SWEN 6301 Software Construction

Module 1: Introduction

Ahmed Tamrawi

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)

amazon

Secure Programming

Static Program Analysis

Data & Pattern Mining

Software Analysis & Security

Bug finding and Malware detection

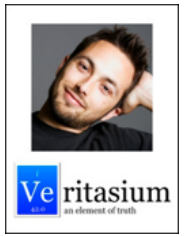
Build System Analysis

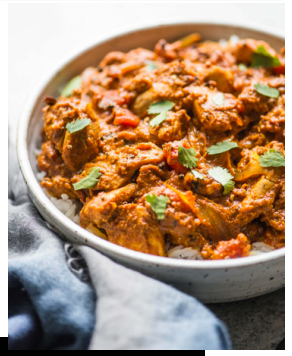
Abstractions and Symbolic Evaluations

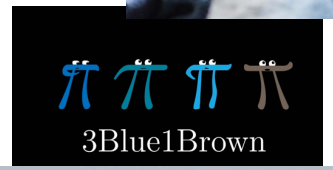
Quantum Physics

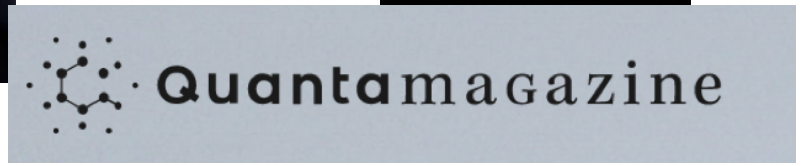
Biology

Astronomy





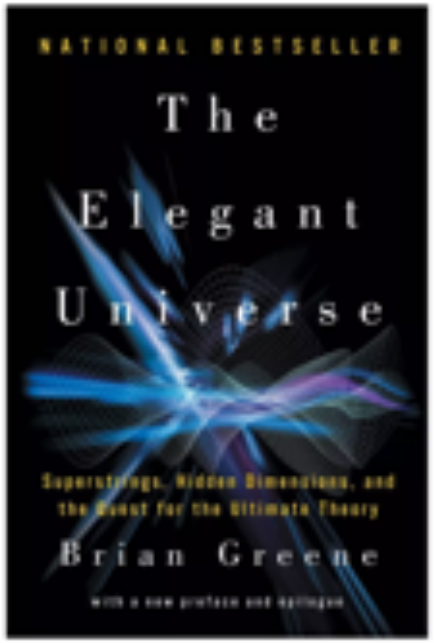




Your turn!

- *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?*

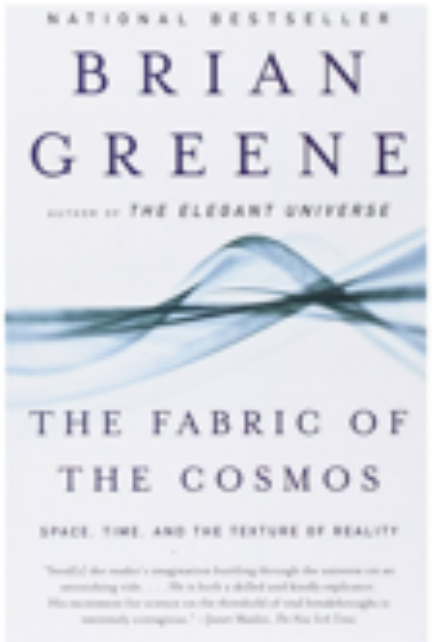
Brian Green's Books



The Elegant Universe

A rare blend of scientific insight and writing as elegant as the theories it explains

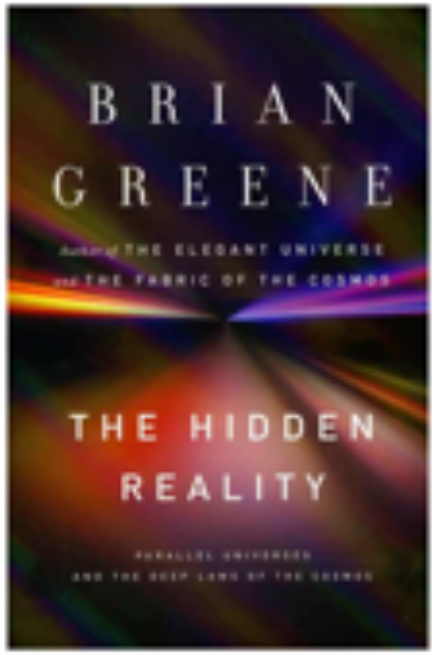
Learn more ›



The Fabric of the Cosmos

An irresistible and revelatory journey to the new layers of reality that modern physics has discovered

Learn More ›



The Hidden Reality

A remarkable adventure through a world more vast and strange than anything we could have imagined

Learn More ›

Icarus at the Edge of Time

A futuristic reimagining of the classic Greek myth, for younger audiences

Learn More ›

Does the method work?

```
1 public static boolean isOdd(int i) {
2         return i % 2 == 1;
3 }
```

Does the method work?

Unfortunately, it doesn't; it returns the wrong answer one quarter of the time.

```
1 public static boolean isOdd(int i) {
2         return i % 2 == 1;
3 }
```

How to fix it?

Does the method work?

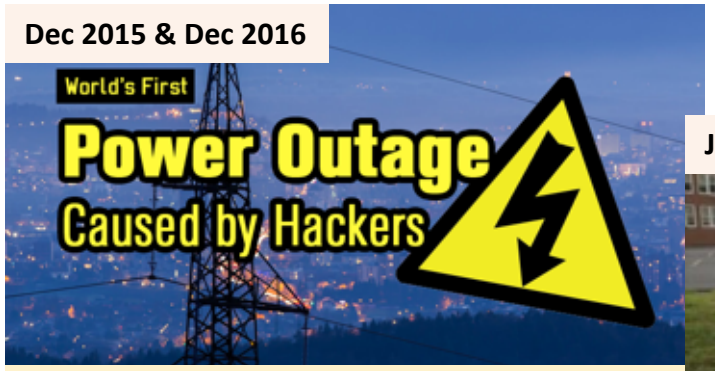
```
1 public static boolean isOdd(int i) {
2         return i % 2 != 0;
3 }
```

Can we do better?

Does the method work?

```
1 public static boolean isOdd(int i) {
2         return (i & 1) != 0;
3 }
```

