How Can SW-Engineering Education Improve SW-Quality?

Yechiel M. Kimchi

The Technion, CS Faculty

VLSI – Verification, Logic Synthesis, Israel Ltd.

A more basic question:

Can SW-Engineering Education Improve SW-Quality?

Roadmap

- **“Why Software is So Bad?”** (The question, not the answer)
- **How to Review the Coding Process?**
- **Expectations from Quality Software**
- **Some Rules for Realization of Expectations**
- **Coding Standards Guides vs. those Principles**
- **Education, Management & in-betweens.**
- **Examples and Observations**
- **Q&A**

Why Software is So Bad?

- “Why software is so bad?” (2002) [1]
- “Why Software Fails” (2005) [2]
- “The Software Conspiracy” (1999) [3]
- An Interview w. Jerry Weinberg (2001) [4]

Q. “What … major milestones of SWEng. discipline
in the last three decades?”

A. “Well, I don’t think there have been any.”

Q. “… what about … testing …?”

A. “… made them sloppier developers;
… more encouraged to throw stuff … to testing.”

Why Software is So Bad? (cont.)

- An Interview w. B. Stroustrup (2006) [5]

Q. “Why is most software so bad? …”

A. “… if software had been as bad as its reputation,
most of us would have been dead by now.”

Q. “How can we fix the mess we are in?”

A. [a full page] “In theory, …: educate our software developers better, … Reward correct, solid, and safe systems. Punish sloppiness. In reality, that’s essentially impossible. People want new fancy gadgets right now and reward people who deliver them cheaply, buggy, and first. …”

Roadmap

- “Why Software is So Bad?” (The question, not the answer)
- How to Review the Coding Process?
- Expectations from Quality Software
- Some Rules for Realization of Expectations
- Coding Standards Guides vs. those Principles
- Education, Management & in-betweens.
- Examples and Observations
- Q&A

How to Review the Coding Process?

I am reluctant to read M-LOC

So I have focused my attention on well known

Coding Standard documents

Coding standards [from Wikipedia: **Coding conventions**]

Where coding conventions have been specifically designed to produce high-quality code, and have then been formally adopted, they then become coding standards. Specific styles, irrespective of whether they are commonly adopted, do not automatically produce good quality code. It is only if they are designed to produce good quality code that they actually result in good quality code being produced, i.e., they must be very logical in every aspect of their design - every aspect justified and resulting in quality code being produced.

How to Review the Coding Process?

I have reviewed

- MISRA-C (Motor Industry Software Reliability Association)
- JSF AV C++ Coding Standards (F-35)
- Google C++ Style Guide
- Linux kernel coding style
- GNU Coding Standards

What should those be compared with
in order to find what they miss?

How to Review the Coding Process?

