SWEN 6301 Software Construction

Lecture 1: Course Overview

Ahmed Tamrawi

Important Equations for the Class

Erwin Schrödinger (1887-1961)

Schrödinger Equation:

$$i\hbar\frac{\partial}{\partial t}\Psi(x,t) = -\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2}\Psi(x,t) + V(x)\Psi(x,t)$$

Schrödinger equation does for a quantum-mechanical particle what Newton's Second Law does for a classical particle. The Solution to Schrödinger equation to determine how a particle evolves in time, just as we use Newton's Second Law to solve for future position and momentum of a classical particle.

Further Reading:

1. https://simple.wikipedia.org/wiki/Schr%C3%B6dinger_equation
2. https://www.quora.com/What-is-the-Schr%C3%B6dinger-wave-equation-and-what-are-its-applications

Werner Heisenberg (1901-1976)

Heisenberg Uncertainty Principal:

$$\Delta x \Delta p \geq \frac{h}{4\pi} = \frac{\hbar}{2}$$

In the world of very small particles, one cannot measure any property of a particle without interacting with it in some way. This introduces an unavoidable uncertainty into the result. Thus, One can never measure all the properties exactly.

The more accurately you know the position (the smaller Δx is), the less accurately you know the momentum (the larger Δp is); and vice versa

Further Reading: https://en.wikipedia.org/wiki/Uncertainty_principle

B.Eng. Computer Engineering
(Class of 2007)

IOWA STATE UNIVERSITY

M.Sc. Computer Engineering
(Class of 2011)

IOWA STATE UNIVERSITY

Ph.D. Computer Engineering
(Class of 2016)

Secure Programming

Static Program Analysis

Data & Pattern Mining

Software Analysis & Security

Bug finding and Malware detection

Build System Analysis

Abstractions and Symbolic Evaluations

Quantum Physics

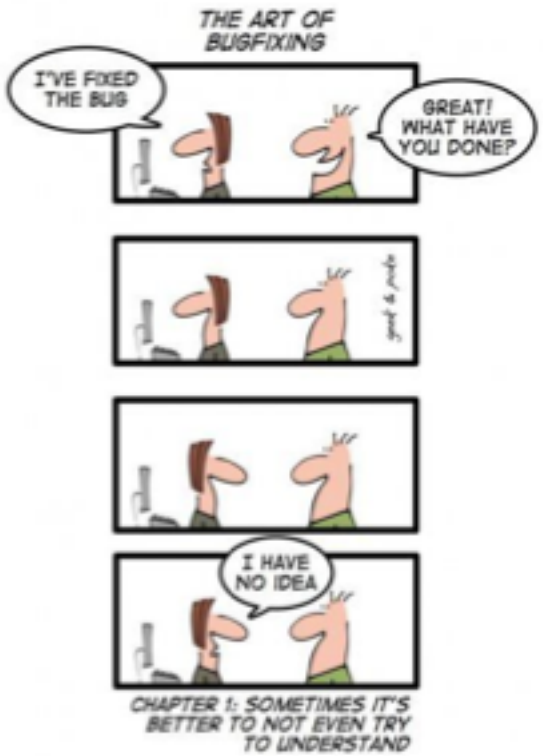
Biology

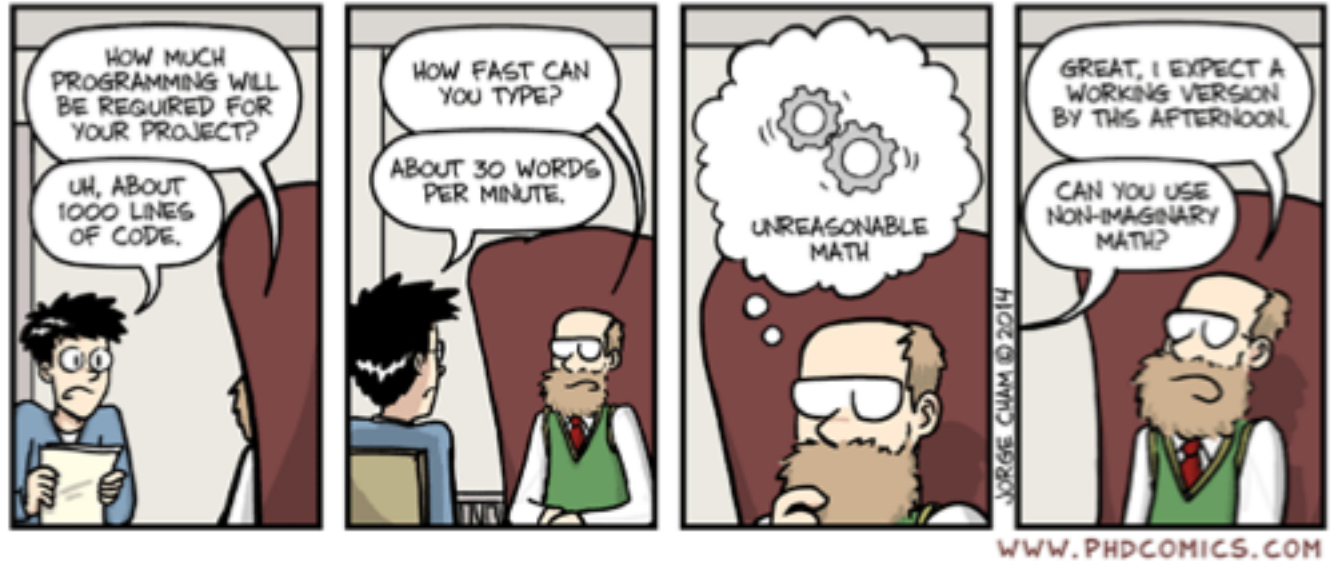
Astronomy


i'm not on facebook®
WhatsApp

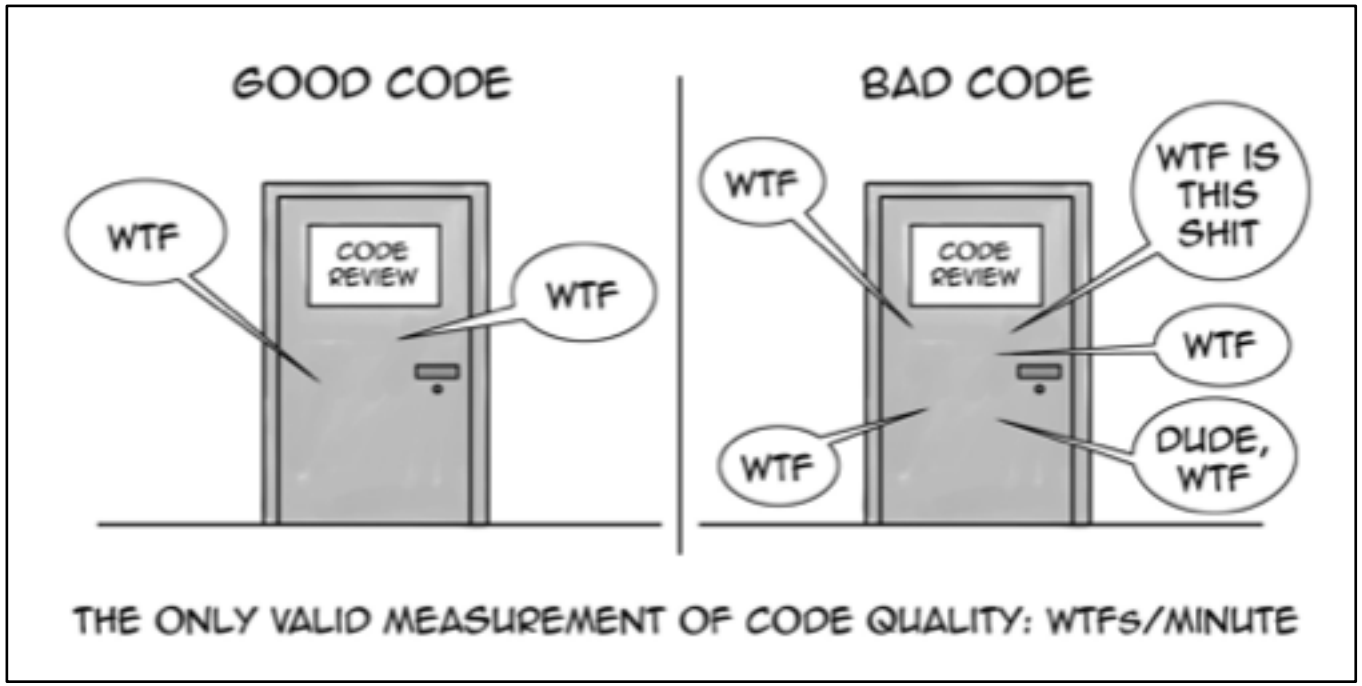
Your turn!