Dec 2015 & Dec 2016



Ukraine power grid attacks

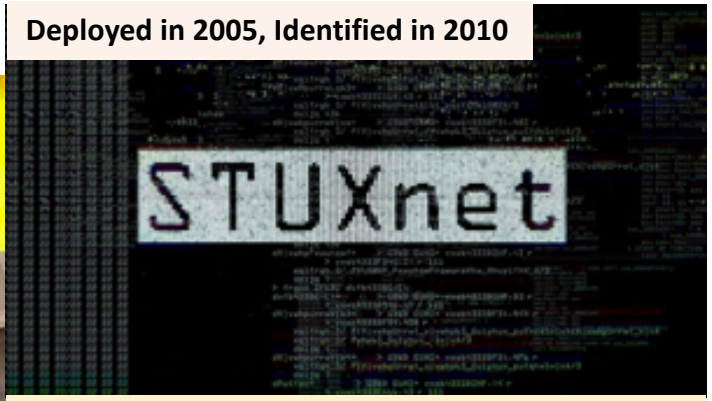
July 21, 2015

Jeep remotely hijacked

November 29, 2011

HP printers remotely set on fire

Deployed in 2005, Identified in 2010

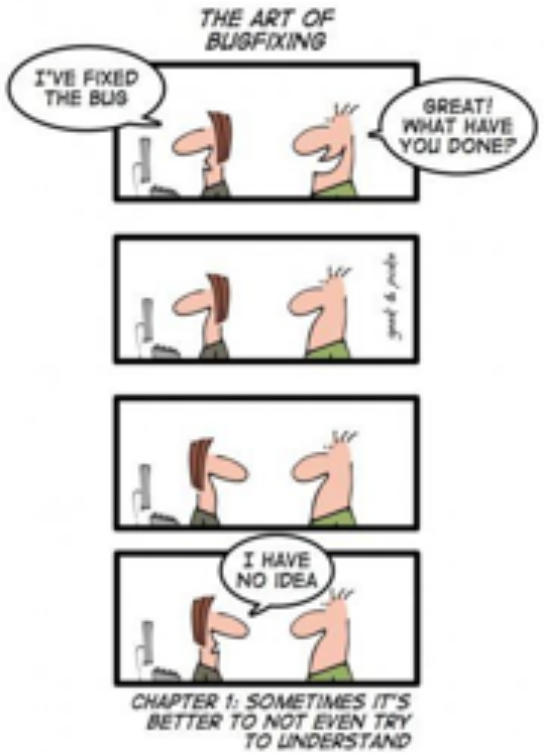


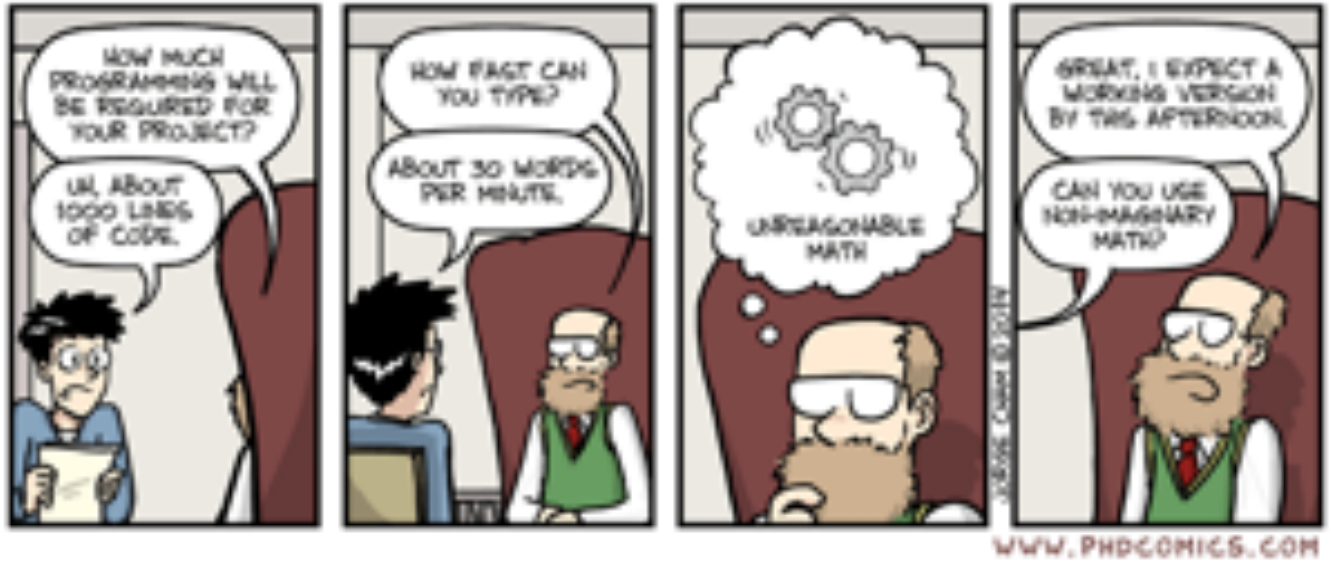
STUXnet Worm

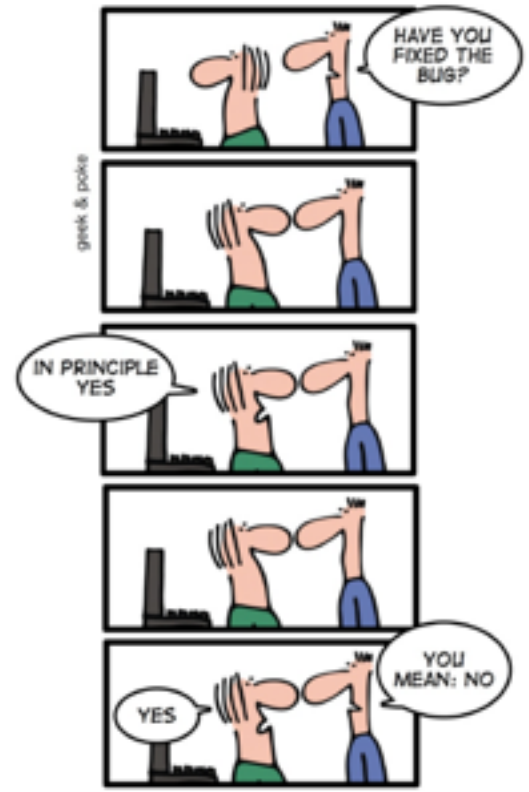
{ Complex Software }

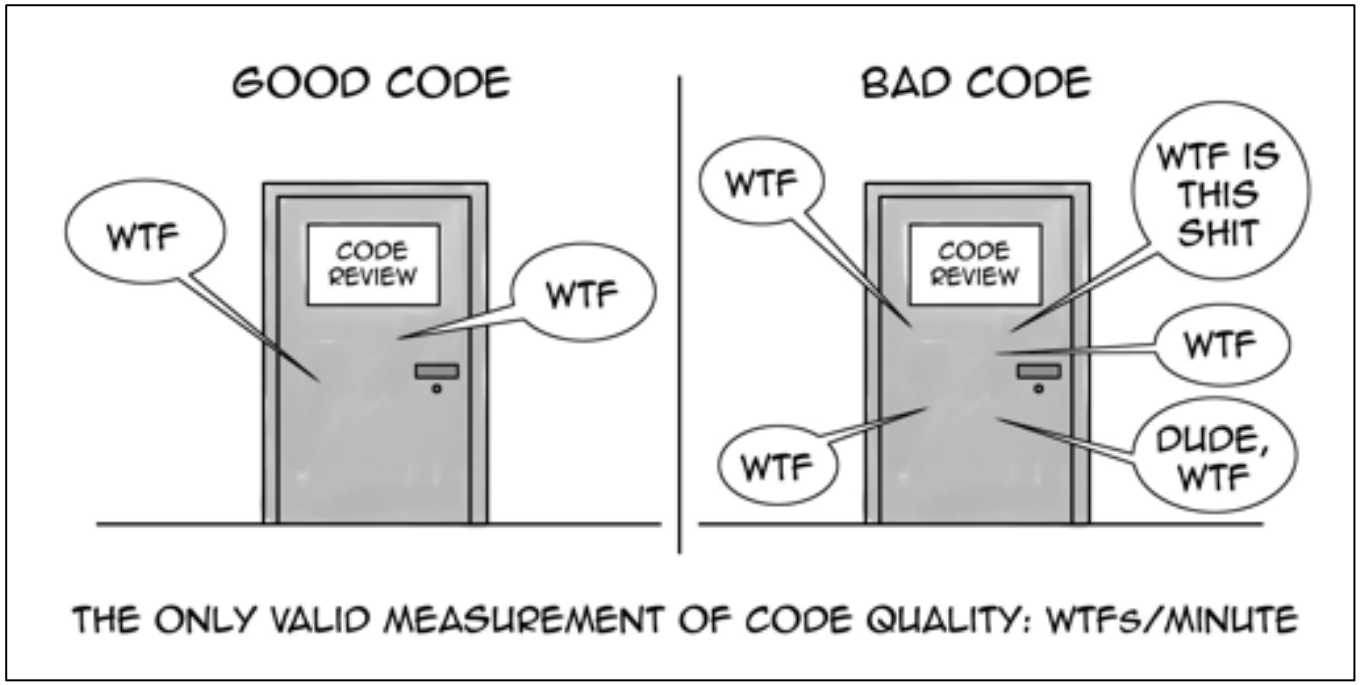
Although software development practice has *advanced rapidly* in recent years, common practice hasn't

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


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?
WWW.PHDCOMICS.COM


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


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

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:

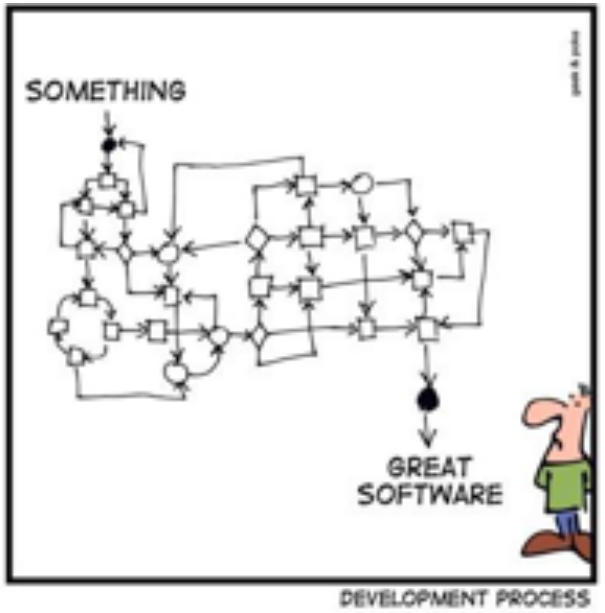
- Steve McConnell's *"Code Complete: A Practical Handbook of Software Construction,"* 2nd Edition, (ISBN 978-0735619678).
- Robert C. Martin's *'Clean Code: A Handbook of Agile Software Craftsmanship',"* 1st Edition (ISBN 978-0132350884).

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*)

