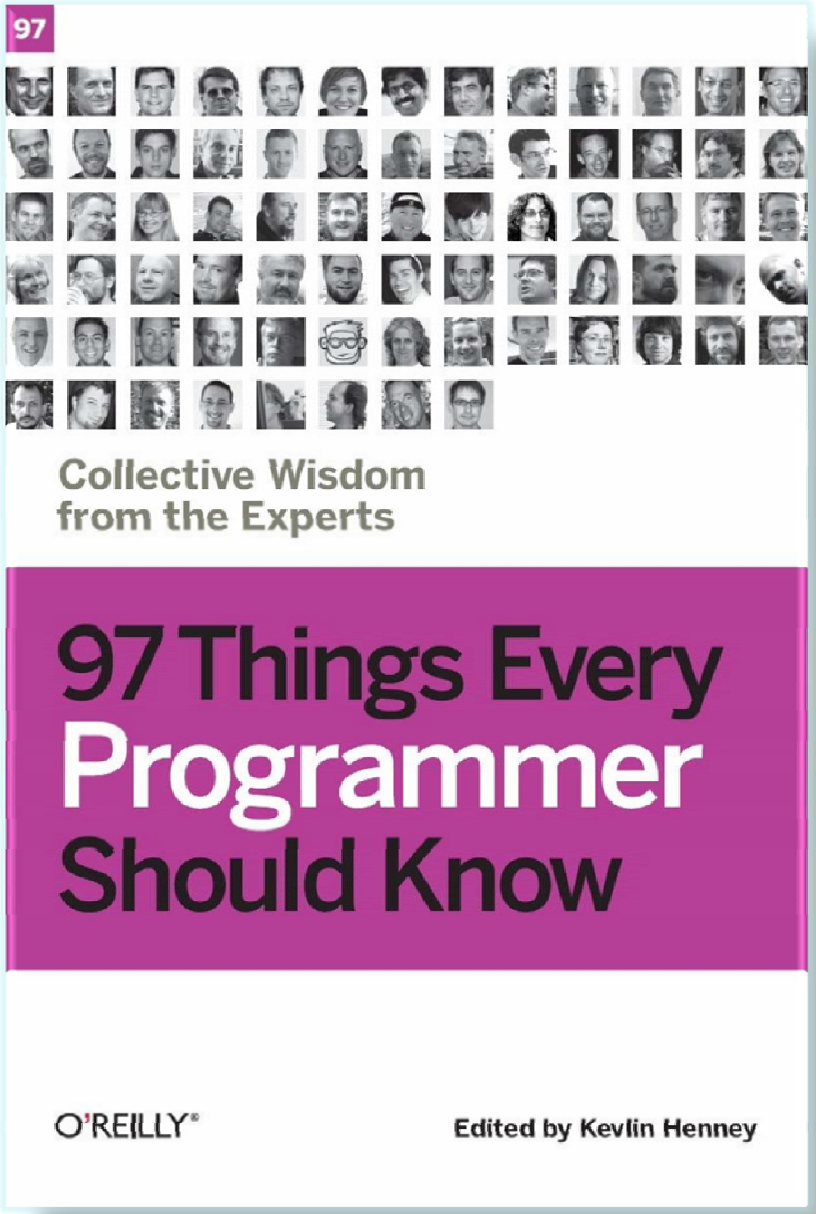
# 97 Things Every Programmer Should Know