- *Name*
- *Undergraduate major and/or current work.*
- *Something about you*
  - *Food you like.*
  - *Programming languages you used.*
  - *Open source projects you contributed to.*
- *What do you think of this course?*
- *What are your goals after graduation?*

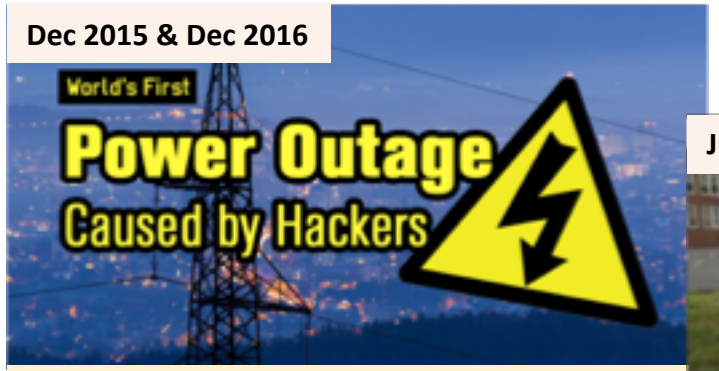

THE ART OF BUGFIXING
I'VE FIXED THE BUG
GREAT! WHAT HAVE YOU DONE?
I HAVE NO IDEA
CHAPTER 1: SOMETIMES IT'S BETTER TO NOT EVEN TRY TO UNDERSTAND


HOW MUCH PROGRAMMING WILL BE REQUIRED FOR YOUR PROJECT?
UH, ABOUT 1000 LINES OF CODE.
HOW FAST CAN YOU TYPE?
ABOUT 30 WORDS PER MINUTE.
UNREASONABLE MATH
GREAT, I EXPECT A WORKING VERSION BY THIS AFTERNOON.
CAN YOU USE NON-IMAGINARY MATH?
JORGE CHAM © 2014
WWW.PHDCOMICS.COM


HAVE YOU FIXED THE BUG?
IN PRINCIPLE YES
YES
YOU MEAN: NO


GOOD CODE
WTF
CODE REVIEW
WTF
BAD CODE
WTF
CODE REVIEW
WTF IS THIS SHIT
WTF
WTF
DUDE, WTF
THE ONLY VALID MEASUREMENT OF CODE QUALITY: WTFS/MINUTE

Dec 2015 & Dec 2016



Ukraine power grid attacks

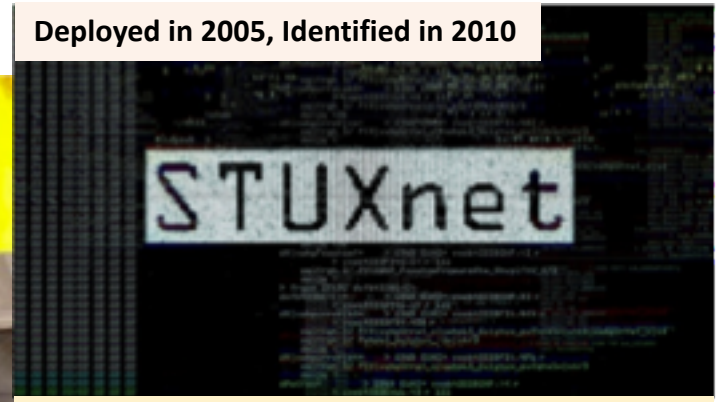
July 21, 2015

Jeep remotely hijacked

November 29, 2011

HP printers remotely set on fire

Deployed in 2005, Identified in 2010

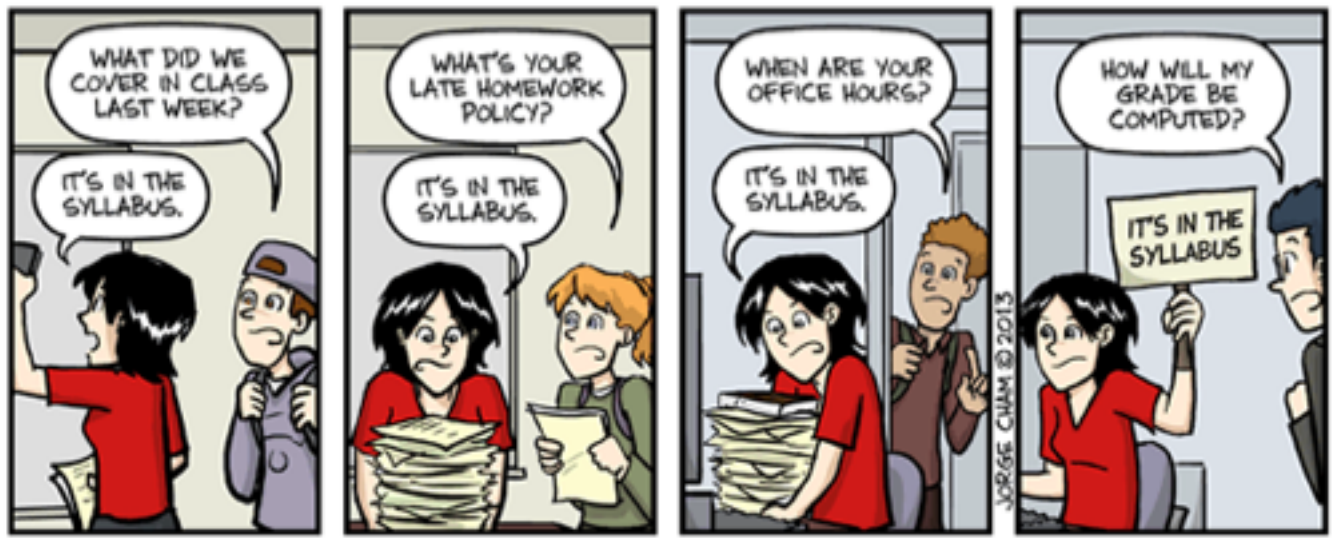


STUXnet Worm

{ Complex Software }

Although software development practice has *advanced rapidly* in recent years, common practice hasn't

Many programs are still **buggy**, **late**, and **over budget**, and many **fail** to satisfy the needs of their users



IT'S IN THE SYLLABUS

This message brought to you by every instructor that ever lived.