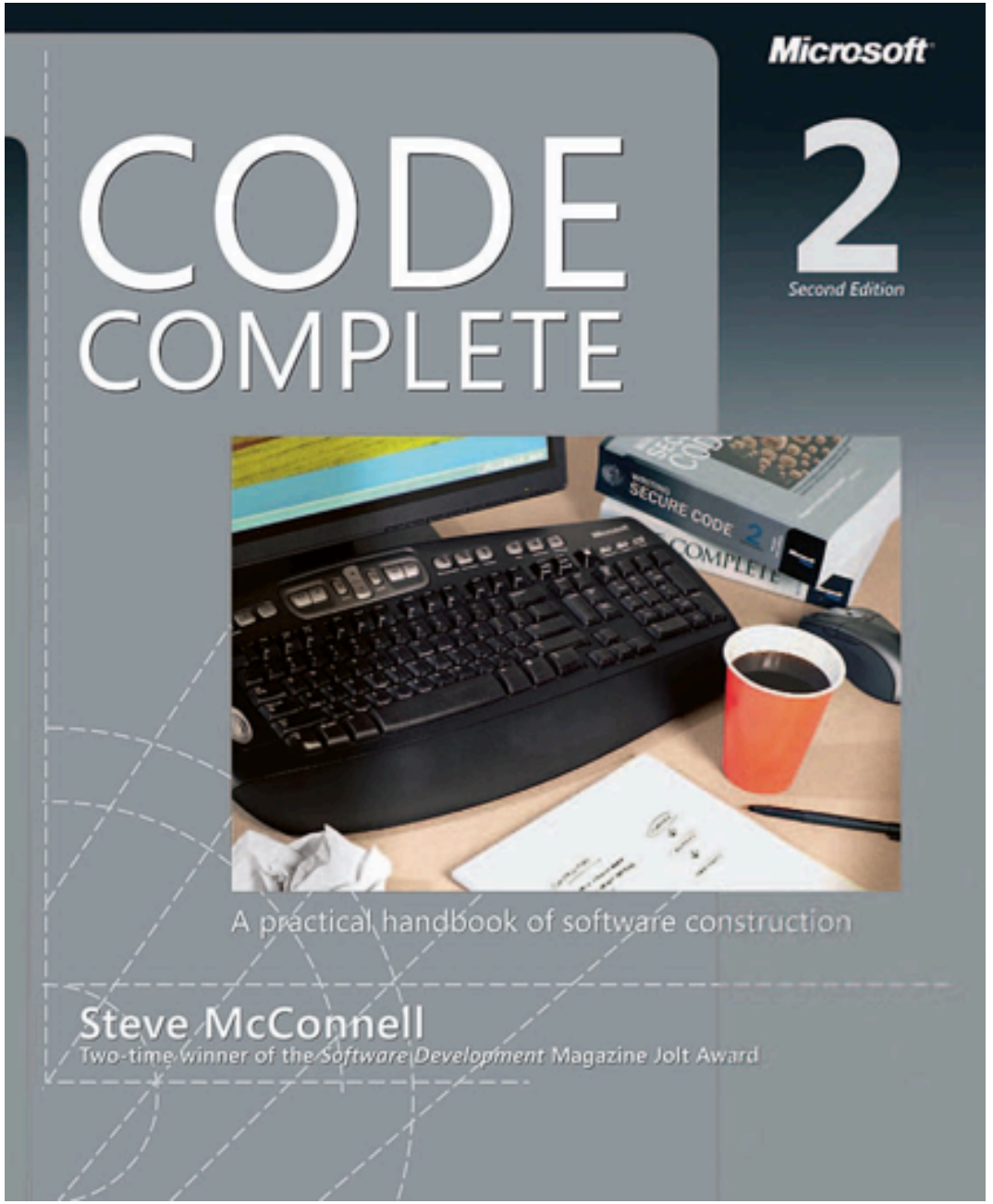
My **Real Goal** for Lectures

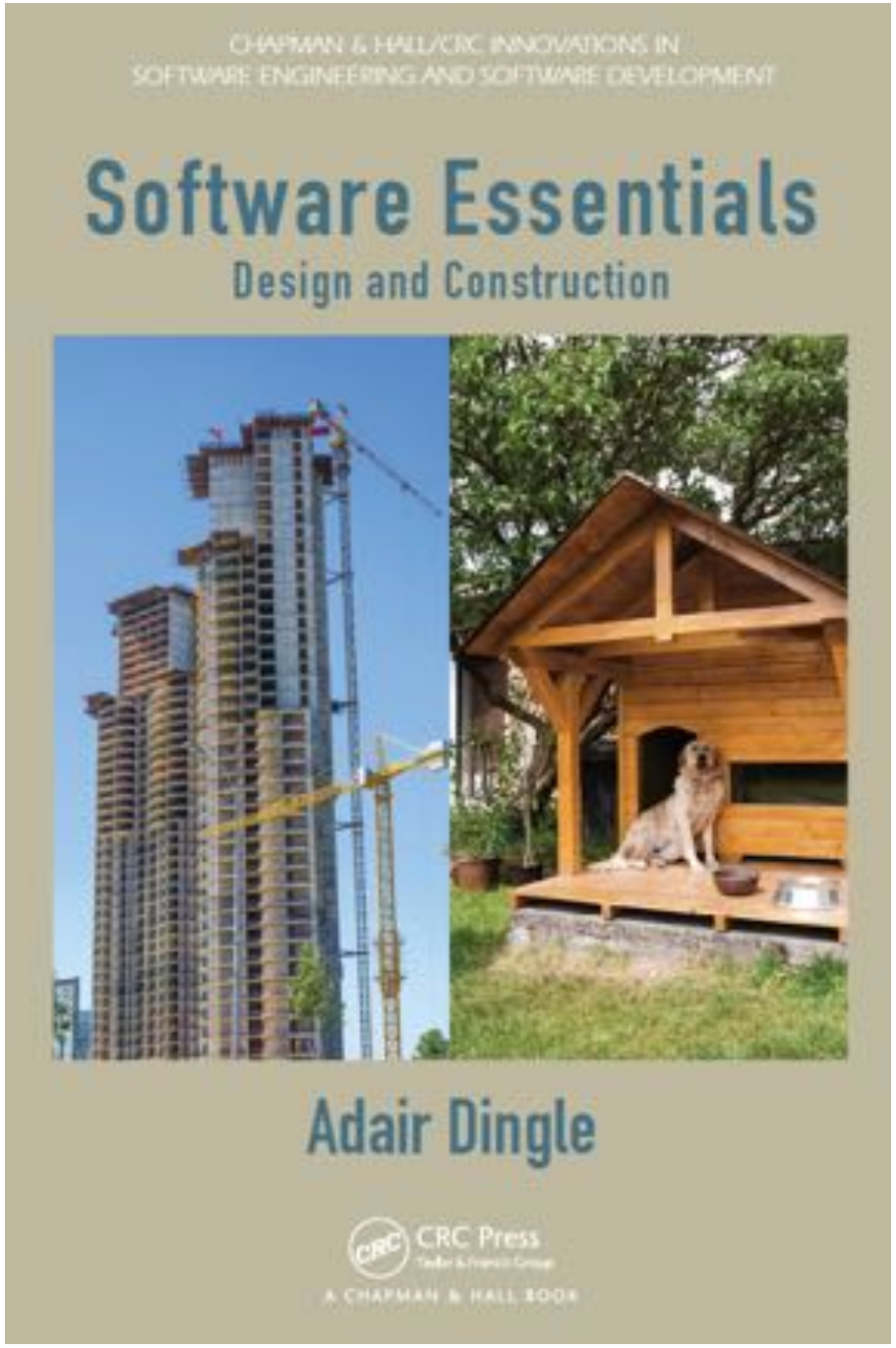
Provide **context** and **meaning** for the things you have or will later **learn on your own**

What is **Software Construction**?

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."





What is **Software 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.

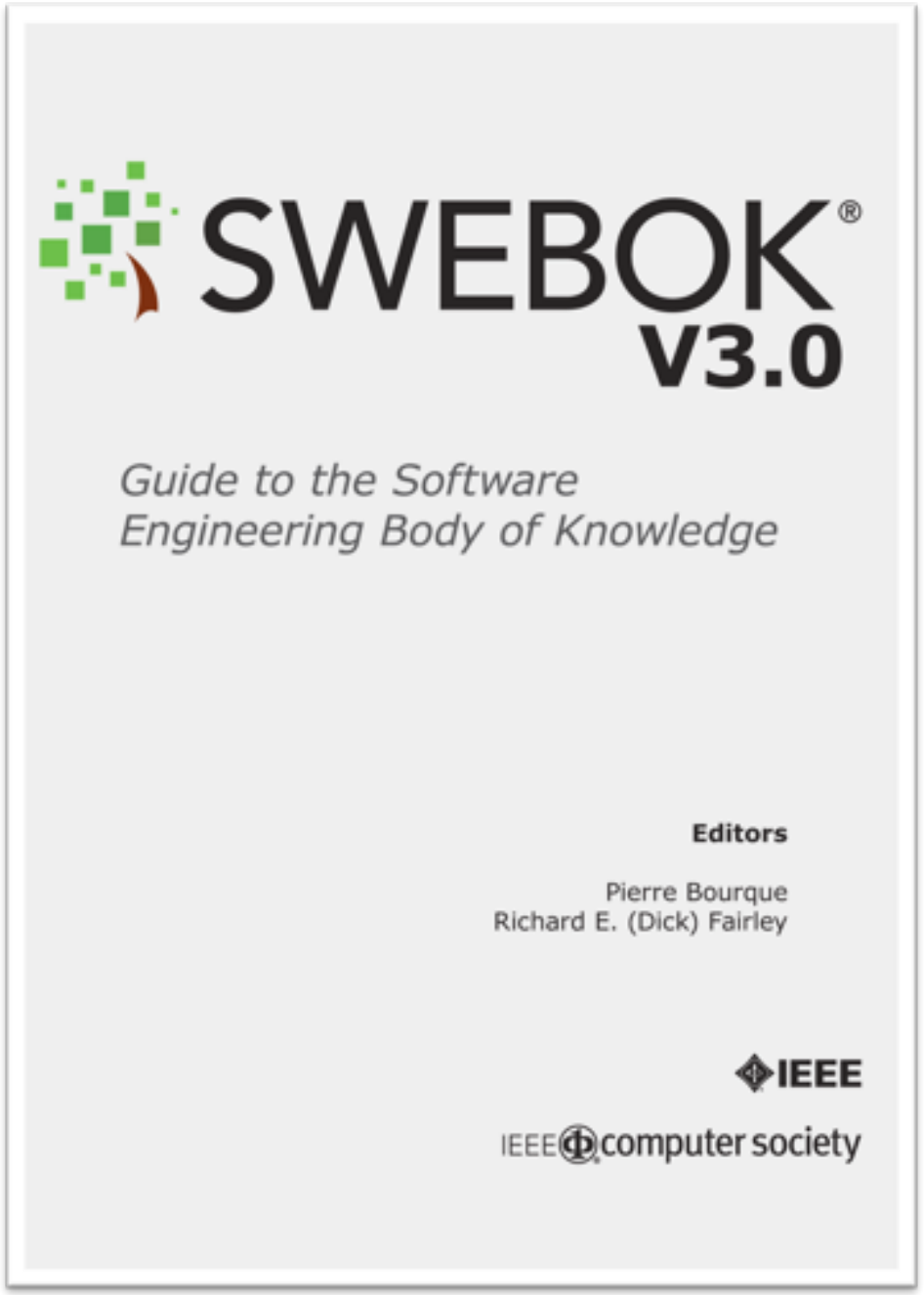
SWEBOK's fundamental to minimize code complexity seeks to produce clear and readable code. Constraining **software complexity** eases the tasks of modeling, documentation, and testing. To limit the software complexity of a code base, good programming practices should be followed, including functional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

What is **Software Construction**?