*http://programmer.97things.oreilly.com*
*@97TEPSK*

**Kevlin Henney**
*kevlin@curbralan.com*
*@KevlinHenney*


97
Collective Wisdom
from the Experts
97 Things Every
Programmer
Should Know
O'REILLY
Edited by Kevlin Henney

Adrian Wible
Alan Griffiths
Alex Miller
Allan Kelly
Anders Norås
Ann Katrin Gagnat
Aslam Khan
Burk Hufnagel
Cal Evans
Carroll Robinson
Cay Horstmann
Chuck Allison
Clint Shank
Dan Bergh Johnsson
Dan North
Daniel Lindner
Diomidis Spinellis
Edward Garson
Einar Landre
Filip van Laenen
Gerard Meszaros
Giles Colborne
Giovanni Asproni
Greg Colvin
Gregor Hohpe
Gudny Hauknes
Heinz Kabutz
Jan Christiaan "JC" van Winkel
Janet Gregory
Jason P Sage
Johannes Brodwall
Jon Jagger
Jørn Ølmheim
Kari Røssland
Karianne Berg
Keith Braithwaite
Kevlin Henney
Kirk Pepperdine
Klaus Marquardt
Linda Rising
Marcus Baker
Matt Doar
Mattias Karlsson
Michael Feathers
Michael Hunger
Mike Lewis
Nate Jackson
Neal Ford
Niclas Nilsson
Olve Maudal
Paul W Homer
Pete Goodliffe
Peter Sommerlad
Rajith Attapattu
Randy Stafford
Richard Monson-Haefel
Robert C Martin (Uncle Bob)
Rod Begbie
Russel Winder
Ryan Brush
Sam Saariste
Sarah Mount
Scott Meyers
Seb Rose
Steve Berczuk
Steve Freeman
Steve Smith
Thomas Guest
Udi Dahan
Verity Stob
Walter Bright
Yechiel Kimchi
Yuriy Zubarev

Act with Prudence
Apply Functional Programming Principles
Ask "What Would the User Do?" (You Are Not the User)
Automate Your Coding Standard
Beauty Is in Simplicity
Before You Refactor
Beware the Share
The Boy Scout Rule
Check Your Code First Before Looking to Blame Others
Choose Your Tools with Care
Code in the Language of the Domain
Code Is Design
Code Layout Matters
Code Reviews
Coding with Reason
A Comment on Comments
Comment Only What the Code Cannot Say
Continuous Learning
Convenience Is Not an –ility
Deploy Early and Often
Distinguish Business Exceptions from Technical
Do Lots of Deliberate Practice
Domain-Specific Languages
Don't Be Afraid to Break Things
Don't Be Cute with Your Test Data
Don't Ignore That Error!
Don't Just Learn the Language, Understand its Culture
Don't Nail Your Program into the Upright Position
Don't Rely on "Magic Happens Here"
Don't Repeat Yourself
Don't Touch That Code!
Encapsulate Behavior, Not Just State
Floating-Point Numbers Aren't Real
Fulfill Your Ambitions with Open Source
The Golden Rule of API Design
The Guru Myth
Hard Work Does Not Pay Off
How to Use a Bug Tracker
Improve Code by Removing It
Install Me
Inter-Process Communication Affects Application Response Time
Keep the Build Clean
Know How to Use Command-line Tools
Know Well More than Two Programming Languages
Know Your IDE
Know Your Limits
Know Your Next Commit
Large Interconnected Data Belongs to a Database
Learn Foreign Languages
Learn to Estimate
Learn to Say "Hello, World"
Let Your Project Speak for Itself
The Linker Is Not a Magical Program
The Longevity of Interim Solutions
Make Interfaces Easy to Use Correctly and Hard to Use Incorrectly
Make the Invisible More Visible
Message Passing Leads to Better Scalability in Parallel Systems
A Message to the Future
Missing Opportunities for Polymorphism
News of the Weird: Testers Are Your Friends
One Binary
Only the Code Tells the Truth
Own (and Refactor) the Build
Pair Program and Feel the Flow
Prefer Domain-Specific Types to Primitive Types
Prevent Errors
The Professional Programmer
Put Everything Under Version Control
Put the Mouse Down and Step Away from the Keyboard
Read Code
Read the Humanities
Reinvent the Wheel Often
Resist the Temptation of the Singleton Pattern
The Road to Performance Is Littered with Dirty Code Bombs
Simplicity Comes from Reduction
The Single Responsibility Principle
Start from Yes
Step Back and Automate, Automate, Automate
Take Advantage of Code Analysis Tools
Test for Required Behavior, Not Incidental Behavior
Test Precisely and Concretely
Test While You Sleep (and over Weekends)
Testing Is the Engineering Rigor of Software Development
Thinking in States
Two Heads Are Often Better than One
Two Wrongs Can Make a Right (and Are Difficult to Fix)
Ubuntu Coding for Your Friends
The Unix Tools Are Your Friends
Use the Right Algorithm and Data Structure
Verbose Logging Will Disturb Your Sleep
WET Dilutes Performance Bottlenecks
When Programmers and Testers Collaborate
Write Code as If You Had to Support It for the Rest of Your Life
Write Small Functions Using Examples
Write Tests for People
You Gotta Care About the Code
Your Customers Do Not Mean What They Say