WWW.PHDCOMICS.COM

SWEN6301: Software Construction

Fall 2019

Instructor:	Ahmed Tamrawi	**Time:**	S 14:00 – 16:50
Email:	ahmedtamrawi@gmail.com	**Place:**	Masri 402
Course Site:	TBD		

Office Hours & Questions: After class, by appointment, or simply send me an email. (Please include "SWEN6301" in the subject line of your emails to me)

Course Overview: Software is everywhere and despite the rapid advances in software development practices and integrated development environments (IDEs), we still produce buggy, late, over budget, and many fail to satisfy the needs of their users. The Software Construction course is about how to perform software development for delivering robust and resilient solutions which minimizes the occurrence of the aforementioned issues. This course aims at providing the theories and techniques of effective and maintainable software development by using modern technologies and concepts, focusing on software construction.

The *primary goal* of this course is to help you to create higher-quality software that is robust, flexible, extensible, scalable, and maintainable. We focus on doing that by understanding what are common software development practices. The other main goal of this course is to provide experiences and knowledge that will help you develop as professional programmers and computing system designers. This includes: (1) experience working in a team, both as a leader and contributor, and (2) experience using tools commonly used by productive developers.

If you do not feel my goals for the course align well with your personal goals, but you need to take this course anyway to satisfy a degree requirement, you should meet with me to figure out a way to make this course useful for satisfying your personal goals.

Expected Background & Prerequisites: Students entering this course are expected to be comfortable reading, designing, and writing *Java* programs that involve code distributed over many modules. You should be comfortable learning how to use new programming language features and APIs by reading their documentation (or source code when no documentation is available), and not be surprised when solving programming assignments requires you to seek documentation beyond what was provided in class. Students should be able to understand and implement different data structures and sorting algorithms.

Textbook: We will closely follow the textbooks from:

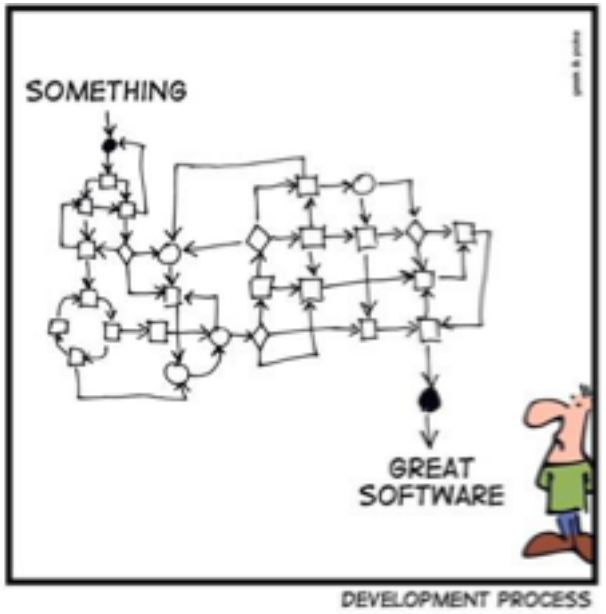
- Steve McConnell's *"Code Complete: A Practical Handbook of Software Construction,"* 2nd Edition, (ISBN 978-0735619678).
- Robert C. Martin's *'Clean Code: A Handbook of Agile Software Craftsmanship',"* 1st Edition (ISBN 978-0132350884).

In addition, we will have several readings from many other resources including:

- Joshua Bloch's *'Effective Java',"* 3rd Edition (ISBN 978-0134685991).
- Edward Crookshanks's *'Practical Software Development Techniques: Tools and Techniques for Building Enterprise Software',"* 1st Edition (ISBN 978-1484207291).
- Frank Tsui, Orlando Karam, and Barbara Bernal 's *'Essentials of Software Engineering',"* 4th Edition (ISBN 978-1284106008).
- Ian Sommerville's *'Software Engineering',"* 10th Edition (ISBN 978-0133943030).

Honor: As a graduate student, you are trusted to be honorable. We will take advantage of this trust to provide a better learning environment for everyone. In particular, students in SWEN 6301 are expected to follow these rules:

Goal of the Class



Improve your ability to **create** higher-quality software that is *robust*, *extensible*, *scalable*, *maintainable*, and *secure* by **understanding** what are **common software construction practices**

My Goals for Lectures?

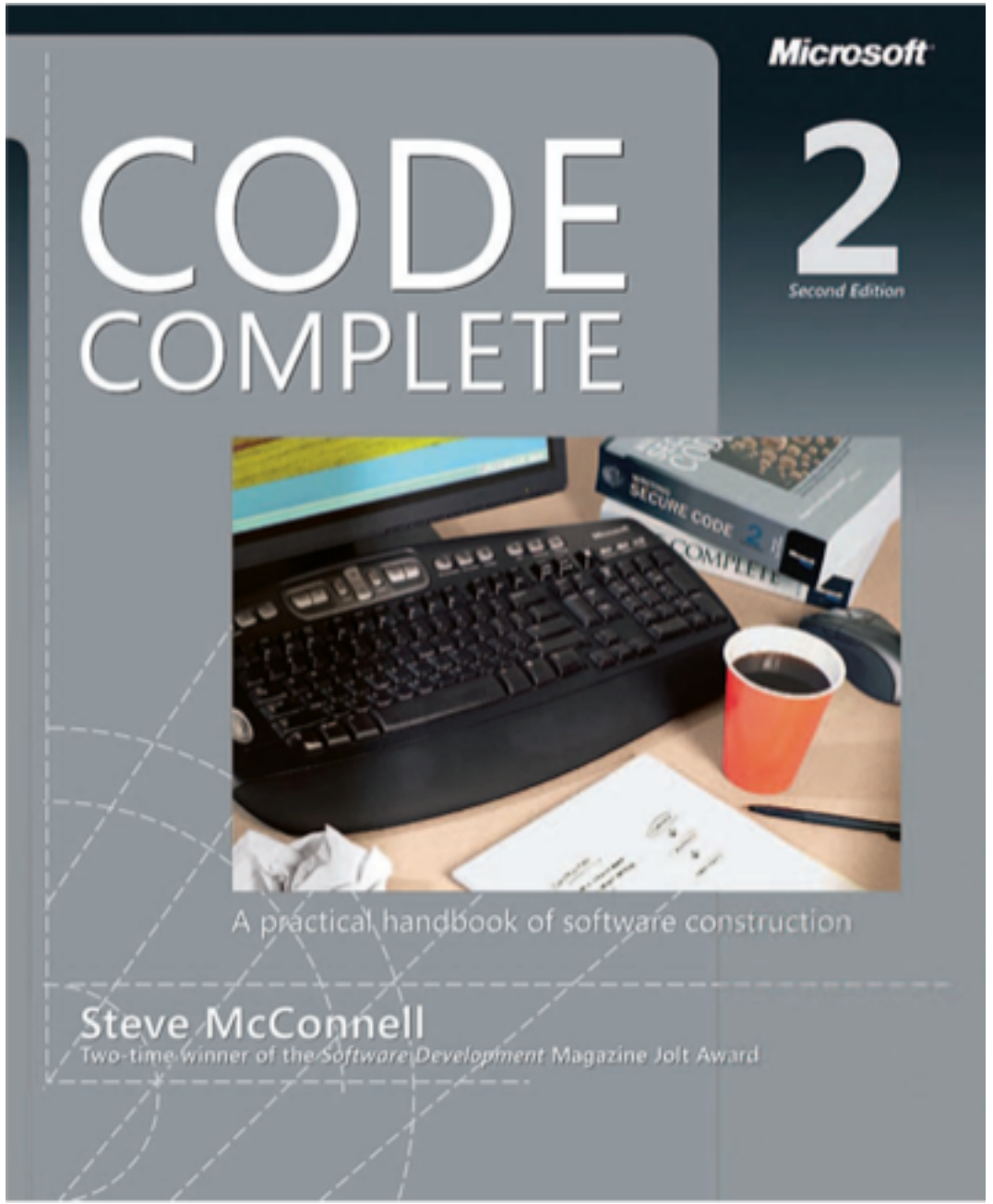
Convey some complex technical ideas	Lectures are *horrible* medium for learning complex ideas, many resource are available online
Teach you what you need to know to do the assignments, exams and the project	The point of assignments, exams and project is to teach you things I want you to learn in the class
Avoid being fired	Avoid being fired
Keep most of you awake for 170 minutes	You probably should be getting more sleep
Get you to laugh at dumb jokes	Gabriel Iglesias is funnier (*check him out*)