**What should those be compared with
in order to find what they miss?**

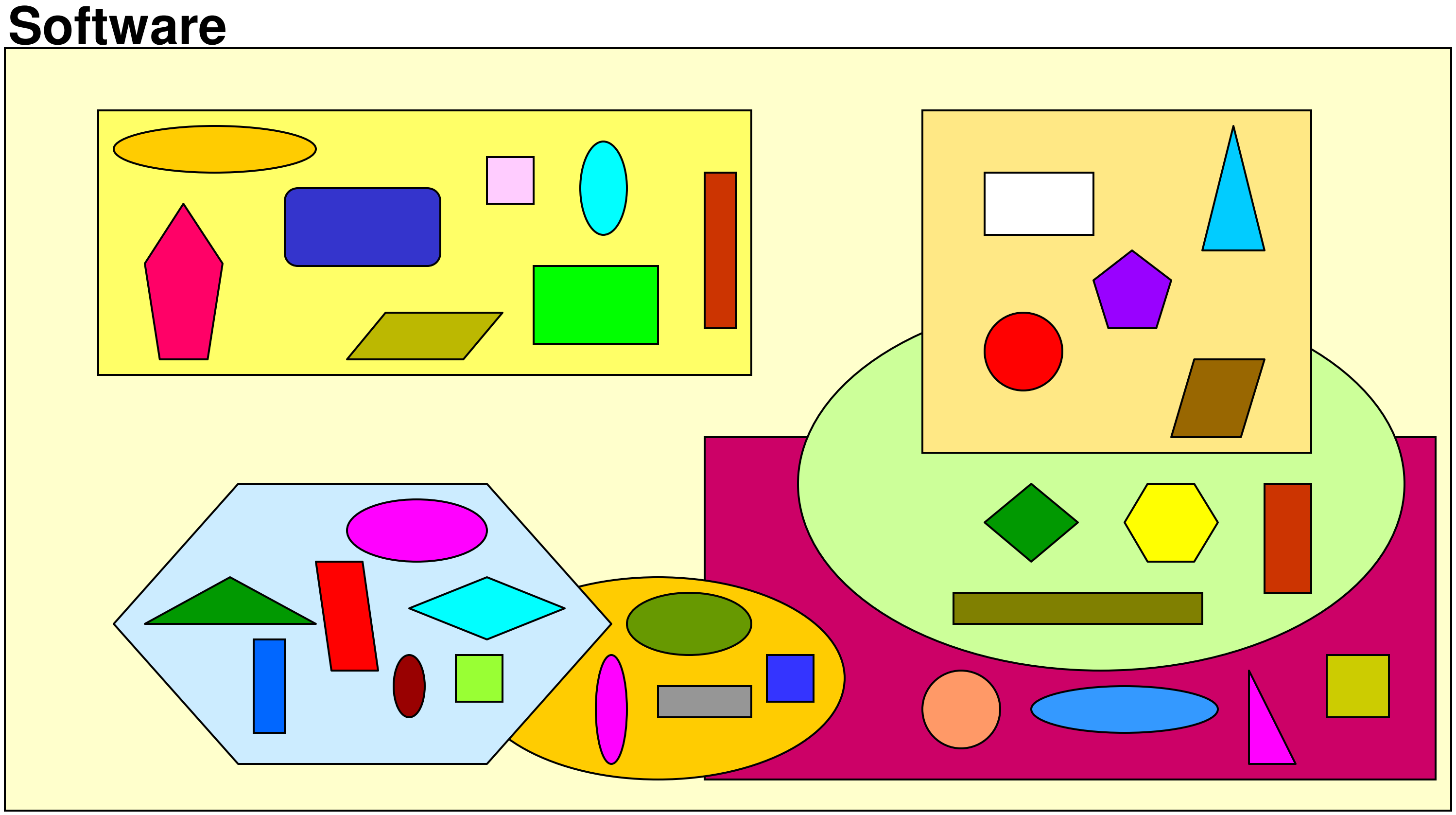
- **How about comparing with the desired “ideal”?**
  - W.r.t. my current knowledge
- **Let’s go for it**

Roadmap

- **“Why Software is So Bad?”** (The question, not the answer)
- **How to Review the Coding Process?**
- **Expectations from Quality Software**
- **Some Rules for Realization of Expectations**
- **Coding Standards Guides vs. those Principles**
- **Education, Management & in-betweens.**
- **Examples and Observations**
- **Q&A**