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


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

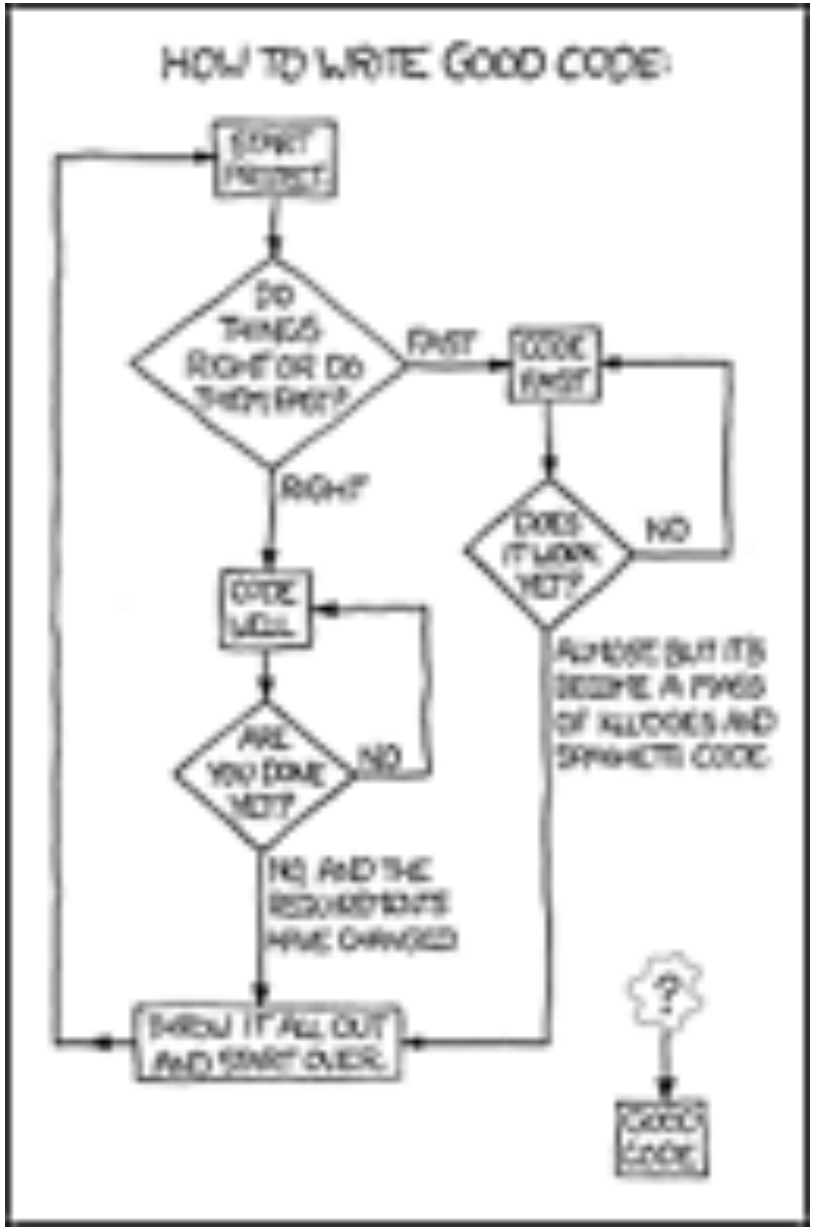
We will create machines that can do what we want rather than what we say. These machines can translate vaguely specified needs into perfectly executing programs that precisely meet those needs.

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**.

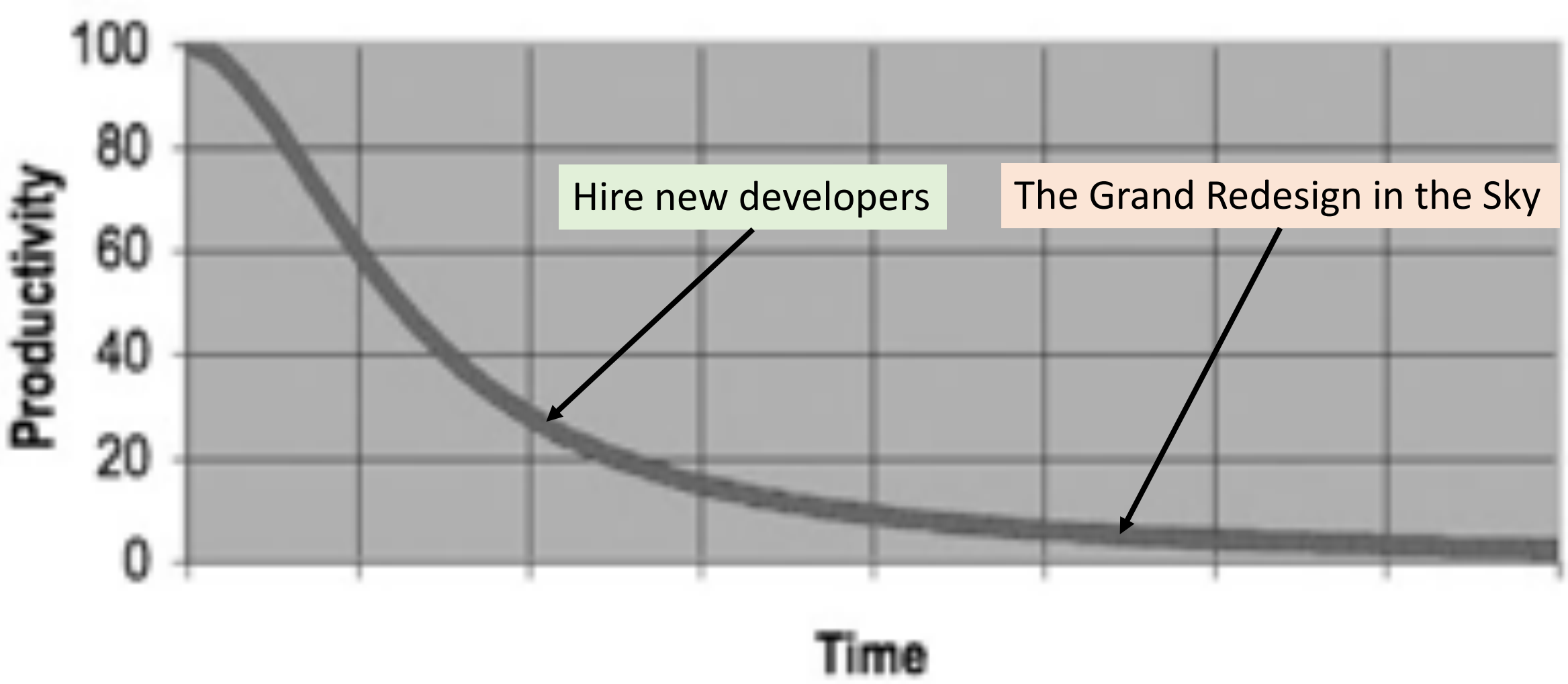
Specifying requirements *in such detail that a machine can execute them is* ***programming****. Such a specification is* ***code****.*



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

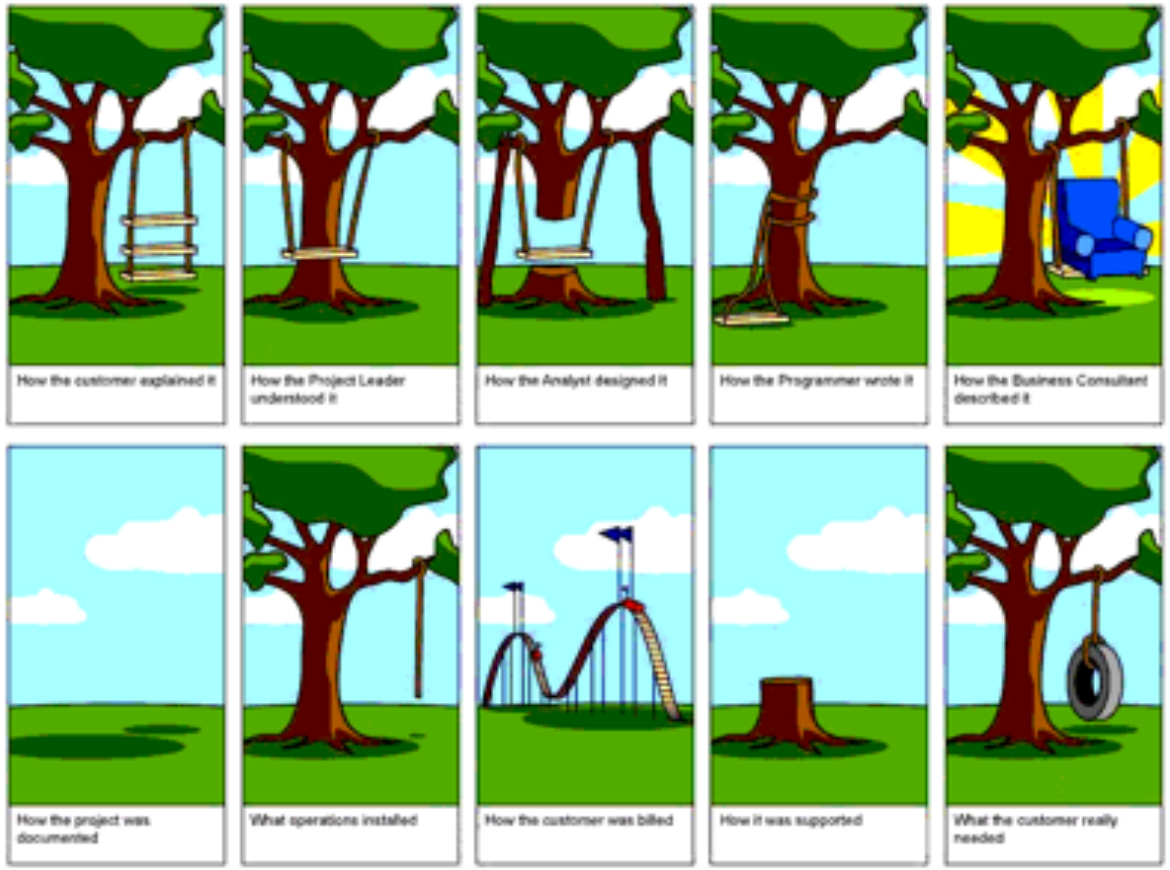
Stupid managers and intolerant customers and useless marketing types and telephone sanitizers