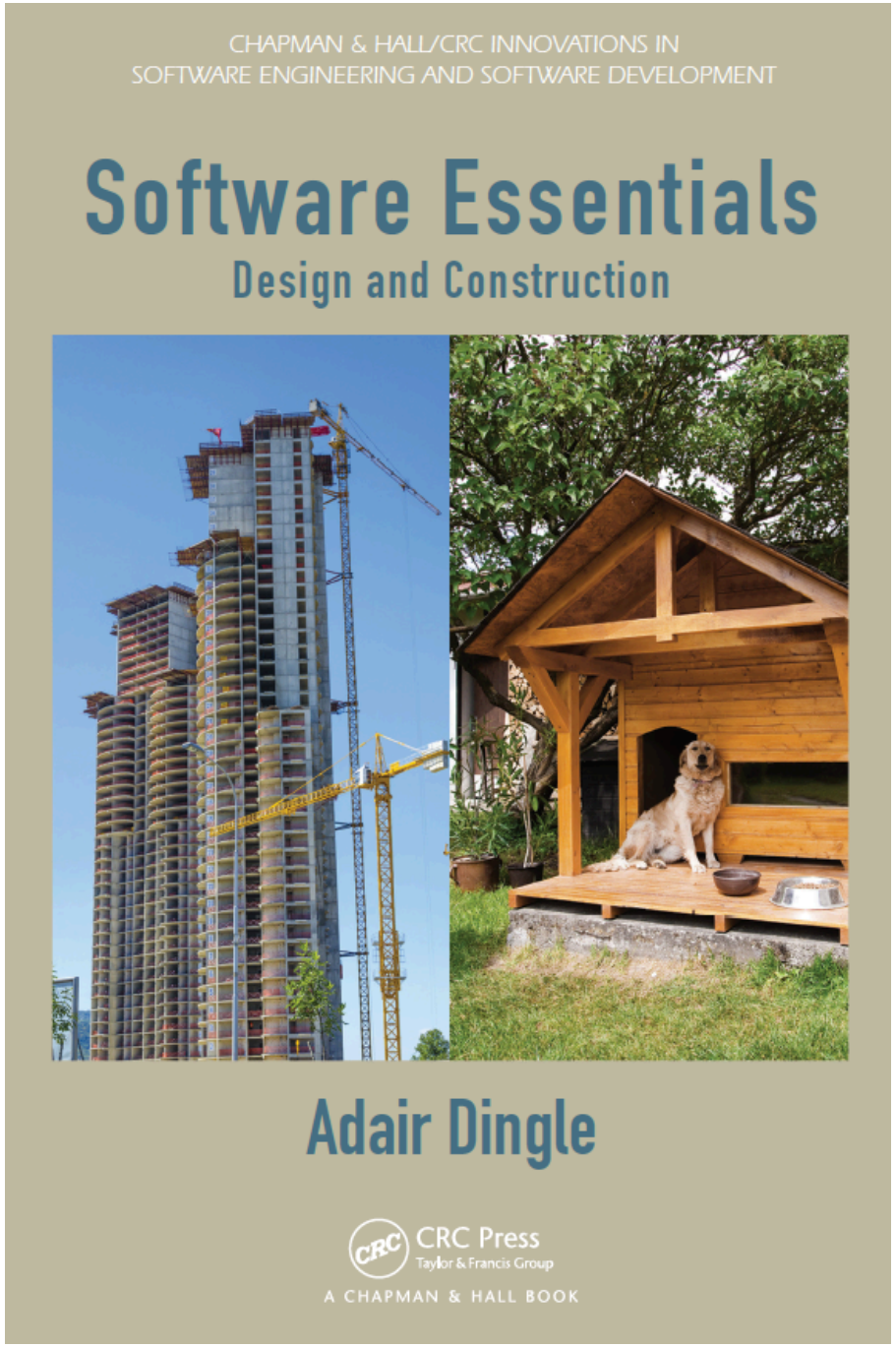
My **Real Goal** for Lectures

Provide **context** and **meaning** for the things you have or will later **learn on your own**

As the figure indicates, construction is mostly coding and debugging but also involves detailed design, construction planning, unit testing, integration, integration testing, and other activities. If this were a book about all aspects of software development, it would feature nicely balanced discussions of all activities in the development process. Because this is a handbook of construction techniques, however, it places a lopsided emphasis on construction and only touches on related topics. If this book were a dog, it would nuzzle up to construction, wag its tail at design and testing, and bark at the other development activities.

Construction is also sometimes known as "coding" or "programming." "Coding" isn't really the best word because it implies the mechanical translation of a preexisting design into a computer language; construction is not at all mechanical and involves substantial creativity and judgment. Throughout the book, I use "programming" interchangeably with "construction."





As the premier professional organization for software engineering, the IEEE Computer Society strives to develop standards for and advance knowledge of software development. SoftWare Engineering Body of Knowledge (SWEBOK) is an endeavor to summarize current best practices of and tool usage within software development. Software construction is one area of focus and covers the concepts reviewed in this chapter. The essential details of software construction, as outlined by SWEBOK, are summarized in Table 2.5. For more details, see http://www.computer.org/portal/web/swebok.