Expectations from Quality Software

The Structure of Software

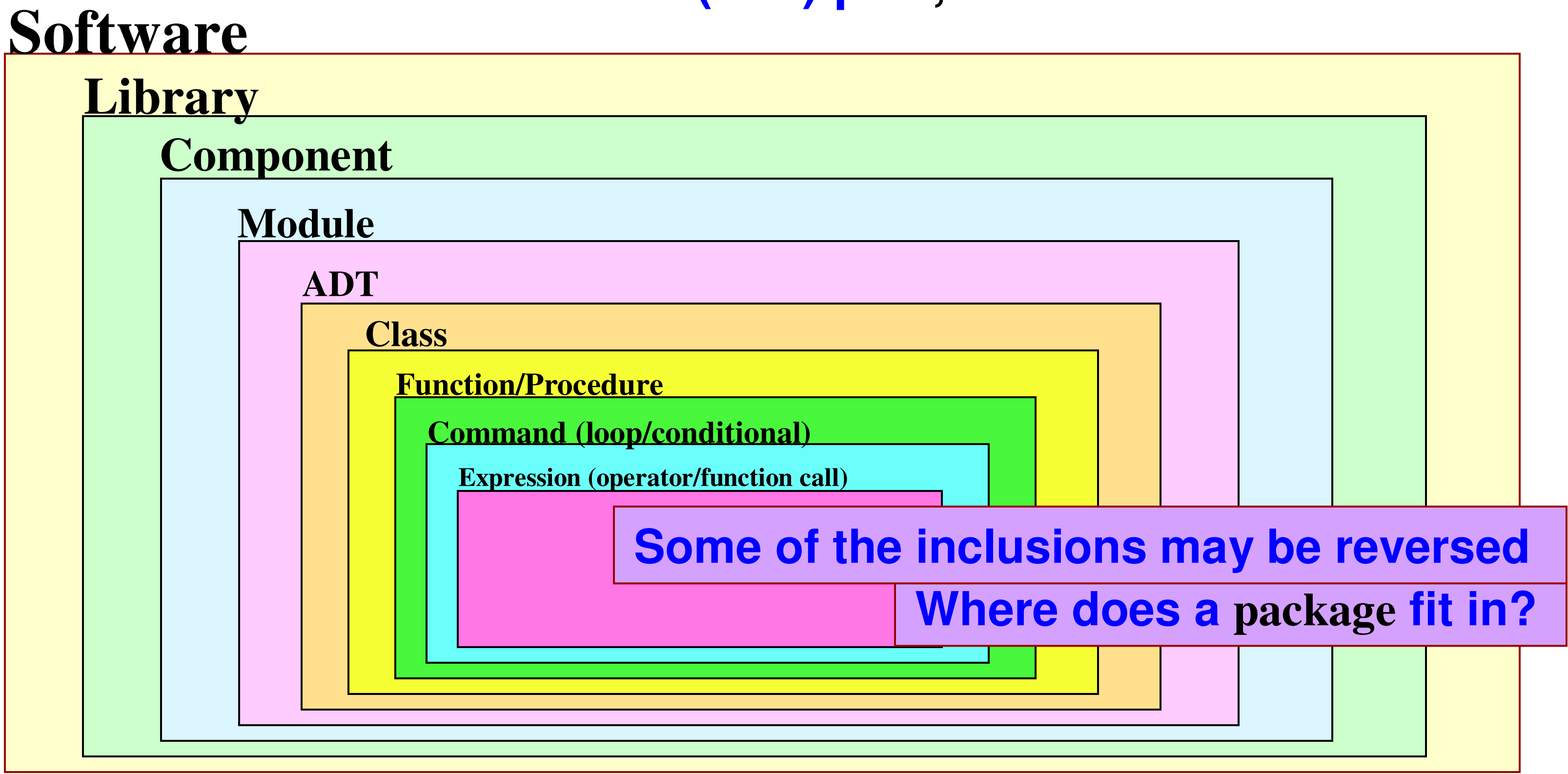


Expectations from Quality Software

Software is Fractal like

What are the recurring *Parts* ?

An atomic (leaf) part, is a *Section* of code



Software

Library

Component

Module

ADT

Class

Function/Procedure

Command (loop/conditional)

Expression (operator/function call)

Some of the inclusions may be reversed

Where does a package fit in?

Expectations from Quality Software

What is Common to these *Parts*?

Service = Interface + Implementation

Interface = Preconditions + Post Conditions
– Invariants ☺

Implementation = As Independent As Can Be(*)

(*) Independent Commands/Expressions?