Act with Prudence
Apply Functional Programming Principles
Ask "What Would the User Do?" (You Are Not the User)
Automate Your Coding Standard
Beauty Is in Simplicity
Before You Refactor
Beware the Share
The Boy Scout Rule
Check Your Code First Before Looking to Blame Others
Choose Your Tools with Care
Code in the Language of the Domain
Code Is Design
Code Layout Matters
Code Reviews
Coding with Reason
A Comment on Comments
Comment Only What the Code Cannot Say
Continuous Learning
Convenience Is Not an –ility
Deploy Early and Often
Distinguish Business Exceptions from Technical
Do Lots of Deliberate Practice
Domain-Specific Languages
Don't Be Afraid to Break Things
Don't Be Cute with Your Test Data
Don't Ignore That Error!
Don't Just Learn the Language, Understand its Culture
Don't Nail Your Program into the Upright Position
Don't Rely on "Magic Happens Here"
Don't Repeat Yourself
Don't Touch That Code!
Encapsulate Behavior, Not Just State
Floating-Point Numbers Aren't Real
Fulfill Your Ambitions with Open Source
The Golden Rule of API Design
The Guru Myth
Hard Work Does Not Pay Off
How to Use a Bug Tracker
Improve Code by Removing It
Install Me
Inter-Process Communication Affects Application Response Time
Keep the Build Clean
Know How to Use Command-line Tools
Know Well More than Two Programming Languages
Know Your IDE
Know Your Limits
Know Your Next Commit
Large Interconnected Data Belongs to a Database
Learn Foreign Languages
Learn to Estimate
Learn to Say "Hello, World"
Let Your Project Speak for Itself
The Linker Is Not a Magical Program
The Longevity of Interim Solutions
Make Interfaces Easy to Use Correctly and Hard to Use Incorrectly
Make the Invisible More Visible
Message Passing Leads to Better Scalability in Parallel Systems
A Message to the Future
Missing Opportunities for Polymorphism
News of the Weird: Testers Are Your Friends
One Binary
Only the Code Tells the Truth
Own (and Refactor) the Build
Pair Program and Feel the Flow
Prefer Domain-Specific Types to Primitive Types
Prevent Errors
The Professional Programmer
Put Everything Under Version Control
Put the Mouse Down and Step Away from the Keyboard
Read Code
Read the Humanities
Reinvent the Wheel Often
Resist the Temptation of the Singleton Pattern
The Road to Performance Is Littered with Dirty Code Bombs
Simplicity Comes from Reduction
The Single Responsibility Principle
Start from Yes
Step Back and Automate, Automate, Automate
Take Advantage of Code Analysis Tools
Test for Required Behavior, Not Incidental Behavior
Test Precisely and Concretely
Test While You Sleep (and over Weekends)
Testing Is the Engineering Rigor of Software Development
Thinking in States
Two Heads Are Often Better than One
Two Wrongs Can Make a Right (and Are Difficult to Fix)
Ubuntu Coding for Your Friends
The Unix Tools Are Your Friends
Use the Right Algorithm and Data Structure
Verbose Logging Will Disturb Your Sleep
WET Dilutes Performance Bottlenecks
When Programmers and Testers Collaborate
Write Code as If You Had to Support It for the Rest of Your Life
Write Small Functions Using Examples
Write Tests for People
You Gotta Care About the Code
Your Customers Do Not Mean What They Say

