm NP match	Password Rules followed	NP not used in last two changes	Password change successful?	Display Message
TRUE	TRUE	TRUE	TRUE	Yes	1
TRUE	TRUE	FALSE	DON'T CARE	No	2
TRUE	FALSE	DON'T CARE	DON'T CARE	No	3
TRUE	TRUE	TRUE	FALSE	No	4
FALSE	DON'T CARE	DON'T CARE	DON'T CARE	No	5

## Messages:

1. "Password successfully changed"
2. "The password entered does not conform to the format specified by your sys admin. Enter a valid password."
3. "New Password and Confirm New Password entries do not match. Try again."
4. "The password you entered has been used recently. Try again."
5. "The Old Password you entered is invalid. Try again."

**How many tests are required to test these requirements?**

# Implications for Testers...

---

- **Test Automation Challenges...**

- **Two-thirds** of all test automation efforts **fail...**
- **Where will test automation be most effective?**

<b>Good Candidates</b>	<b>Poor Candidates</b>
Mature, stable applications	New, changing applications
Short simple transactions	Long, complex transactions
Many data combinations	One-offs
Results easy to generate	Results difficult to predict
Tests executed regularly	Tests crossing multiple applications
Tasks difficult to do manually	Tasks difficult to do manually

Brian LeSeur, Star Quality, Inc.

---

# Implications for Testers...

---

- **Test Automation Challenges...**
  - Applications need to be **designed for automation**
    - Naming conventions
    - Use of only standard widgets
    - Addition of “hidden” controls
    - Unique page names
  - Always makes sense to use automation for **smoke tests**
    - Emphasize breadth rather than depth
    - Run smoke tests against each build
    - Use results to assess quality of build

Brian LeSeur, Star Quality, Inc.

---

# Implications for Testers...

---

- **Take Away Message for Testers...**
  - **Actively participate in peer review of requirements**
    - Is every requirement testable?
    - Identify ambiguous requirements...
  - **Acquire domain knowledge and storytelling skills**
    - Visit customers if you can, else spend time with tech support
  - **Increase effectiveness of testing by including:**
    - Scenario-based tests
    - Requirements-based tests
    - Exploratory testing
    - Balanced approach to Test Automation...
  - **Maintain good working relationships with Developers...**
    - Take a developer to lunch...

# Summary

---

- **Getting it right the first time is ALWAYS** the most efficient way to work
- Instead of zero-defects, focus on reducing **customer reported defects...**
- **Key areas to focus on:**
  - Requirements
  - Peer Reviews
  - Domain knowledge and story telling
  - Combination of testing approaches
  - Conceptual Integrity

# All Software is Defective

---

- There's always an exception...

```
main ()  
{  
    printf ("hello, ");  
    printf ("world");  
    printf ("\n");  
}
```



Kernighan, B. and Ritchie, D., The C Programming Language, Prentice-Hall, 1978, p. 7

# Workshops from Software Quality Consulting

---

- **Writing Software Requirements**
- **Building Realistic Project Schedules from Software Requirements**
- **Software Verification & Validation**
- **Predictable Software Development**
- **Peer Reviews and Inspections**
- **Project Retrospectives**
- **Root Cause Analysis for Customer Reported Problems**
- **Improving the Effectiveness of Testing**
- **For more information, please visit**
  - [www.swqual.com](http://www.swqual.com)



# E-newsletter

---



**Current topics of interest to developers and testers  
Opt-in at [www.swqual.com](http://www.swqual.com) to receive Food for Thought™**

## **Recent e-newsletter topics:**

- **"Design for Testability"**
- **"All Software is Defective"**
- **"What's the Story? Storytelling as a Critical Testing Skill"**
- **"Agile Methods - Beyond the Hype"**
- **"Is Your Software Development Process Predictable?"**
- **"To Release or Not To Release"**
- **"Estimating and Scheduling - Do You Believe in Magic?"**
- **"It's the Requirements, Stupid!"**

Food for Thought is a trademark of Software Quality Consulting, Inc.

---

# Thank you...

---

**...for taking time to attend this seminar.**

**If you have any questions, please call or e-mail...**

## **Software Quality Consulting Inc.**

---

**Steven R. Rakitin**  
President

- Consulting
- Training
- Auditing

**Phone:** 508.529.4282

**Fax:** 508.529.7799

[www.swqual.com](http://www.swqual.com)

[info@swqual.com](mailto:info@swqual.com)