Practical Excellence

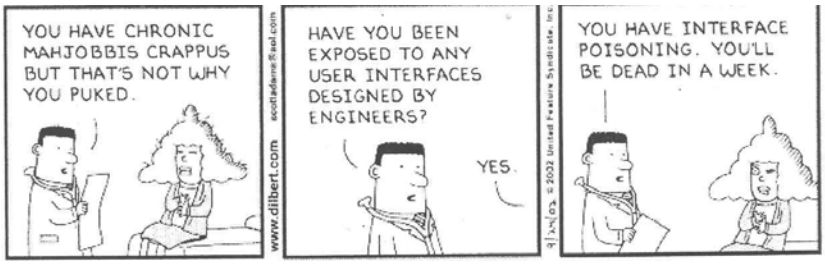
ACCU - 2004

A Perspective on Code Quality

Chuck Allison

The Nature of Software

- Many skills are required to produce quality software
- These skills are not possessed by a single individual



BANGKOK (Reuters – May 12, 2003) – Security guards smashed their way into an official limousine with sledgehammers on Monday to rescue Thailand's finance minister after his car's computer failed… All doors and windows had locked automatically when the computer crashed, and the air-conditioning stopped, officials said. "We could hardly breathe for over 10 minutes… It took my guard a long time to realize that we really wanted the window smashed so that we could crawl out. It was a harrowing experience."

"Malfunctions caused by bizarre and frustrating glitches are becoming harder and harder to escape now that software controls everything from stoves to cell phones, trains, cars, and power plants."

-- "Spread of buggy software raises new questions", CNN.com, April 27, 2003.

Summit on Code Quality

- Portland, Oregon, January 15-17, 2003
- Agenda:
  - How bad are things now?
  - How did they get that way?
  - How can they be fixed?

Attendees

- Scott Meyers
- Bruce Eckel
- Chuck Allison
- Bill Venners
- Joshua Bloch
- Alistair Cockburn
- Matt Gerrans
- Kevlin Henney
- Andrew Hunt
- Angelika Langer
- Pete McBreen
- Chris Sells
- Randy Stafford
- Dave Thomas

How Bad Are Things Now?

- Scott Meyers is Upset
  - that should be enough to get our attention!

How Bad Are Things Now?

- Scott Meyers is Upset
  - that should be enough to get our attention!
- So are a lot of other people:
  - ~50% of projects never get deployed
  - too many programs hang or fail to meet spec.
  - schedule/budget overruns are a cliche'
  - help desks are woefully understaffed and overworked
  - NIST* estimates the cost of software bugs in the U.S. is $60 billion annually

* (National Institute of Standards and Technology)
http://www.nist.gov/director/prog-ofc/report02-3.pdf

Software Disasters

- Mars Polar Lander
  - www.nasa.gov/newsinfo/marsreports.html
- American Airlines Flight 965, Dec. 1995
  - see "The Inmates are Running the Asylum" by Alan Cooper
- $60,000,000,000 wasted annually is a disaster!

How Did Things Get So Bad?

- The effect of:
  - Management
  - Development tools
  - Programmer skill level
  - … among other things

Haste Makes Waste

- "The purpose of publishing this piece of code is to let you see that working under pressure can result in a marked deterioration of code quality."

  -- Francis Glassborow, *You Can Do It!*, page 307.

Stupid Management Tricks

- Expect the impossible from the under-supported
  - micro-manage scheduling and technical decisions
  - demand overtime

Stupid Management Tricks

- Expect the impossible from the under-supported
  - micro-manage scheduling and technical decisions
  - demand overtime
- Rush to meet a market window
  - Not always a bad thing
  - But don't allow for periodic code "clean-up"

Stupid Management Tricks

- Expect the impossible from the under-supported
  - micro-manage scheduling and technical decisions
  - demand overtime
- Rush to meet a market window
  - Not always a bad thing
  - But don't allow for periodic code "clean-up"
- Don't support needed training/mentoring
  - My experiences with NASA, Hughes Aircraft

"I have met managers who would rather fail knowing whom to blame rather than succeed without knowing whom to credit."