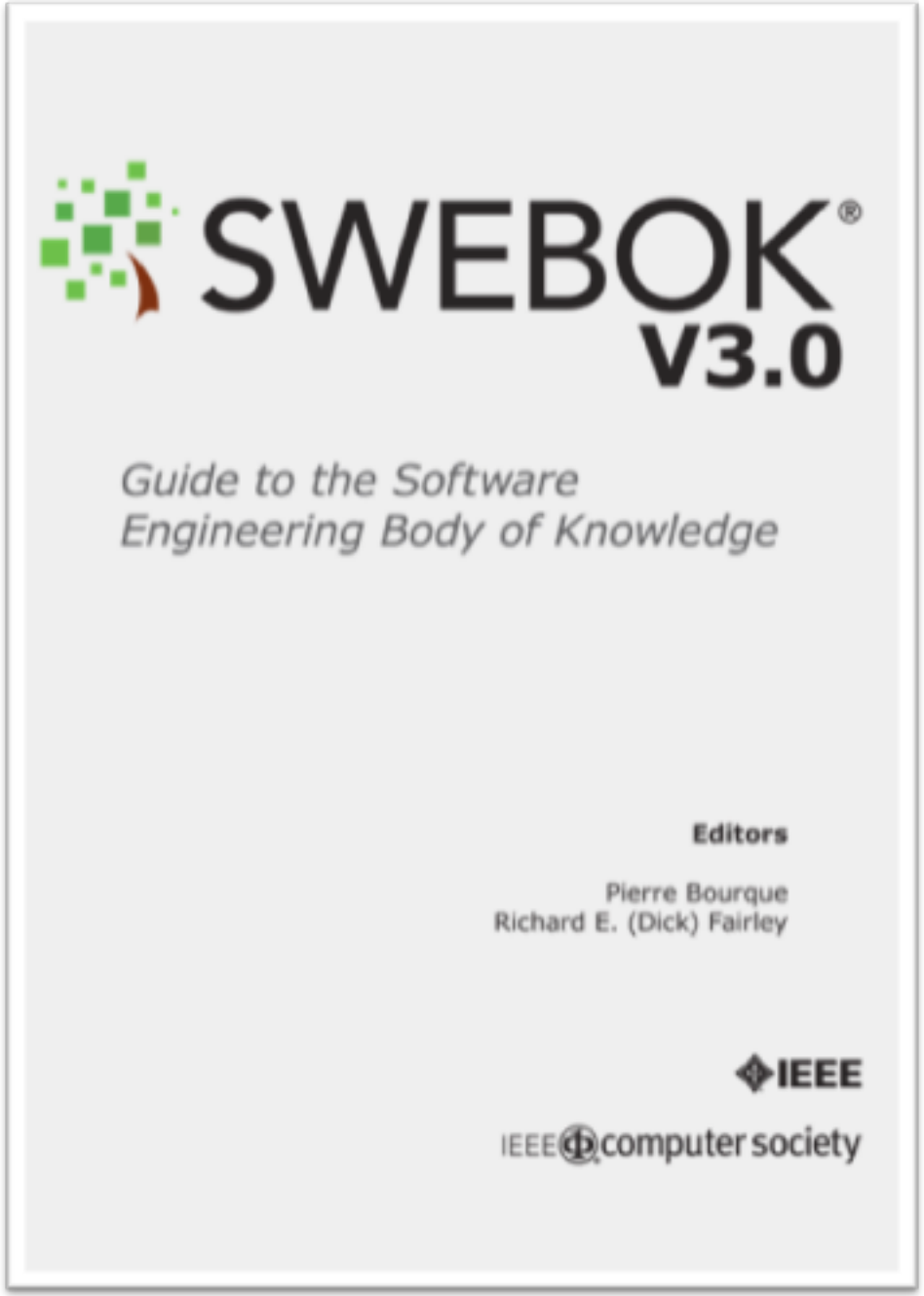
Software construction suggests code development but spans verification of functionality via unit and integration testing as well as debugging. Although the four fundamentals listed in Table 2.5 apply also to the modeling and design phases of software development, we examine them here in the context of coding.

SWEBOK's fundamental to minimize code complexity seeks to produce clear and readable code. Constraining **software complexity** eases the tasks of modeling, documentation, and testing. To limit the software complexity of a code base, good programming practices should be followed, including functional decomposition, encapsulation, appropriate use of control structures, **self-documenting code** using mnemonic names and constants, conscious design and implementation of error processing

The term software construction refers to the detailed creation of working software through a combination of coding, verification, unit testing, integration testing, and debugging.

The Software Construction knowledge area (KA) is linked to all the other KAs, but it is most strongly linked to Software Design and Software Testing because the software construction process involves significant software design and testing. The process uses the design output and provides an input to testing ("design" and "testing" in this case referring to the activities, not the KAs). Boundaries between design, construction, and testing (if any) will vary depending on the software life cycle processes that are used in a project.

Although some detailed design may be performed prior to construction, much design work is performed during the construction activity. Thus, the Software Construction KA is closely linked to the Software Design KA.



Do we like any of these definitions?

No universally accepted definition

I know that I like Mansaf!

What does this function do?

```
1  float[] foo(float[] array, float val1) {
2      float[] array_2 = null;
3      float val2 = 0;
4      for(int i = 1; i < array.length; i++) {
5          array_2 = new float[i];
6          for(int j = 0; j < i; j++) {
7              array_2[j] = array[j];
8          }
9
10         float avg = average(array_2);
11         if(avg <= val1) {
12             continue;
13         } else {
14             break;
15         }
16     }
17     return array_2;
18 }
```

What is wrong?

```
1  float[] foo(float[] array, float val1) {
2      float[] array_2 = null;
3      float val2 = 0;
4      for(int i = 1; i < array.length; i++) {
5          array_2 = new float[i];
6          for(int j = 0; j < i; j++) {
7              array_2[j] = array[j];
8          }
9
10         float avg = average(array_2);
11         if(avg <= val1) {
12             continue;
13         } else {
14             break;
15         }
16     }
17     return array_2;
18 }
```

What is wrong?

```
1  float[] foo(float[] array, float val1) {
2      float[] array_2 = null;
3      float val2 = 0;
4      for(int i = 1; i < array.length; i++) {
5          array_2 = new float[i];
6          for(int j = 0; j < i; j++) {
7              array_2[j] = array[j];
8          }
9
10         float avg = average(array_2);
11         if(avg <= val1) {
12             continue;
13         } else {
14             break;
15         }
16     }
17     return array_2;
18 }
```

- Not a descriptive function name
- No comments about what this function does
- Variable names are not descriptive either
- Calculates average at each iteration from scratch instead of updating it, therefore, being not scalable for very large arrays
- One branch of the `if` is used for loop continuation
- Unused variable `val2`
- Do not check for `null` array return

A Better Version

```
1  List<Float> findSubListExceedingTargetAverage(List<Float> values, Float targetAverage) {
2      Float sum = 0.0;
3      List<Float> result = new ArrayList<Float>();
4      for(Float value: values) {
5          result.add(value);
6          sum += value;
7          Float average = sum / (Float) result.size();
8          if(average > targetAverage) {
9              break;
10         }
11     }
12     return result;
13 }
```


انا مش فاهم حاجة خالص

SWEN 6301 Software Construction Definition

Software construction is the process of **creating** and **evolving** software source code that results on *extensible*, *maintainable*, *robust*, and *secure* software

Main Ideas in SWEN 6301

Creating Code

How do you *create* **code** that is robust, extensible, maintainable, and secure?

Evolving Code

How do you *evolve* **code** in an efficient way with minimum complexity to keep the overall code robust, extensible, maintainable, and secure?


فلـــــة شـمعـة منـــــورة


Software Construction
a very complicated task

How complicated is **Software Construction** for a Tesla car?