# Do Lots of Deliberate Practice

***Jon Jagger***

*You do deliberate practice to improve your ability to perform a task. It's about skill and technique. Deliberate practice means repetition. It means performing the task with the aim of increasing your mastery of one or more aspects of the task. It means repeating the repetition. Slowly, over and over again, until you achieve your desired level of mastery. You do deliberate practice to master the task, not to complete the task.*

# Learn to Estimate

*Giovanni Asproni*

- An **estimate** is an approximate calculation or judgement of the value, number, quantity, or extent of something. This definition implies that [...] hopes and wishes must be ignored when calculating it. The definition also implies that, being approximate, an estimate cannot be precise, e.g., a development task cannot be estimated to last 234.14 days.
- A **target** is a statement of a desirable business objective, e.g., "The system must support at least 400 concurrent users."
- A **commitment** is a promise to deliver specified functionality at a certain level of quality by a certain date or event.

# Know Your Next Commit

*Dan Bergh Johnsson*

I tapped three programmers on their shoulders and asked what they were doing. “I am refactoring these methods,” the first answered. “I am adding some parameters to this web action,” the second answered. The third answered, “I am working on this user story.”

It might seem that the first two were engrossed in the details of their work, while only the third could see the bigger picture, and that he had the better focus. However, when I asked when and what they would commit, the picture changed dramatically. The first two were pretty clear about what files would be involved, and would be finished within an hour or so. The third programmer answered, “Oh, I guess I will be ready within a few days. I will probably add a few classes and might change those services in some way.”

# Comment Only What the Code Cannot Say

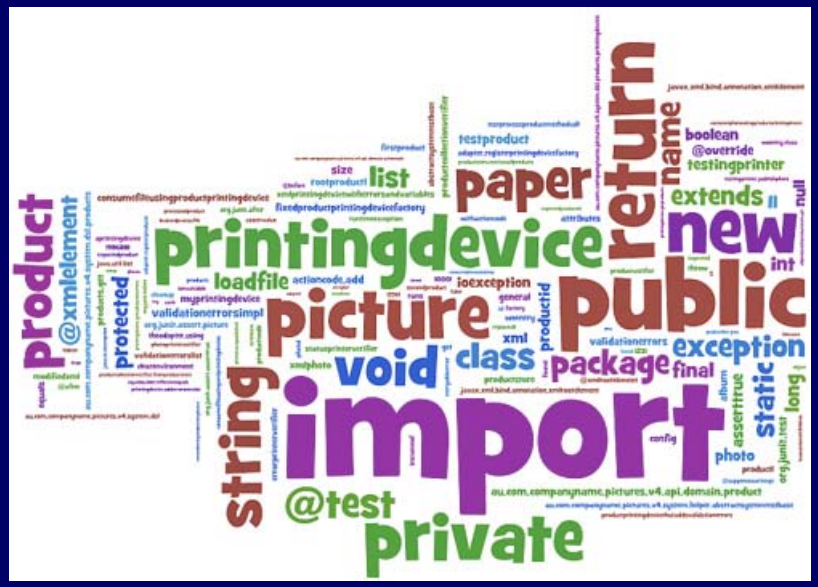
***Kevlin Henney***

1. *If a program is incorrect, it matters little what the documentation says.*
2. *If documentation does not agree with the code, it is not worth much.*
3. *Consequently, code must largely document itself. If it cannot, rewrite the code rather than increase the supplementary documentation. Good code needs fewer comments than bad code does.*
4. *Comments should provide additional information that is not readily obtainable from the code itself. They should never parrot the code.*
5. *Mnemonic variable names and labels, and a layout that emphasizes logical structure, help make a program self-documenting.*