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*

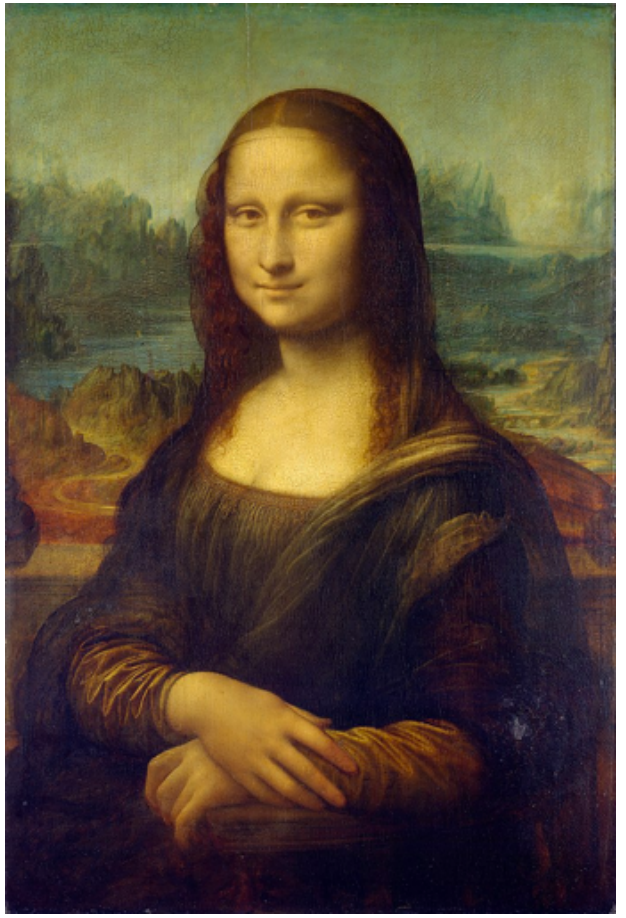


It is **unprofessional** for programmers to *bend* to the will of managers who <u>don't understand</u> the risks of making messes.

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?

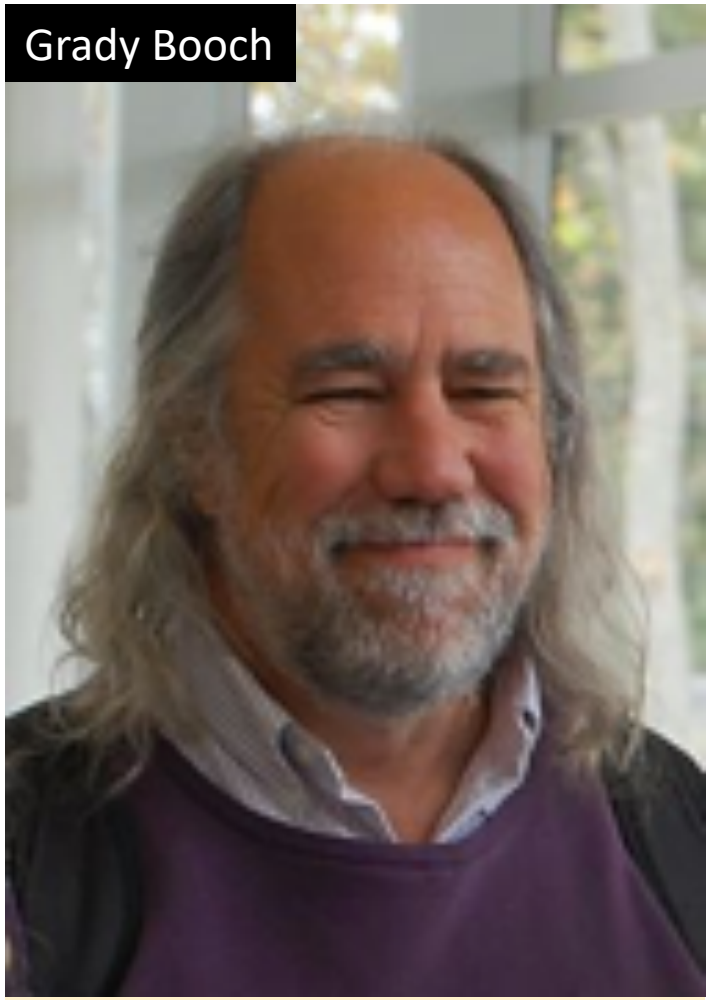
*I like my code to be **elegant** and **efficient**. The **logic should be straightforward** to make it hard for bugs to hide, the **dependencies minimal** to ease **maintenance**, **error handling complete** according to an articulated strategy, and **performance close to optimal** so as not to tempt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

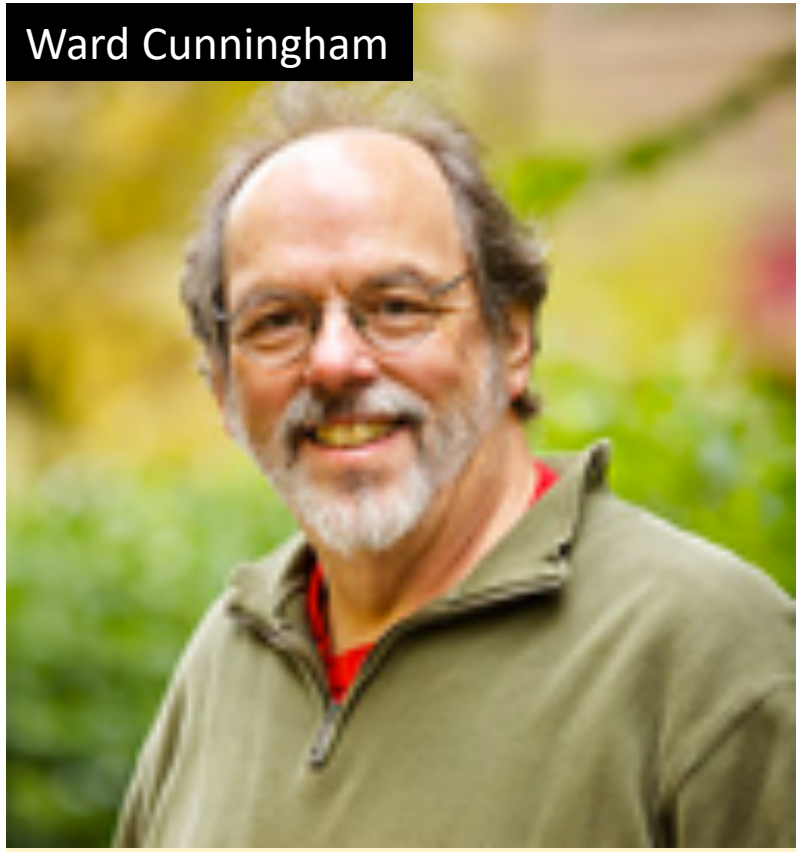
Ron Jeffries

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

What Is Clean Code?