Course Overview

Laying the Foundation

Variables & Statements

Concurrency

Software Craftsmanship

Creating High-Quality Code

Code Improvements

System Considerations


LET US GET SERIOUS
AND CLEAN SOME CODE

We are close to the end of code

Soon all code will be generated instead of written

Programmers simply won’t be needed because business people will generate programs from specifications

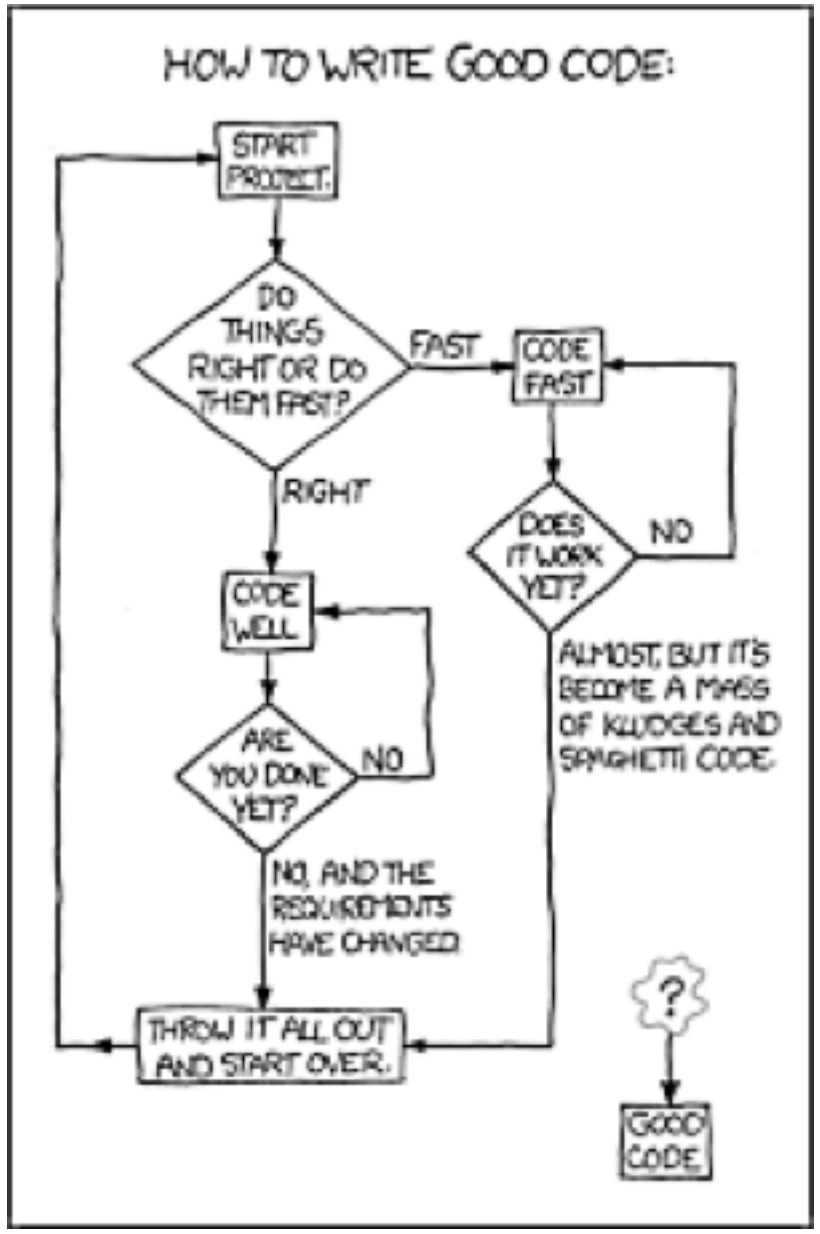
We will create machines that can do what we want rather than what we say. These machines can translate vaguely specified needs into perfectly executing programs that precisely meet those needs.

There Will Be Code

We will never be rid of code

Code represents the **details of the requirements**. At some level those details cannot be ignored or abstracted; they have to be **specified**.

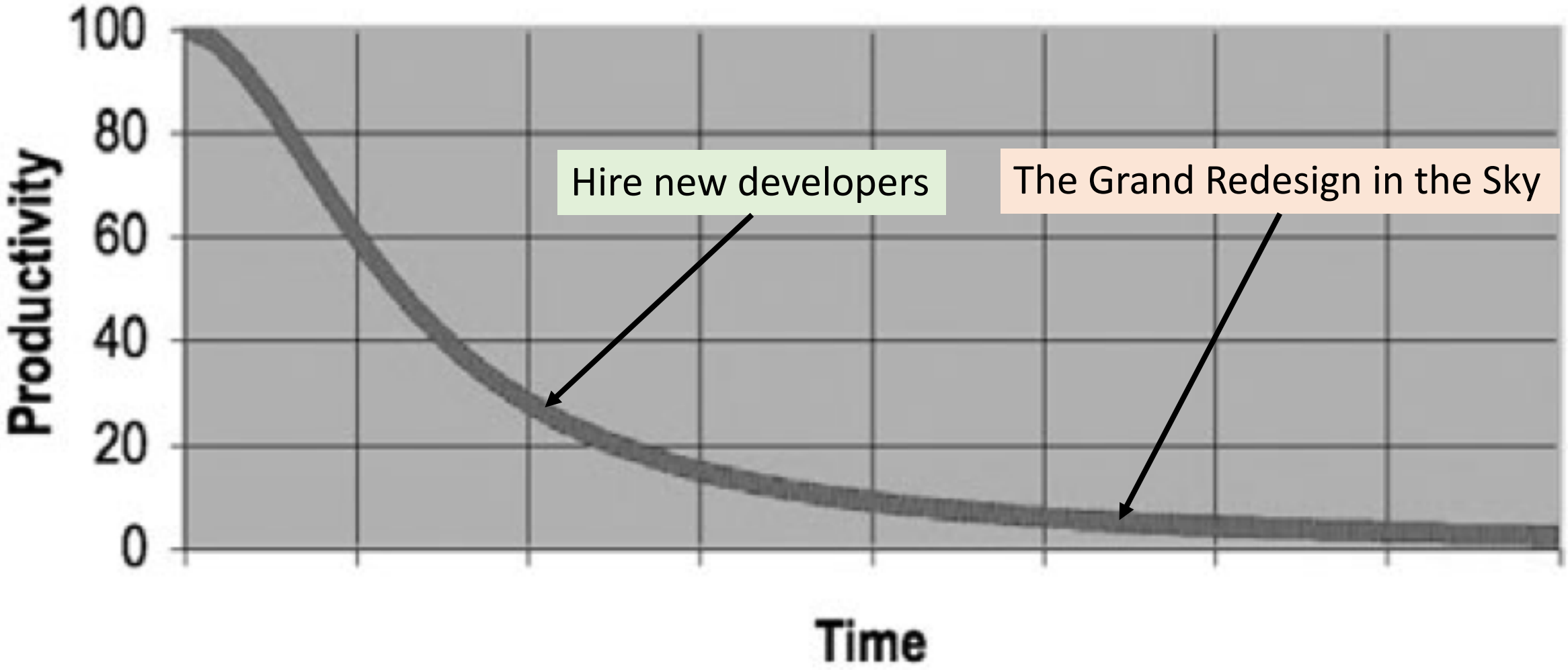
Specifying requirements *in such detail that a machine can execute them is* ***programming****. Such a specification is* ***code****.*



Good Code vs Bad Code

It was the bad code that brought the company down

The Total Cost of Owning a Mess



Why does good code rot so quickly into bad code?