**Kernighan and Plauger**
***The Elements of Programming Style***

# Code in the Language of the Domain

*Dan North*


**Phillip Calçado**
***http://fragmental.tw/2009/04/29/tag-clouds-see-how-noisy-your-code-is/***

# Encapsulate Behavior, Not Just State

*Einar Landre*

> An object encapsulates both state and behavior, where the behavior is defined by the actual state. [...] This inherent property of an object makes the design process conceptually simple. It boils down to two simple tasks: allocation and delegation of responsibility to the different objects including the interobject interaction protocols.

# Don't Repeat Yourself

*Steve Smith*

Duplication Is Waste

Repetition in Process Calls for Automation

Repetition in Logic Calls for Abstraction

# Beware the Share

***Udi Dahan***

The fact that two wildly different parts of the system performed some logic in the same way meant less than I thought. Up until I had pulled out those libraries of shared code, these parts were not dependent on each other. Each could evolve independently. Each could change its logic to suit the needs of the system's changing business environment. Those four lines of similar code were accidental—a temporal anomaly, a coincidence. That is, until I came along.

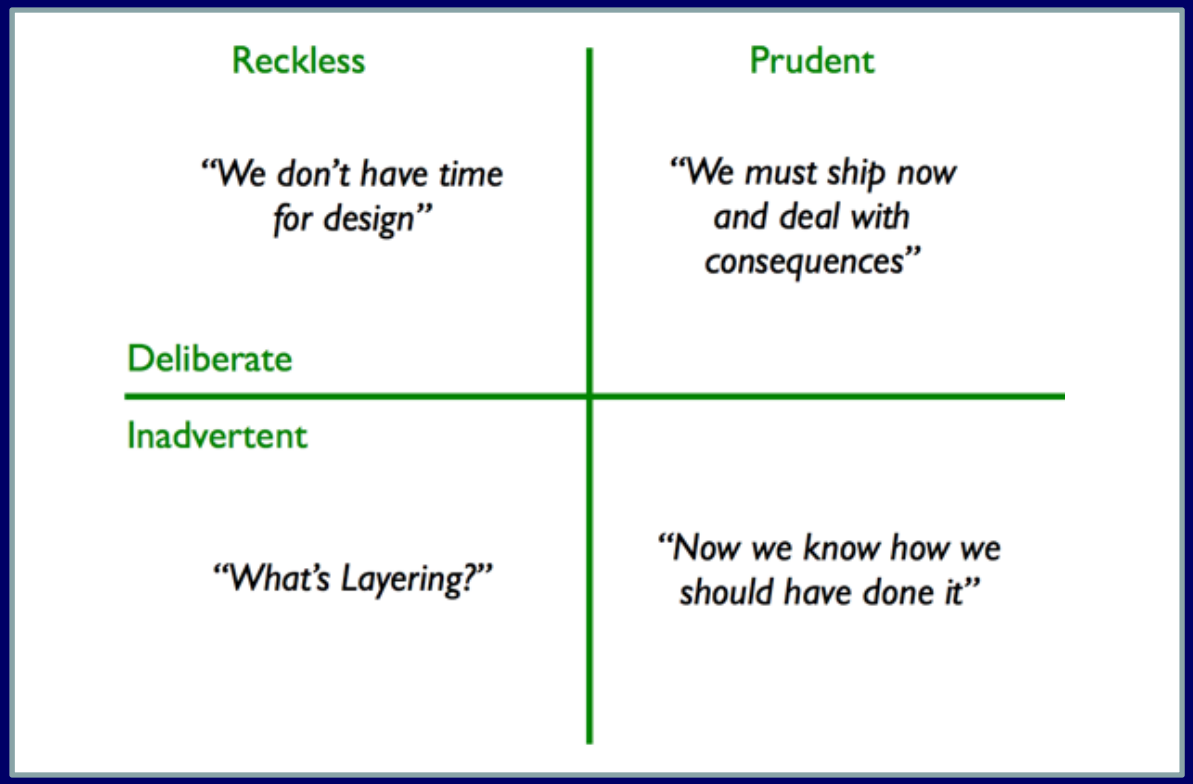
# The Road to Performance Is Littered with Dirty Code Bombs