-- Gerald Weinberg, Keynote Address, *Agile Development conference*, August 2003

You are very, very scary

- People have been trying to eliminate programmers for years
  - It's Black Magic
- COBOL
- 4GLs
- Visual BASIC

The Deception of Visual, Event-driven Programming

- Only deals with UI
- Seems simple, but encourages poor practice
  - Prototypes become products before their time
  - UI should be totally separate from object model
  - Programming is not Mouse Engineering
- "Who Moved My State", C/C++ Users Journal, April 2003, Miro Samek
- Programmers don't need to design GUIs
  - Let the users do it!

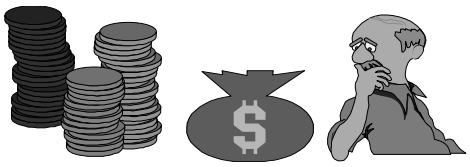
"I Click, Therefore I Program"

- Tools like Visual BASIC have "lowered the bar" for entering the field of programming
  - "90-day wonders" have filled the ranks
    - thanks in part to the dot-com boom
  - They build screens, not software
- Their numbers exceed our capacity to properly mentor them
  - We're losing this "Battle of the Exponents"
  - "Dumbing down" programming has back-fired

The Tidal Wave of Trainees
(from Alistair Cockburn)

- Training:
- 3 weeks of training per person @ $2,000 /week
- 6 work months / person relearning
- Programmers in company:

6	200	3,000	programmers
36K$	1.2M$	18M$	direct training cost
5	100	1,500	work-yrs spent relearning



Programming is more than Visual

- "The effective exploitation of his powers of abstraction must be regarded as one of the most vital activities of a competent programmer." -- Dijkstra

Programming will always be difficult

"Programming will remain very difficult, because once we have freed ourselves from the circumstantial cumbersomeness, we will find ourselves free to tackle the problems that are now well beyond our programming capacity." -- Dijkstra

The Degradation of Software Quality

"Architecture degradation begins simply enough. When market pressures for key features are high and the needed capabilities to implement them are missing, an otherwise sensible engineering manager may be tempted to coerce the development team into implementing the requested features without the requisite architectural capabilities."

-- Luke Hohmann, *Beyond Software Architecture*

Technical Debt

- Market and budget pressures can "require" cutting corners
  - This shouldn't be the "norm", but it is
- This constitutes a technical "debt"
  - The software is not in a state suitable for long-term maintenance
  - Such a system must be abandoned before its time

Software Entropy

- Entropy:
  - A measure of the disorder in a closed system.
  - Inevitable and steady deterioration of a system.
- Software Entropy
  - Code can easily become an unmanageable patchwork
  - Complexity encourages disorder

How do you fight entropy?

Rearchitecting/Refactoring

Paying-off your Technical Debt

- Improving the internal design of existing code
- Examples:
  - Add parameter
  - Extract class
  - Extract hierarchy
  - Extract method
  - Substitute algorithm
  - Replace parameter with method…

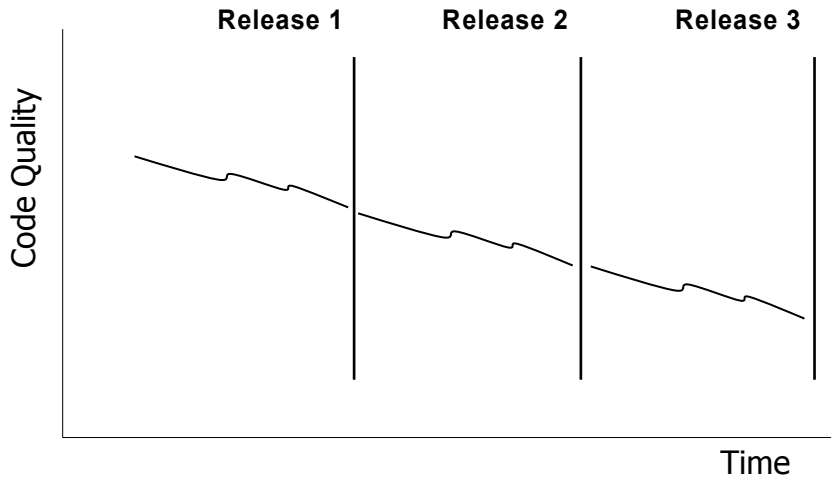
Post-release Entropy Reduction

(Luke Hohmann)

- No time for refactoring near release
  - Quick hacks pay off near release time
  - Market windows *are* important
- There must be time to refactor after release
  - Otherwise entropy kills you prematurely
  - Yes, there is a short-term cost
  - The long-term reduction of technical debt is worth it!