The **schedules** were too tight to do things right

The **requirements** changed in ways that thwart the original design

Stupid managers and intolerant customers and useless marketing types and telephone sanitizers

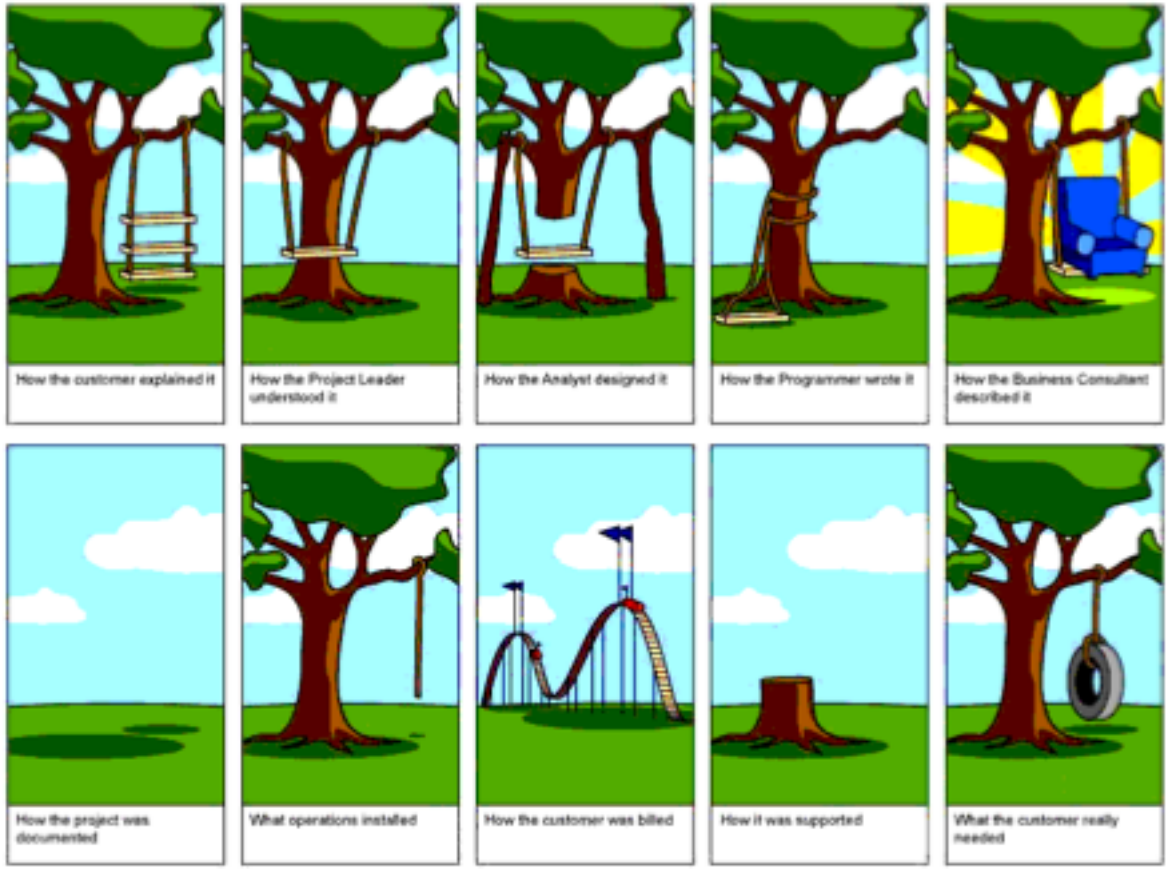
But the fault is not in our stars, but in ourselves. *We are **unprofessional**.*

*The project managers look to us to help work out the **schedule***

*The users look to us to validate the way the **requirements** will fit into the system.*

*The **managers and marketers** look to us for the information they need to make promises and commitments*

It's your job to defend the code with equal passion.



It is **unprofessional** for programmers to *bend* to the will of managers who <u>don't understand</u> the risks of making messes.

The **only** way to make the **deadline**—*the only way to go fast*—is to keep the **code** as **clean** as possible at all times.

The Art of Clean Code?

A programmer who writes clean code is an artist who can take a blank screen through a series of transformations until it is an elegantly coded system.

What Is Clean Code?

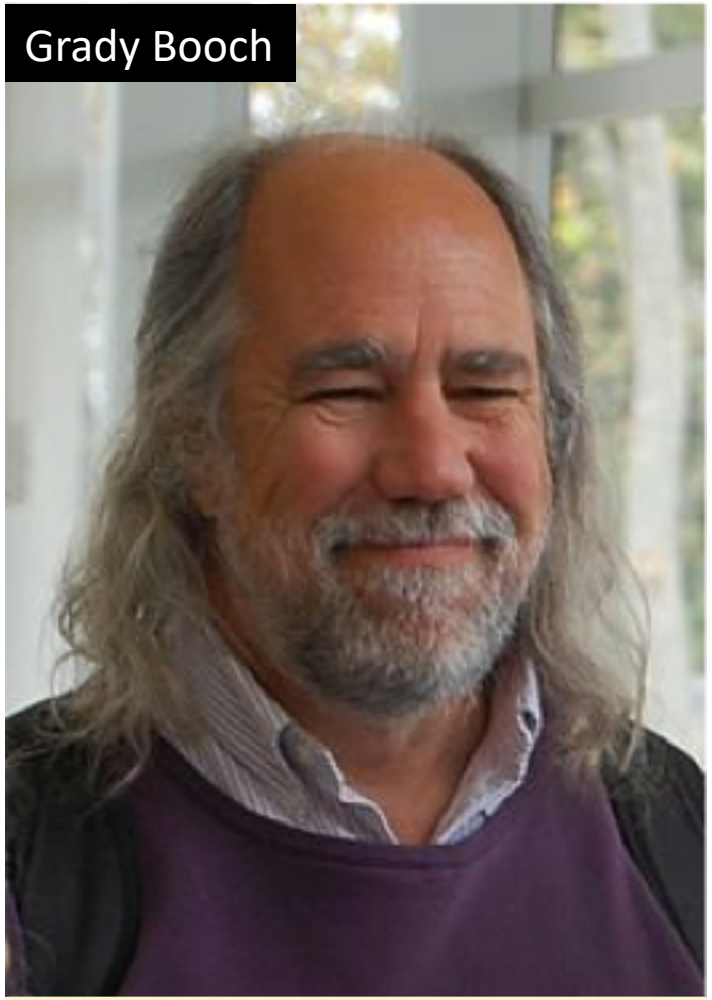
*I like my code to be **elegant** and **efficient**. The **logic should be straightforward** to make it hard for bugs to hide, the **dependencies minimal** to ease **maintenance**, **error handling complete** according to an articulated strategy, and **performance close to optimal** so as not to tempt people to make the code messy with unprincipled optimizations. Clean code does one thing well.*

Inventor of C++ and author of The C++ Programming Language

What Is Clean Code?

*Clean code is **simple** and **direct**. Clean code reads like **well-written prose**. Clean code never obscures the designer's intent but rather is full of crisp abstractions and straightforward lines of control.*

Grady Booch



Author of Object Oriented Analysis and Design with Applications

What Is Clean Code?

*Clean code can be read, and **enhanced by a developer** other than its original author. It has **unit and acceptance tests**. It has **meaningful names**. It provides one way rather than many ways for doing one thing. It has **minimal dependencies**, which are explicitly defined, and provides a **clear and minimal API**. Code should be **literate** since depending on the language, not all necessary information can be expressed clearly in code alone.*

Founder of OTI, godfather of the Eclipse strategy

What Is Clean Code?