*Kirk Pepperdine*

MORE OFTEN THAN NOT, PERFORMANCE TUNING A SYSTEM REQUIRES YOU TO ALTER CODE. WHEN WE NEED TO ALTER CODE, EVERY CHUNK THAT IS OVERLY COMPLEX OR HIGHLY COUPLED IS A DIRTY CODE BOMB LYING IN WAIT TO DERAIL THE EFFORT. THE FIRST CASUALTY OF DIRTY CODE WILL BE YOUR SCHEDULE.

# Act with Prudence

***Seb Rose***




**Martin Fowler**
***http://martinfowler.com/bliki/TechnicalDebtQuadrant.html***

# The Boy Scout Rule

***Robert C Martin (Uncle Bob)***

*Try and leave this world a little better than you found it.*

**Robert Stephenson Smyth Baden-Powell**

# Apply Functional Programming Principles

*Edward Garson*

An expression is said to be referentially transparent if it can be replaced with its value without changing the program (in other words, yielding a program that has the same effects and output on the same input). [...]

The importance of referential transparency is that it allows a programmer (or compiler) to reason about program behavior. This can help in proving correctness, simplifying an algorithm, assisting in modifying code without breaking it, or optimizing code by means of memoization, common subexpression elimination or parallelization.

http://en.wikipedia.org/wiki/Referential_transparency_(computer_science)