An Opportunity for Concurrency

Expectations from Quality Software

Meta-Rule: ***Software is a collection of parts that are governed by same requirements***

- Independence
- Separation
- Controlled communication (Interface)
- Simplicity

These four are not at all separated/independent of one another

How can these requirements be translated into language-independent rules?

I ignore uniformity rules, which are really stylistic

Roadmap

- **“Why Software is So Bad?”** (The question, not the answer)
- **How to Review the Coding Process?**
- **Expectations from Quality Software**
- **Some Rules for Realization of Expectations**
- **Coding Standards Guides vs. those Principles**
- **Education, Management & in-betweens.**
- **Examples and Observations**
- **Q&A**

Practical Rules

Independence ⇨ No `goto`

Did you know? *There are three versions of* `goto`

- Control: The well known `goto` command
  - Allows two sections to mix t
- Value: Global **variables**
  - Allows several sections to s
  - Using a value created by an unknown section
- Type: Using ptr/ref **casting**[*]
  - It’s not a conversion, it is an assumption

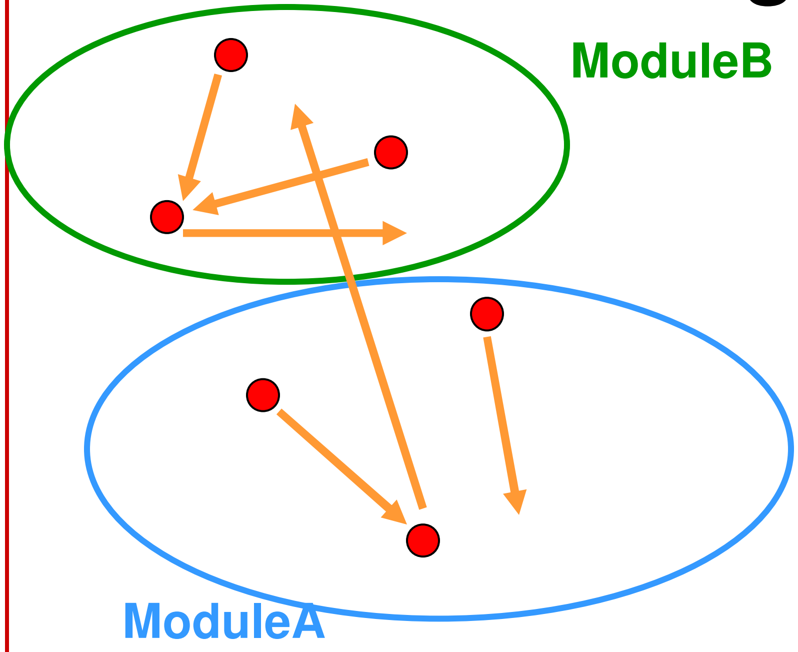
Which one is worst?

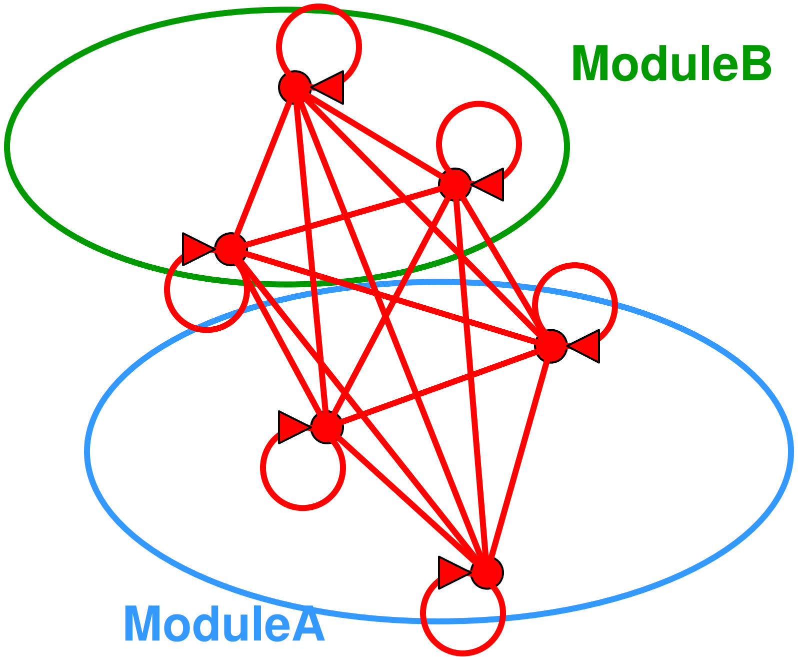
[*] Thanks to Marshall Cline, owner of C++ FAQs