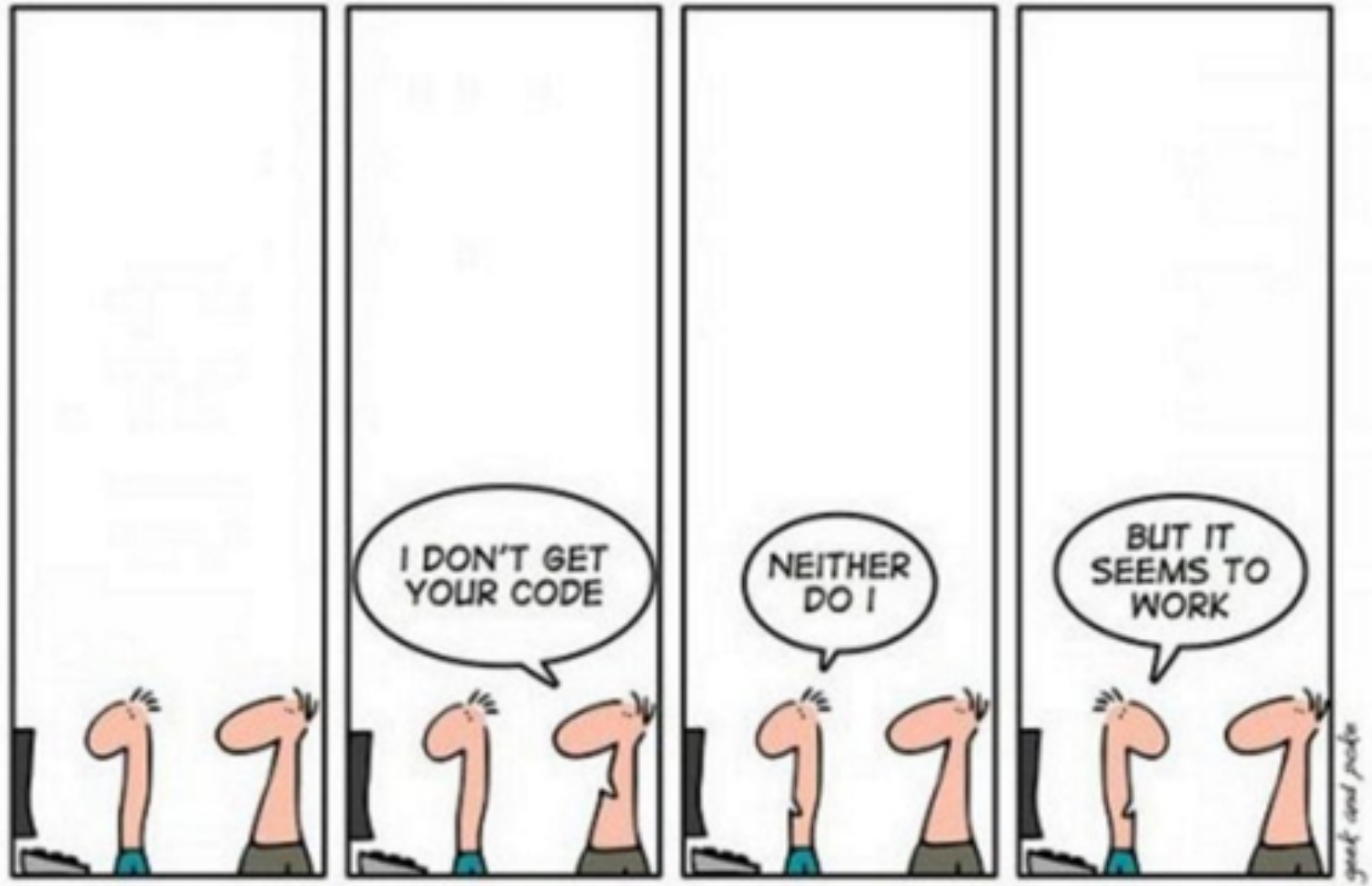
You know you are working on clean code when ***each routine you read turns out to be pretty much what you expected****. You can call it beautiful code when the code also makes* ***it look like the language was made for the problem****.*

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!


I DON'T GET YOUR CODE
NEITHER DO I
BUT IT SEEMS TO WORK
THE ART OF PROGRAMING

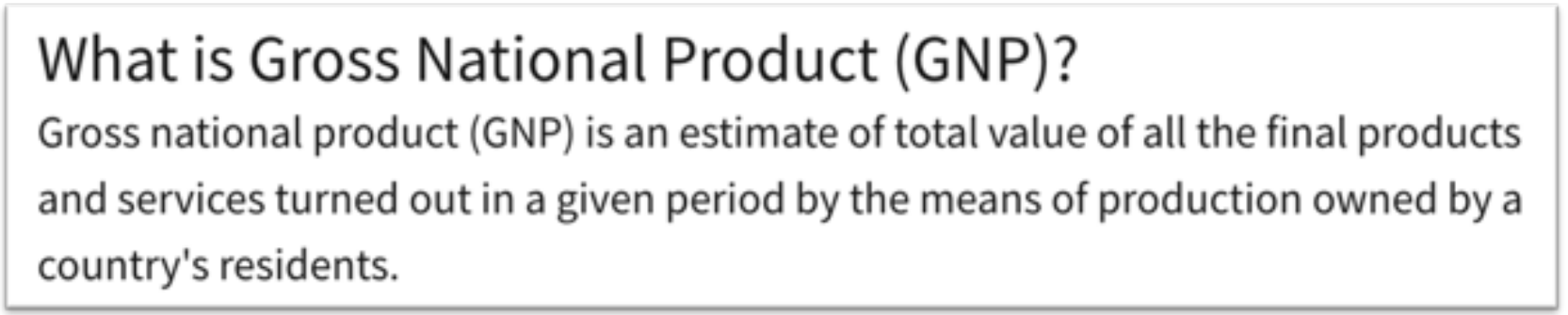
Introduction to Software Engineering

Topics Covered

- *Professional software development*
  - What is meant by software engineering?
- *Software engineering ethics*
  - A brief introduction to ethical issues that affect software engineering.
- *Case studies*
  - An introduction to three examples that are used in this course.

Software Engineering

- The economies of ALL developed nations are **dependent on software.**
- More and more systems are software controlled.
- Software engineering is ***concerned*** *with theories, methods and tools for professional software development.*
- Expenditure on software represents a significant fraction of Gross National Product (GNP) in all developed countries.



Software Costs

- Software costs often **dominate** computer system costs.
- Costs of software on a PC are *often greater* than the hardware cost.
- Software costs more to **maintain than it does to develop**. For systems with a long life, *maintenance costs* may be several times development costs.
- Software engineering is concerned with **cost-effective software development.**

Software Project Failure

- **Increasing system complexity**
  - As new software engineering techniques help us to build larger, more complex systems, the demands change. Systems have to be built and delivered more quickly; larger, even more complex systems are required; systems have to have new capabilities that were previously thought to be impossible.
- **Failure to use software engineering methods**
  - It is fairly easy to write computer programs without using software engineering methods and techniques.
  - Many companies have drifted into software development as their products and services have evolved. They do not use software engineering methods in their everyday work. Consequently, their software is often more expensive and less reliable than it should be.

Professional Software Development

FAQs About Software Engineering

Question	Answer
What is software?	**Computer programs and associated documentation**. Software products may be developed for a particular customer or may be developed for a general market.
What are the attributes of good software?	Good software should deliver the required **functionality** and **performance** to the user and should be **maintainable**, **dependable** and **usable**.
What is software engineering?	Software engineering is an engineering discipline that is concerned with all aspects of software production.
What are the fundamental software engineering activities?	Software **specification**, software **development**, software **validation** and software **evolution**.
What is the difference between software engineering and computer science?	Computer science focuses on theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software.
What is the difference between software engineering and system engineering?	System engineering is concerned with all aspects of computer-based systems development including hardware, software and process engineering. Software engineering is part of this more general process.

FAQs About Software Engineering

Question	Answer
What are the key challenges facing software engineering?	Coping with increasing diversity, demands for reduced delivery times and developing trustworthy software.
What are the costs of software engineering?	Roughly 60% of software costs are development costs, 40% are testing costs. For custom software, evolution costs often exceed development costs.
What are the best software engineering techniques and methods?	While all software projects have to be professionally managed and developed, **different techniques are appropriate for different types of system**. For example, games should always be developed using a series of prototypes whereas safety critical control systems require a complete and analyzable specification to be developed. You can't, therefore, say that one method is better than another.
What differences has the web made to software engineering?	The web has led to the availability of software services and the possibility of developing highly distributed service-based systems. Web-based systems development has led to important advances in programming languages and software reuse.

Software Products

- Software engineers are concerned with developing software products, that is, software that can be sold to a customer. There are two kinds of software product:
- **Generic products** - *Stand-alone systems that are marketed and sold to any customer who wishes to buy them.*
  - **Examples** – PC software such as graphics programs, project management tools; CAD software; software for specific markets such as appointments systems for dentists.
  - The specification of what the software should do is *owned by the software developer* and *decisions on software change* are made by the **developer**.
- **Customized products** - *Software that is commissioned by a specific customer to meet their own needs.*
  - **Examples** – embedded control systems, air traffic control software, traffic monitoring systems.
  - The specification of what the software should do is *owned by the customer* for the software and *they make decisions on software changes* that are required.

Attributes of Good Software

- When we talk about the **quality of professional software**, we have to consider that the software is used and changed by people apart from its developers.
- **Quality** is therefore not just concerned with what the software does. Rather, it has to include the software's behavior while it is executing and the structure and organization of the system programs and associated documentation.
- This is reflected in the software's quality or non-functional attributes.
- **Examples of these attributes** are the software's response time to a user query and the understandability of the program code.
- The specific set of attributes that you might expect from a software system obviously *depends on its application*. Therefore, an aircraft control system must be safe, an interactive game must be responsive, a telephone switching system must be reliable, and so on.

Essential Attributes of Good Software

Product characteristic	Description
Maintainability	Software should be written in such a way so that it can ***evolve to meet the changing needs of customers***. This is a critical attribute because software change is an **inevitable** requirement of a changing business environment.
Dependability and security	Software dependability includes a range of characteristics including **reliability**, **security** and **safety**. Dependable software should not cause physical or economic damage in the event of system failure. Malicious users should not be able to access or damage the system.
Efficiency	Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, memory utilisation, etc.
Acceptability	Software must be ***acceptable to the type of users for which it is designed***. This means that it must be understandable, usable and compatible with other systems that they use.

Software Engineering

- Software engineering is an *engineering discipline that is concerned with all aspects of software production from the early stages of system specification through to maintaining the system after it has gone into use.*
- **Engineering discipline**
  - Using appropriate theories and methods to solve problems bearing in mind organizational and financial constraints.
- **All aspects of software production**
  - Not just technical process of development. Also project management and the development of tools, methods etc. to support software production.