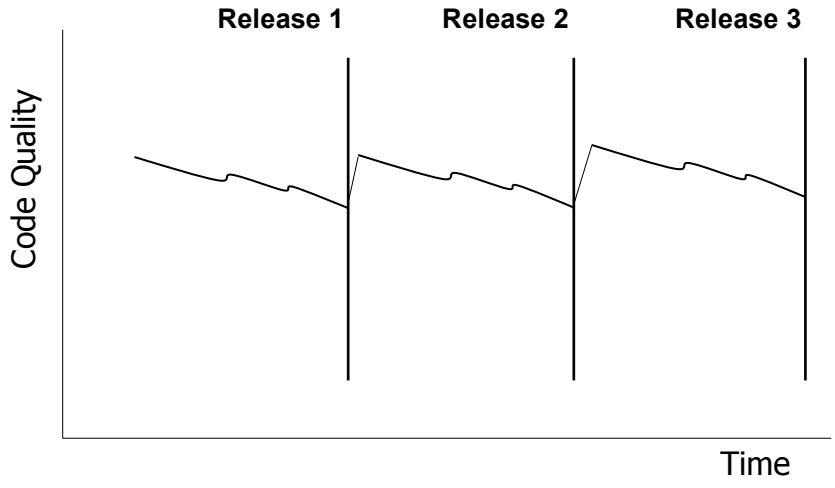
Code Quality without PRER

(suggested by Randy Stafford)



Code Quality with PRER

(suggested by Randy Stafford)

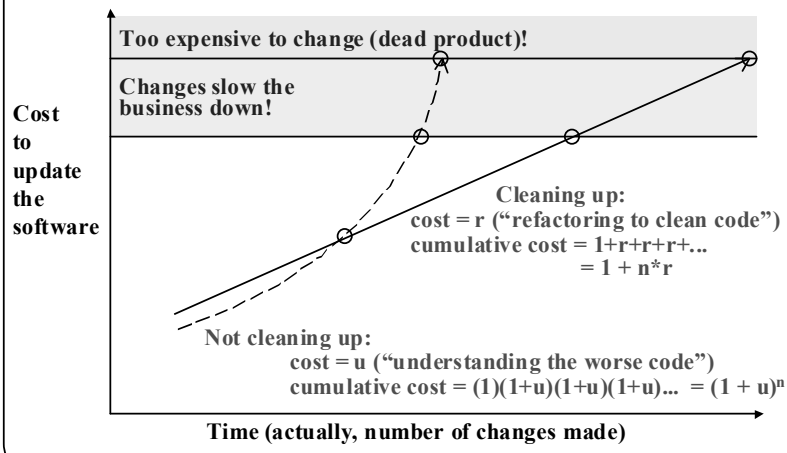


Technical Debts Interact

(from Alistair Cockburn)

- First maintenance visit: 1 + u
- Second visit: 1 + u + v + uv (interaction)
  - approximates $1 + 2u + u^2$
- Third visit: 1 + u + v + w + uv + uw + vw + uvw
  - approximates $1 + 3u + 3u^2 + u^3 = (1 + u)^3$

Poor code quality penalizes exponentially;
Cleaning up penalizes linearly.



True or False?

- "It is just as easy to write good software as it is to write bad software"

True or False?

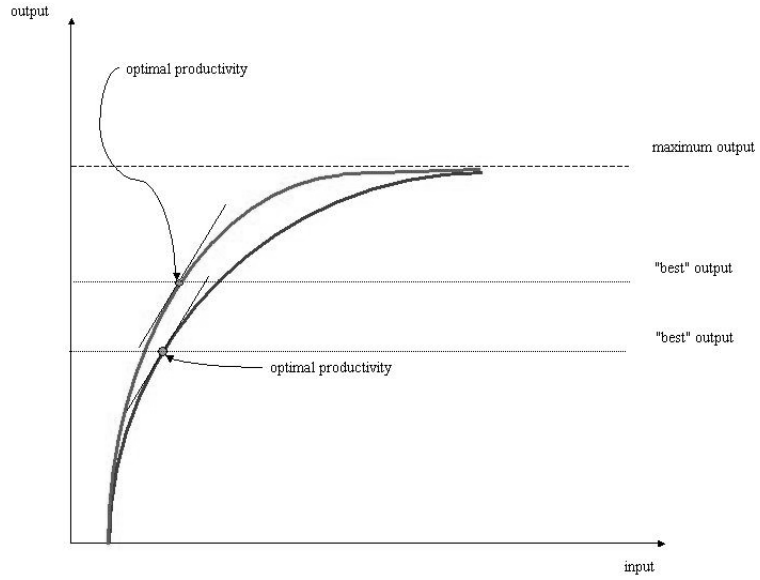
- "It is just as easy to write good software as it is to write bad software"…
- It is if you know how!