Practical Rules

What’s Worse?

a cross-module goto or a global variable





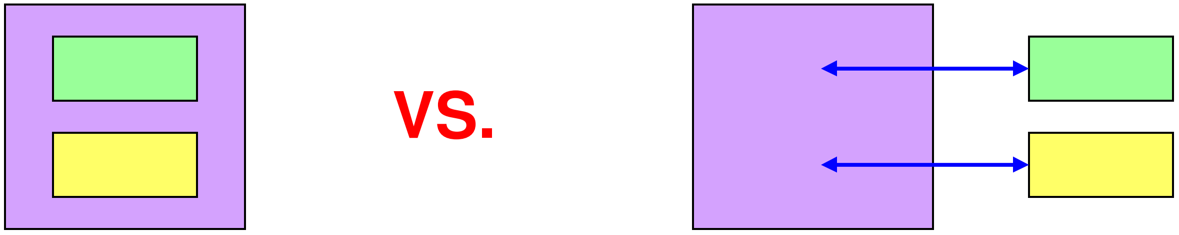
Practical Rules

Separation

Two common techniques for separation are hiding & hiding implementation

Practical Rules

Separation ⇨ Modularity


vs.

__Example__: If a function contains two loops, it is almost impossible to test one of them separated from the other

Practical Rules

Separation ⇨ Functions

The Biggest Misconception About Functions(◈)

(◈) Except interface functions

Practical Rules

Separation ⇨ Functions

~~The Purpose of Functions is~~

~~to Eliminate Code Duplication~~

Practical Rules

Separation ⇨ Functions

The Purpose of Functions
is to make the Code
Easier to Understand

- **By naming a piece of code** (saving comments)
- **By hiding its implementation** (high level code)
- **By making pre/post-conditions explicit**
  - **Also allowing** (partial) **isolation for testing**
- **By making the hosting code/function shorter**

Practical Rules

Separation

- Functions are for easy understanding
- Separate different concerns
- The evil of code-duplication (*)
- Encapsulation
- No getters.

(*) The lesson of Ariane5

Practical Rules

Controlled communication (Interface)

- Minimize number of users $(|\text{Width}| = |I|^*|U| = \Sigma_{(i\ \varepsilon\ I)}\mathbf{U}_{\mathbf{i}})$
- Minimal and complete (S. Meyers Eff. C++ 2nd)
- Make pre/post-conditions explicit
- Interface should preserve invariants
  - No setters

Practical Rules

Simplicity

- Short functions - single task
- Shallow nesting - low (cyclomatic) complexity
- Minimize function's side-effect
- Function side-effects via interface (visible)

Roadmap

- **“Why Software is So Bad?”** (The question, not the answer)
- **How to Review the Coding Process?**
- **Expectations from Quality Software**
- **Some Rules for Realization of Expectations**
- **Coding Standards Guides vs. those Principles**
- **Education, Management & in-betweens.**
- **Examples and Observations**
- **Q&A**

Coding Standards Guides

I argue that coding standards documents:

- **Miss most of the aforementioned coding rules**
- **Have stuff that should be put elsewhere.**

Indeed, they are more about low-level style
– e.g., uniformity and language don'ts + mini-rules.
Those are very important in practice,
but they do not replace the general rules.