Importance of Software Engineering

- *More and more, individuals and society rely on advanced software systems*. We need to be able to produce reliable and trustworthy systems economically and quickly.
- *It is usually cheaper, in the long run, to use software engineering methods and techniques for software systems* rather than just write the programs as if it was a personal programming project. For most types of system, the majority of costs are the costs of changing the software after it has gone into use.

Software Process Activities

- *A software process* is the sequence of activities that leads to the production of a software product.
- Four fundamental activities are common to all software processes:
  - **Software specification**, where customers and engineers define the software that is to be produced and the constraints on its operation.
  - **Software development**, where the software is designed and programmed.
  - **Software validation**, where the software is checked to ensure that it is what the customer requires.
  - **Software evolution**, where the software is modified to reflect changing customer and market requirements.

General Issues that Affect Software

- **Heterogeneity** – Increasingly, systems are required to operate as distributed systems across networks that include different types of computer and mobile devices.
- **Business and social change** – Business and society are changing incredibly quickly as emerging economies develop and new technologies become available. They need to be able to change their existing software and to rapidly develop new software.
- **Security and trust** – As software is intertwined with all aspects of our lives, it is essential that we can trust that software.
- **Scale** – Software has to be developed across a very wide range of scales, from very small embedded systems in portable or wearable devices through to Internet-scale, cloud-based systems that serve a global community.

To address these challenges, we will need new tools and techniques as well as innovative ways of combining and using existing software engineering methods.

Software Engineering Diversity

- Software engineering is a systematic approach to the production of software that takes into account practical cost, schedule, and dependability issues, as well as the needs of software customers and producers.
- There are many different types of software system and there is no universal set of software techniques that is applicable to all of these.
- The software engineering methods and tools used depend on the *type of application being developed*, the *requirements of the customer* and the *background of the development team*.

Application Types

- **Stand-alone applications**

  These are application systems that run on a local computer, such as a PC. They include all necessary functionality and do not need to be connected to a network.

- **Interactive transaction-based applications**

  Applications that execute on a remote computer and are accessed by users from their own PCs or terminals. These include web applications such as e-commerce applications.

- **Embedded control systems**

  These are software control systems that control and manage hardware devices. Numerically, there are probably more embedded systems than any other type of system.

- **Batch processing systems**

  These are business systems that are designed to process data in large batches. They process large numbers of individual inputs to create corresponding outputs.

Application Types – Cont’d

- **Entertainment systems**

  These are systems that are primarily for personal use and which are intended to entertain the user.

- **Systems for modelling and simulation**
  - These are systems that are developed by scientists and engineers to model physical processes or situations, which include many, separate, interacting objects.
- **Data collection systems**
  - These are systems that collect data from their environment using a set of sensors and send that data to other systems for processing.
- **Systems of systems**
  - These are systems that are composed of a number of other software systems.

Software Engineering Fundamentals

Some fundamental principles apply to all types of software system, irrespective of the development techniques used:

- Systems should be developed using a managed and understood development process. Of course, *different processes are used for different types of software.*
- *Dependability and performance* are important for all types of system.
- Understanding and managing the *software specification and requirements* (what the software should do) are important.
- Where appropriate, you should *reuse software* that has already been developed rather than write new software.

- These fundamental notions of process, dependability, requirements, management, and reuse are important themes. Different methods reflect them in different ways, but they underlie all professional software development.

Internet Software Engineering

- The Web is now a platform for running application and organizations are increasingly developing web-based systems rather than local systems.
- Before the web, business applications were mostly **monolithic** running on single computers. Communications were local, within an organization.
- Now, software is **highly distributed**. Business applications are not programmed from scratch but involve reuse of components and programs.
- Web services (discussed in later in the course) allow application functionality to be accessed over the web.
- Cloud computing is an *approach to the provision of computer services where applications run remotely on the 'cloud'.*
  - Users do not buy software but pay according to use.

Web-based Software Engineering

- Web-based systems are complex **distributed** systems but the fundamental principles of software engineering discussed previously are as applicable to them as they are to any other types of system.
- The fundamental ideas of software engineering apply to web-based software in the same way that they apply to other types of software system.

Web Software Engineering

- **Software reuse**

  Software reuse is the dominant approach for constructing web-based systems. When building these systems, you think about how you can assemble them from pre-existing software components and systems.

- **Incremental and agile development**

  Web-based systems should be developed and delivered incrementally. It is now generally recognized that it is impractical to specify all the requirements for such systems in advance.

- **Service-oriented systems**

  Software may be implemented using service-oriented software engineering, where the software components are stand-alone web services.

- **Rich interfaces**

  Interface development technologies such as AJAX and HTML5 have emerged that support the creation of rich interfaces within a web browser.

Software Engineering Ethics

Software Engineering Ethics

- Software engineering is carried out within a social and legal framework that limits the freedom of people working in that area.
- Software engineering involves wider responsibilities than simply the application of technical skills.
- Software engineers must behave in an **honest** and **ethically** responsible way if they are to be respected as professionals.
- Ethical behaviour is more than simply upholding the law but involves following a set of principles that are morally correct.

Issues of Professional Responsibility

- **Confidentiality**

  Engineers should normally respect the confidentiality of their employers or clients irrespective of whether or not a formal confidentiality agreement has been signed.

- **Competence**

  Engineers should not misrepresent their level of competence. They should not knowingly accept work which is outside their competence.

- **Intellectual property rights**

  Engineers should be aware of local laws governing the use of intellectual property such as patents, copyright, etc. They should be careful to ensure that the intellectual property of employers and clients is protected.

- **Computer misuse**

  Software engineers should not use their technical skills to misuse other people's computers. Computer misuse ranges from relatively trivial (game playing on an employer's machine, say) to extremely serious (dissemination of viruses).

ACM/IEEE Code of Ethics

- The professional societies in the US have cooperated to produce a code of ethical practice.
- Members of these organisations sign up to the code of practice when they join.
- The Code contains **eight** principles related to the behaviour of and decisions made by professional software engineers, including practitioners, educators, managers, supervisors and policy makers, as well as trainees and students of the profession.

Rationale for the Code of Ethics

- Computers have a central and growing role in commerce, industry, government, medicine, education, entertainment and society at large.
- Software engineers are those who contribute by direct participation or by teaching, to the analysis, specification, design, development, certification, maintenance and testing of software systems.
- Because of their roles in developing software systems, software engineers have significant opportunities to do good or cause harm, to enable others to do good or cause harm, or to influence others to do good or cause harm.
- To ensure, as much as possible, that their efforts will be used for good, software engineers must commit themselves to making software engineering a beneficial and respected profession.

Software Engineering Code of Ethics and Professional Practice

ACM/IEEE-CS Joint Task Force on Software Engineering Ethics and Professional Practices

PREAMBLE

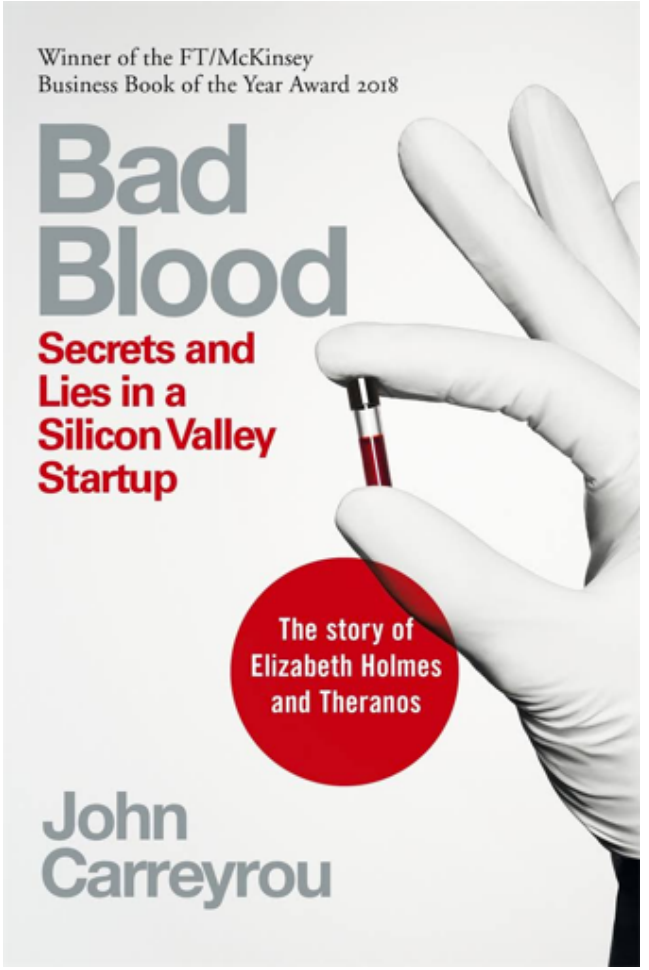
The short version of the code summarizes aspirations at a high level of the abstraction; the clauses that are included in the full version give examples and details of how these aspirations change the way we act as software engineering professionals. Without the aspirations, the details can become legalistic and tedious; without the details, the aspirations can become high sounding but empty; together, the aspirations and the details form a cohesive code.