I could list all of the qualities that I notice in clean code, but there is one overarching quality that leads to all of them. ***Clean code always looks like it was written by someone who cares****. There is* ***nothing obvious that you can do to make it better****. All of those things were thought about by the code's author, and if you try to imagine improvements, you're led back to where you are, sitting in appreciation of the code someone left for you—code left by someone who cares deeply about the craft.*

Author of Working Effectively with Legacy Code

What Is Clean Code?

In recent years I begin, and nearly end, with Beck's rules of simple code. In priority order, simple code:

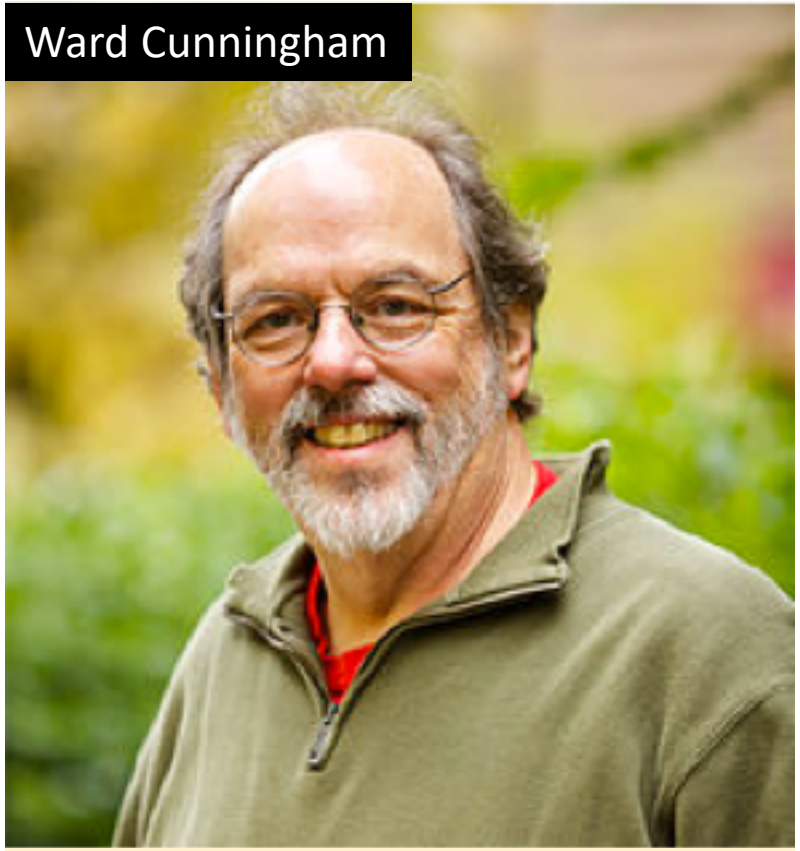
- *Runs all the tests;*
- *Contains no duplication;*
- *Expresses all the design ideas that are in the system;*
- *Minimizes the number of entities such as classes, methods, functions, and the like.*

Author of Extreme Programming Installed and Extreme Programming Adventures in C#

What Is Clean Code?

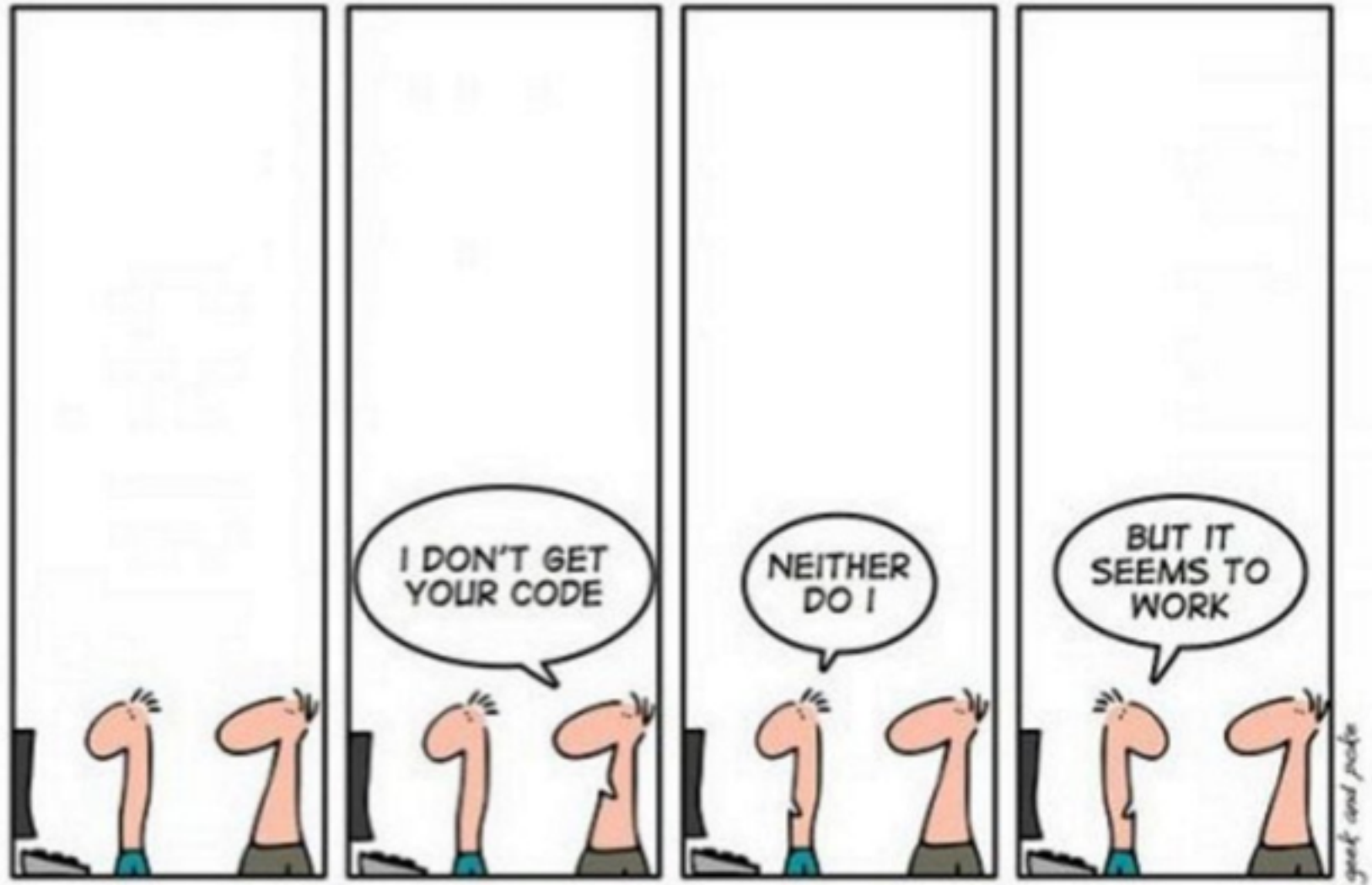
You know you are working on clean code when ***each routine you read turns out to be pretty much what you expected****. You can call it beautiful code when the code also makes* ***it look like the language was made for the problem****.*

Ward Cunningham



Inventor of Wiki, inventor of Fit, coinventor of eXtreme Programming. Motive force behind Design Patterns. Smalltalk and OO thought leader. The godfather of all those who care about code.


THE BOY SCOUT RULE
ALWAYS
LEAVE
CODE
CLEANER
THAN YOU
FOUND IT!



THE ART OF PROGRAMING

つづく