Coding Standards Guides

MISRA-C (2004) **has:**

- "Minimal" scope for variables [in a function]
  - Whether objects are declared at the outermost or innermost block is largely a matter of style [?]
- (adv) Restrictions on pointer casting
- No goto/continue (break is restricted)
- Functions have a single point of exit at its end

Coding Standards Guides

JSF-AV C++ (2005) **has:**

- Class interface should be complete and minimal
- Const member functions are better
- (adv) usage of invariants
- No goto/continue (break is restricted)
- (adv) avoiding global variables
- Restricts down-casting (and casting in general)

Coding Standards Guides

A Possible Reservation:

The missing rules are expected to be well known by their knowledgeable engineers.

Indeed, it is possible. But then,

how come they have the following rules?

Coding Standards Guides

MISRA-C (2004) has:

- 12.3 (req) The `sizeof` operator shall not be used on expressions that contain side effects.
  - [They are worried programmers will expect evaluation]
- 16.8 (req) All exit paths from a function with non-void return type shall have an explicit return statement with an expression.
- 17.6 (req) The address of an object with automatic storage shall not be assigned to another object that may persist after the first object has ceased to exist.
  - [See below] (*)

Coding Standards Guides

JSF-AV C++ (2005) has:

- **#60** (as MISRA-C) **The `sizeof` operator …**
- **#81 The assignment operator shall handle self-assignment correctly**
- **#82 An assignment operator shall return a reference to `*this`**
- **#111 A function shall not return a pointer or reference to a non-static local object**
  - [See below] (*)

Coding Standards Guides

(*)The first day I’ve got the new, 3rd edition, of Stan Lippman’s ***C++ Primer***, I found three related errors: an automatic variable returned by reference.

Stan’s response to my e-mail was not just apologetic – he couldn’t understand how that error eluded both his review as well as the technical reviewers.

Do you think that a rule such as the above could have helped them?

– Such rules belong to learning

– Most are checked by lint-like tools

Coding standard is about **conscious activity** not about unintentional errors

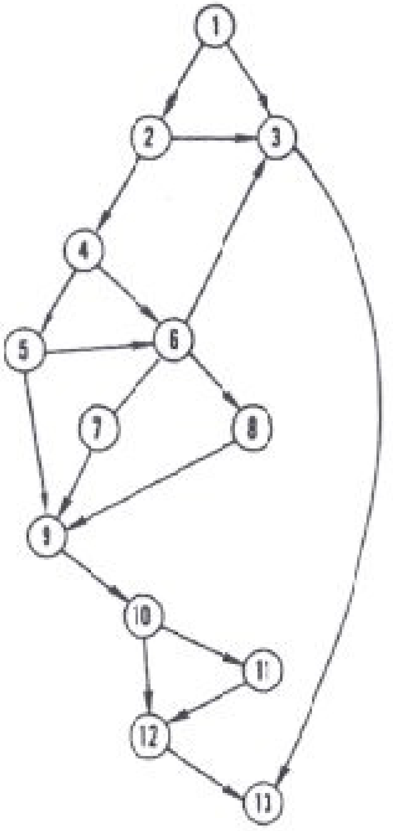
Coding Standards Guides

More from JSF-AV C++ (2005) has:

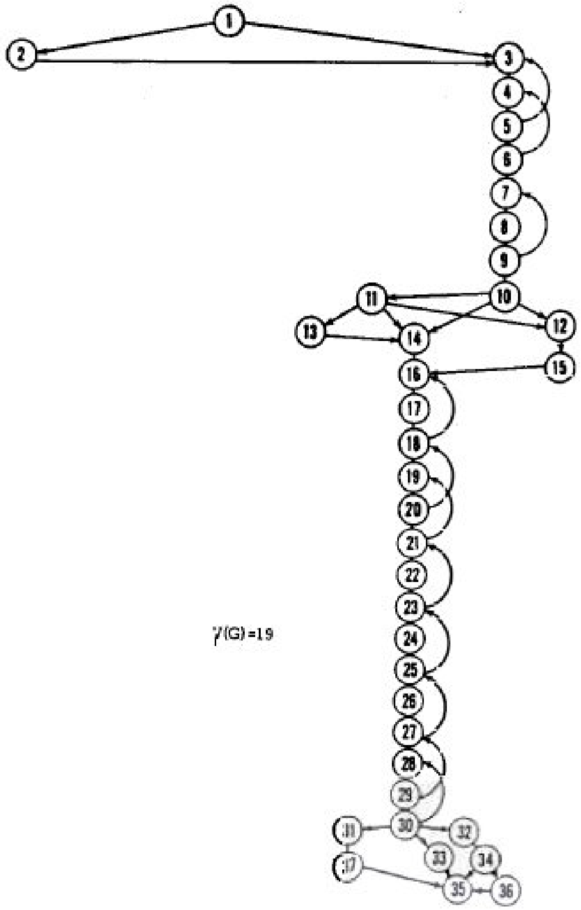
- #1 Any one function (or method) will contain no more than 200 logical source lines of code (L-SLOCs).
  - Rationale: Long functions tend to be complex and therefore difficult to comprehend and test.
- #3 All functions shall have a cyclomatic complexity number of 20 or less
  - Rationale: Limit function complexity.

Coding Standards Guides

Cyclomatic Complexity: McCabe, 1976



V(G) =8



V(G) =19

Roadmap

- “Why Software is So Bad?” (The question, not the answer)
- How to Review the Coding Process?
- Expectations from Quality Software
- Some Rules for Realization of Expectations
- Coding Style Guides vs. those Principles
- Education, Management & in-betweens.
- Examples and Observations
- Q&A

From Education to Management

Unless one objects to most what I have presented, the conclusions are mostly obvious.

Here is my take:

- First programming course – by SW-Eng literate staff.
  - Bad habits are hard to change (contrary to good habits)
  - Otherwise, programming is a tool – not a **profession**.
- Other programming courses – by SW-Eng. aware staff.
  - E.g., if HW is programming, TAs should be knowledgeable.
- Gradually introduce the rules (only half were presented)
  - Explain the rules' rational (they are **essence**, not style)

From Education to Management

Cooperate with the industry – when you're welcome
(I know of a case were even success didn't change attitude)

Here is my take:

- Industry is good at fighting bugs – not at eliminating them
  - There are many great **bug** tracking systems
  - There is no single **non-bug** tracking system
- Industry spends M-$ on testing
  - But much less on educating their engineers
  - "Your code must be maintainable by the least experienced team member"
- Industry spends M-$ on process
  - But much less on contents [?]

Resisting Changes

ACCU Meeting, speaker: Dan Saks 10/25/11 (abstract)

Most programmers fancy themselves to be rational and objective, more so than the general population. Recent research suggests this self image might have a basis in fact.
Nonetheless, C and C++ programmers still cling to programming styles and practices which are unsupported by evidence and sometimes even contradicted by it.

Comedian Stephen Colbert has popularized the word "truthiness" to describe the human trait of knowing something "from the gut" without regard to actual facts. This talk takes a lighthearted look at C and C++ programmers' truthiness in the hope of inspiring more truthfulness.

Roadmap

- **“Why Software is So Bad?”** (The question, not the answer)
- **How to Review the Coding Process?**
- **Expectations from Quality Software**
- **Some Rules for Realization of Expectations**
- **Coding Style Guides vs. those Principles**
- **Education, Management & in-betweens.**
- **Examples and Observations**
- **Q&A**

Examples and Observations

A Simple Industrial Example

```
PlumberStatus
Tap::open_tap(const string& tap_name)
{
    LockSys<Mutex> LL(tap_lock_);
    TapMap::const_iterator it =
                         taps_.find(tap_name);
    if (it == taps_.end()) {
        return PLUMB_TAP_NOT_FOUND;
    }
    it->second->operate(true);
    return PLUMB_OK;
}
```

Examples and Observations

What’s the difference?

```
PlumberStatus
Tap::close_tap(const string& tap_name)
{
    LockSys<Mutex> LL(tap_lock_);
    TapMap::const_iterator it =
                         taps_.find(tap_name);
    if (it == taps_.end()) {
        return PLUMB_TAP_NOT_FOUND;
    }
    it->second->operate(false);
    return PLUMB_OK;
}
```

Examples and Observations

What’s The Problem?