# Thinking in States

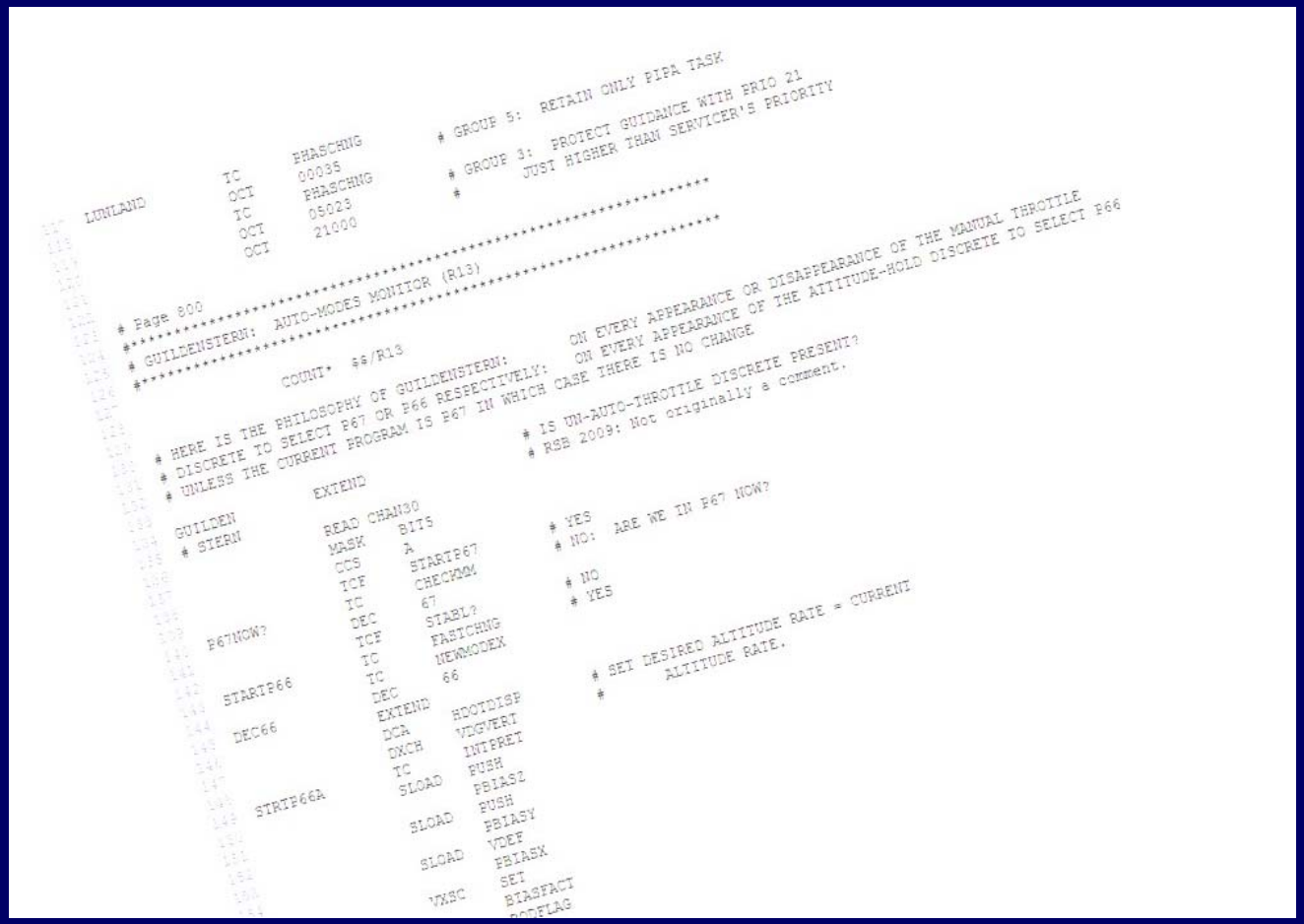
*Niclas Nilsson*

People in the real world have a weird relationship with state.

In most real-world situations, people's relaxed attitude toward state is not an issue. Unfortunately, however, many programmers are quite vague about state too — and that is a problem.

# Two Wrongs Can Make a Right (and Are Difficult to Fix)

*Allan Kelly*



# Code Reviews

*Mattias Karlsson*

Making code reviews fun is perhaps the most important contributor to success. Reviews are about the people reviewing. If the review meeting is painful or dull, it will be hard to motivate anyone. Make it an informal code review whose principal purpose is to share knowledge among team members. Leave sarcastic comments outside, and bring a cake or brown-bag lunch instead.

# Testing Is the Engineering Rigor of Software Development

*Neal Ford*

Testing “hard” things is tough because you have to build them to test them, which discourages speculative building just to see what will happen. But the building process in software is ridiculously cheap.

# Write Tests for People

*Gerard Meszaros*

So *who* should you be writing the tests for? For the person trying to understand your code.

Good tests act as documentation for the code they are testing. They describe how the code works. For each usage scenario, the test(s):

- Describe the context, starting point, or preconditions that must be satisfied
- Illustrate how the software is invoked
- Describe the expected results or postconditions to be verified

Different usage scenarios will have slightly different versions of each of these.

# Don't Be Cute with Your Test Data

*Rod Begbie*

# Ask "What Would the User Do?" (You Are Not the User)

***Giles Colborne***

We all tend to assume that other people think like us. But they don't. Psychologists call this the *false consensus bias*.

This bias explains why programmers have such a hard time putting themselves in the users' position. Users don't think like programmers.

The best way to find out how a user thinks is to watch one.

# Ubuntu Coding for Your Friends

*Aslam Khan*

Umuntu ngumuntu ngabantu

*The newest computer can merely compound, at speed, the oldest problem in the relations between human beings, and in the end the communicator will be confronted with the old problem, of what to say and how to say it.*

**Edward R Murrow**