Don't prevent bugs, avoid them

- "Those who want really reliable software will discover that they must find means of avoiding the majority of bugs to start with, and as a result the programming process will become cheaper. *If you want more effective programmers, you will discover that they should not waste their time debugging, they should not introduce the bugs to start with.*" – Dijkstra

The Productivity Curve

(from Angelika Langer)



The Cost of Competence

- Less than the cost of incompetence
  - Assertion: Changing curves is cheaper than the consequences of producing poor quality
    - Proof: Japanese Auto Industry

The Cost of Competence

- Less than the cost of incompetence
  - Assertion: Changing curves is cheaper than the consequences of producing poor quality
    - Proof: Japanese Auto Industry
- Mentoring vs. Training
  - A trainer loves 'em and leaves 'em
    - "drinking from the fire hose" approach
    - although this can be a good "jump start"
  - A mentor takes an apprentice under his wing
    - "Expert within earshot"

Stupid Programmer Tricks

- Poor design
  - poor choice of algorithm or data structure
  - not enough critical thinking up front
    - bug *prevention* is better than cure

Stupid Programmer Tricks

- Poor design
  - poor choice of algorithm or data structure
  - not enough critical thinking up front
    - bug *prevention* is better than cure
- Reinventing the Proverbial Wheel
  - insufficient mastery of library, tools, and techniques

Stupid Programmer Tricks

- Poor design
  - poor choice of algorithm or data structure
  - not enough critical thinking up front
    - bug *prevention* is better than cure
- Reinventing the Proverbial Wheel
  - insufficient mastery of library, tools, and techniques
- Many don't continue to learn
  - stuck in their initial "imprint"

Stupid Programmer Tricks

- Poor design
  - poor choice of algorithm or data structure
  - not enough critical thinking up front
    - bug *prevention* is better than cure
- Reinventing the Proverbial Wheel
  - insufficient mastery of library, tools, and techniques
- Many don't continue to learn
  - stuck in their initial "imprint"
- Copy-and-paste "reuse"
  - automated bug replication!

Stupid Programmer Tricks

- Poor design
  - poor choice of algorithm or data structure
  - not enough critical thinking up front
    - bug *prevention* is better than cure
- Reinventing the Proverbial Wheel
  - insufficient mastery of library, tools, and techniques
- Many don't continue to learn
  - stuck in their initial "imprint"
- Copy-and-paste "reuse"
  - automated bug replication!
- Inadequate testing/error handling

The Evils of Code Duplication

- Two Forms
  - Explicit:
    - You copy and paste code (bugs)
    - You should've extracted a function to call
  - Implicit:
    - You create similar functions
    - You should extract the common part
- Becomes a maintenance nightmare
- Example: Manifest constants, C++ string class

Manifest Constants

- Better to use symbols instead of magic numbers
  - Only have to change them in one place
  - No risk of getting one of the values wrong
- *All code* should follow this pattern:
  - Have only one definition
  - A form of "automation"
    - Because you're not repeating yourself!

A C++ String Class

- Constructor allocates memory
- Destructor releases it:

```
class String
{
   char* data;
public:
   String(const char* str = "") {
      data = new char[strlen(str) + 1];
      strcpy(data,str);
   }
   ~String() {
      delete [] data;
   }
};
```

A C++ String Class

- What about copying strings?
- Need a Copy and Assignment:

```
   String(const String& s) {
      data = new char[strlen(s.data) + 1];
      strcpy(data, s.data);
   }
   String& operator=(const String& s) {
      if (this != &s) {
         char* newData = new char[strlen(s.data) + 1];
         strcpy(newData, s.data);
         delete [] data;
         data = newData;
      }
      return *this;
   }
```

A C++ String Class

- Factor out common code into a private member function:

```
char* dup(const char* from) {
     char* to = new char[strlen(from) + 1];
     return strcpy(to, from);
  }
```