- Is it **code duplication**?
  - Let’s see:
    - **After extracting out the common parts we get**

```
PlumberStatus
Tap::open_tap(const string& tap_name)
{
    LockSys<Mutex> LL(tap_lock_);
    if (!tap_found(tap_name)) {
        return PLUMB_TAP_NOT_FOUND;
    }
    it->second->operate(true);
    return PLUMB_OK;
}
```

Code Duplication is just the Symptom

The real problem:
Each one of them has two tasks

- Delegation (of a function call)
- Wrapping: Transforming boolean value => name

Single Task Implementation - Delegation

```
PlumberStatus
Tap::operate_tap(const string& name, bool open)
{   LockSys<Mutex> LL(tap_lock_);
    TapMap::const_iterator it =
                          taps_.find(tap_name);
    if (it == taps_.end()) {
        return PLUMB_TAP_NOT_FOUND;
    }
    it->second->operate(open);
    return PLUMB_OK;
}
```

Single Task Implementation – Wrappers

With appropriate design, these may be made

non-member non-friend functions

```
inline PlumberStatus
Tap::open_tap(const string& tap_name)
{ return operate_tap(tap_name, true);}
```

Both functions are inlined,
so they consume no executable space

```
inline PlumberStatus
Tap::close_tap(const string& tap_name)
{ return operate_tap(tap_name, false);}
```

The Original has a Third Problem

It enforces awkward usage

```
if (activation_required) {
    open_tap(name);
} else {
    close_tap(name);
}
```

Instead of

```
operate_tap(name, activation_required);
```

Resisting Changes (industrial example)

```
    if (A) {
        value = true;
    } else if (B) {
        value = false;
    } else if (C) {
        value = false;
    } else {
        value = true;
    }

////// An Alternative ////////

    value = A || (!B && !C);
```

Some developers claim the alternative will not be understood by new hires.

The above is idiomatic in C and C++. Therefore, we can choose between

1. Gradually elevating our new hires' knowledge to a professional level.
2. Adjusting our professional code to meet our new hires' knowledge.

Resisting Changes (cont.)

```
    if (A) {
        value = true;
    } else if (B) {
        valve = false;
    } else if (C) {
        value = false;
    } else {
        value = true;
    }

//////  An Alternative  ////////

    value = A || (!B && !C);
```

Some developers claim the alternative will not be understood by new college graduates.

The latter version is the way to **guarantee** assignment is to a single variable

Q & A

Sources

[1] Charles C. Mann "Why software is so bad?"
MIT Technology Review, 2002
http://www.technologyreview.com/featuredstory
/401594/why-software-is-so-bad/

[2] Robert N. Charette "Why Software Fails"
IEEE Spectrum 2005
http://spectrum.ieee.org/computing/software
/why-software-fails/

[3] Mark Minasi, "The Software Conspiracy",
Mcgraw-Hill, 1999

Sources (cont.)

[4] Beth Layman "An Interview w. Jerry Weinberg" Software Quality Professional, v.3 no.4, 2001 ASQ

http://www.stickyminds.com/interview/software-engineering-state-practice-interview-jerry-weinberg

[5] J. Pontin, "The problem with Programming: Interview w. B. Stroustrup", MIT Technology Review, 2006

http://www.technologyreview.com/news/406923/the-problem-with-programming

[6] Y. Kimchi, "Coding with Reason", in "97 Things Every Programmer Should Know", ed. K.Henney, O'Reily 2010