An aside: C++0x will have delegating constructors

A C++ String Class

- Now call it from all 3 places:

```
String(const char* str = "") {
   data = dup(str);
}
String(const String& s) {
   data = dup(s.data);
}
String& operator=(const String& s) {
   if (this != &s) {
      char* newData = dup(s.data);
      delete [] data;
      data = newData;
   }
   return *this;
}
```

Practices that Work

- Big List!
- Takes wisdom born of experience to apply:
  - DRY (don't repeat yourself)
  - find invariants and assert them
  - design in terms of interfaces
    - separate interface and implementation
  - minimize coupling, maximize cohesion
  - test-first programming
  - pair programming (review as you go)
  - automate builds, tests, deployments
  - study good code
  - keep a technical diary

Practices that Work

- (Continued…)
  - write to be read
    - use a consistent style
    - publish your code somewhere
  - avoid premature optimization
    - use a profiler
  - have post-partum reviews
  - design in public places (white board)
  - refactor mercilessly
    - "remove to improve" (omit needless code)
  - take pride in your work

Revealing Questions

- Are any of the practices news to you?
  - probably not
- Do you employ them?
- Do your colleagues employ them?
  - mentor them

Pride Implies Ownership

- "[The Company] shall not be liable in any manner whatsoever for results obtained for using this software. THESE MATERIALS ARE PROVIDED 'AS-IS' WITHOUT WARRANTY OF ANY KIND… [THE COMPANY] AND ITS SUPPLIERS DISCLAIM ANY AND ALL WARRANTIES AND CONDITIONS, EITHER EXPRESS OR IMPLIED."
- "Bugs would be greatly reduced if software makers were held legally responsible for defects. 'Software is being treated in a way that no other consumer products are,' said Barbara Simons, former president of the ACM. 'We all know that you can't produce 100% bug-free software. But to go to the other extreme, and say that software makers should have no liability whatsoever, strikes me as absurd.'" (CNN.com)

Talent vs. Knowledge

- The practices just mentioned are more valuable than understanding the technology *du jour*
- You can know Java, XML, XSLT, Jini, JNI, J2EE, C++, STL, Bluetooth, SOAP, … and still produce low quality
- Master the enduring fundamentals first
  - Technology is a vehicle for quality, not its substance

The Great Programmer
(from Gerald Weinberg)

- Has a Service Posture
- Learns continually, no matter how skilled he is today
- Openly shares his code, insights and expertise
- Gives credit where credit is due
  - colleagues, vendors, good management, staff, family)
- Does not sacrifice quality or integrity for job security
- Takes care of his whole self (he rests, plays…)

Practical Excellence

Be True to your Inner Programmer

Highly ethical programmers do things as correctly as they can all the time. They fight management's stupid decisions. To an ethical programmer, some things (e.g., buffer overrun vulnerabilities) just cannot happen."

-- Jack Ganssle

Practical Excellence

Care, Be Connected

- "When one isn't dominated by feelings of *separateness* from what he's working on, then one can be said to "care" about what he's doing. That is what caring really is, a feeling of identification with what one's doing. When one has this feeling then he also sees the inverse side of caring, Quality itself."

-- Robert Pirsig, *Zen and the Art of Motorcycle Maintenance*

Practical Excellence

Persist in working at Meaningful Change

- "You cannot undo a [or introduce a new] habit by any single act of willpower; only a time-consuming training process can undo a habit."

  -- Jef Raskin

Practical Excellence

Keep Current

- "In times of change learners inherit the earth while the learned find themselves beautifully equipped to work in a world that no longer exists." -- Eric Hoffer

"He who learns from one who is learning drinks from a running stream."

A Challenge for ACCU Delegates

- You are the best of developers
- Set the example
- Mentor others

Announcement

C++ Online

- An Authoritative online C++ Journal
- Hosted at www.artima.com
- Articles, tutorials, archives, and more
  - peer reviewed
- Editorial Board:
  - Bjarne Stroustrup, Andrew Koenig, Scott Meyers, Bruce Eckel, Angelika Langer, Kevlin Henney, Jeremy Siek, David Abrahams, Greg Colvin, Andrei Alexandrescu, Stephen Dewhurst, Bjorn Karlsson…
- Editor: Me!
- Launch in Summer 2004