Software engineers shall commit themselves to making the analysis, specification, design, development, testing, and maintenance of software a beneficial and respected profession. In accordance with their commitment to the health, safety, and welfare of the public, software engineers shall adhere to the following Eight Principles:

1. PUBLIC – Software engineers shall act consistently with the public interest.
2. CLIENT AND EMPLOYER – Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.
3. PRODUCT – Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.
4. JUDGMENT – Software engineers shall maintain integrity and independence in their professional judgment.
5. MANAGEMENT – Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
6. PROFESSION – Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.
7. COLLEAGUES – Software engineers shall be fair to and supportive of their colleagues.
8. SELF – Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

Ethical Dilemmas

- Disagreement in principle with the policies of senior management.
- Your employer acts in an unethical way and releases a safety-critical system without finishing the testing of the system.
- Participation in the development of military weapons systems or nuclear systems.



https://www.youtube.com/watch?v=9wf_2KYRPWQ

Case Studies

Case Studies

- **A personal insulin pump**

  An embedded system in an insulin pump used by diabetics to maintain blood glucose control.
- **Mentcare: a mental health case patient management system**

  A system used to maintain records of people receiving care for mental health problems.
- **A wilderness weather station**

  A data collection system that collects data about weather conditions in remote areas.
- **iLearn: a digital learning environment**

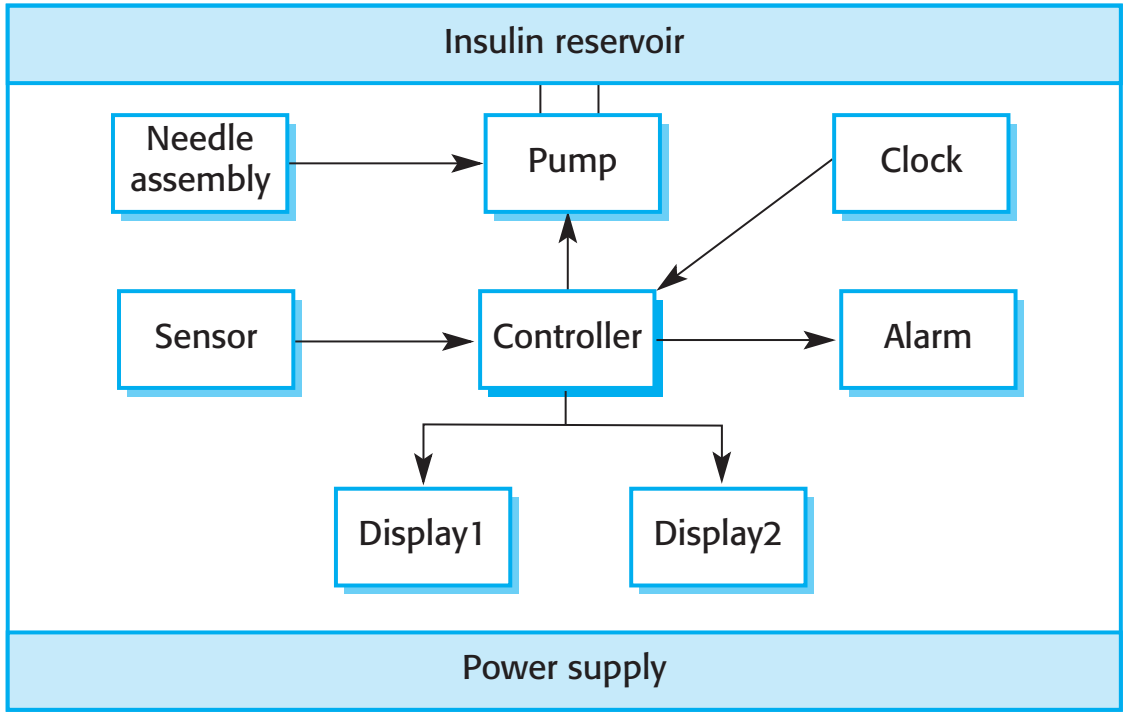
  A system to support learning in schools

Insulin Pump Control System

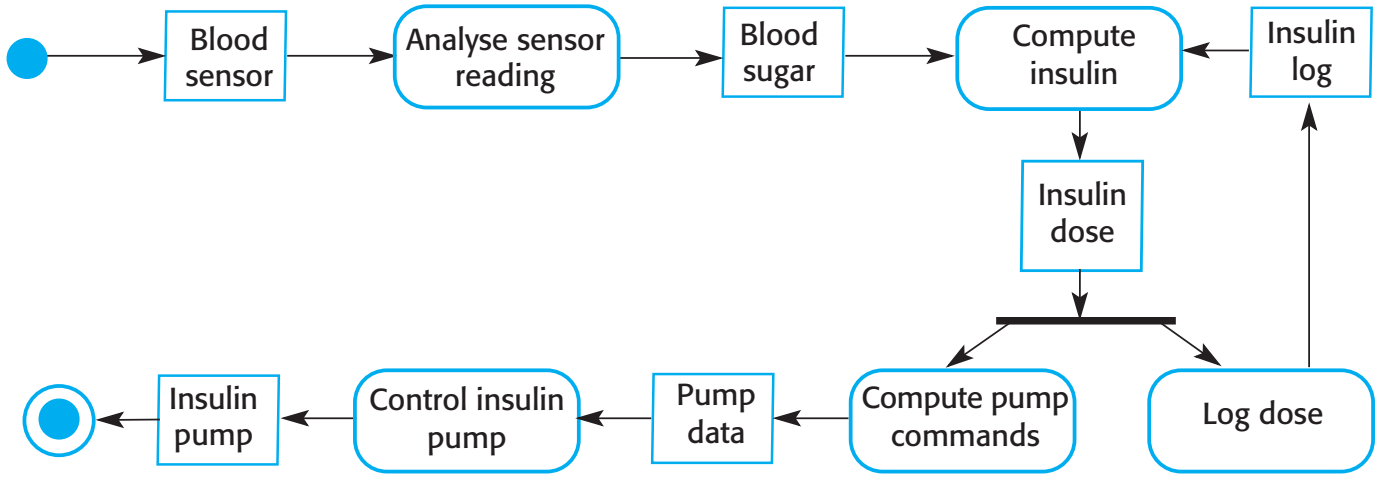
- Collects data from a blood sugar sensor and calculates the amount of insulin required to be injected.
- Calculation based on the rate of change of blood sugar levels.
- Sends signals to a micro-pump to deliver the correct dose of insulin.
- Safety-critical system as low blood sugars can lead to brain malfunctioning, coma and death; high-blood sugar levels have long-term consequences such as eye and kidney damage.

Insulin Pump Hardware Architecture

- A software-controlled insulin delivery system uses a microsensor embedded in patient to measure some blood parameter that is proportional to the sugar level.
- This is then sent to the pump controller to compute the sugar level and the amount of insulin that is needed.
- It then sends signals to a miniaturized pump to deliver the insulin via a permanently attached needle.



UML Activity Model of the Insulin Pump



illustrates how the software transforms an input blood sugar level to a sequence of commands that drive the insulin pump

Essential High-Level Requirements

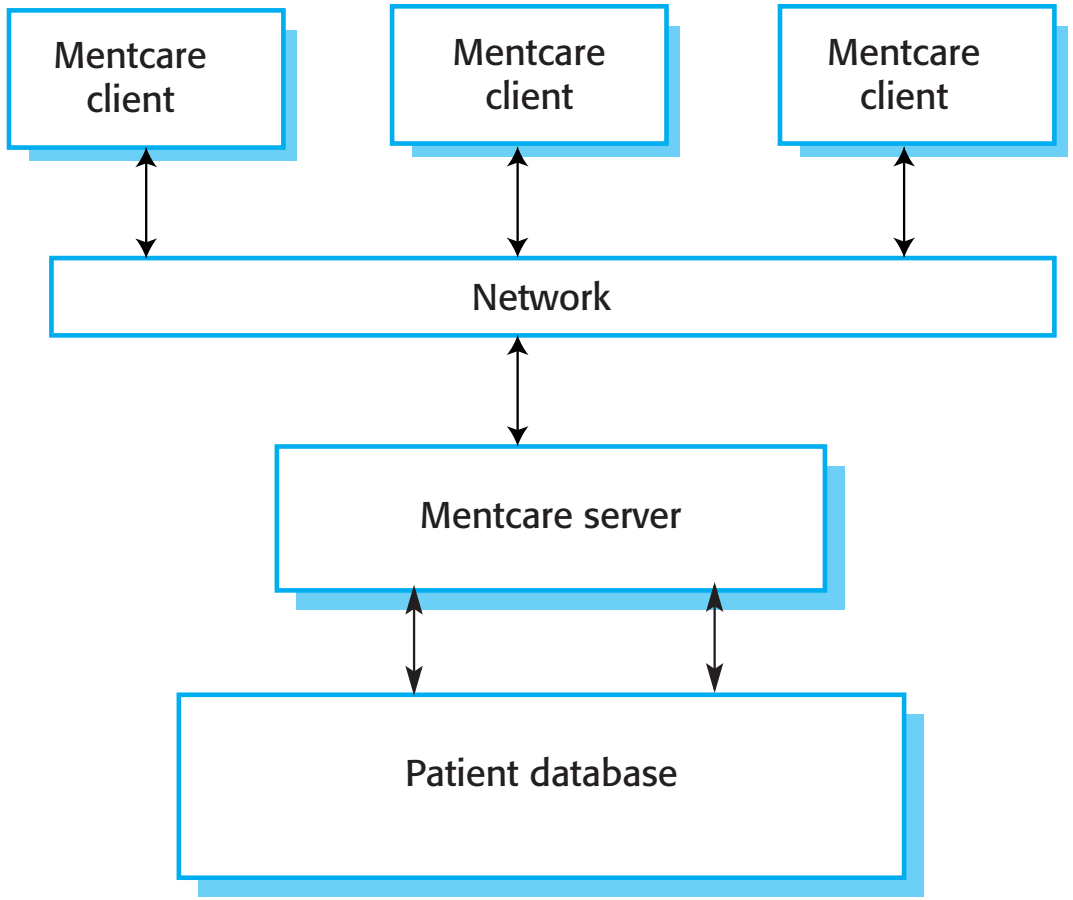
- The system shall be available to deliver insulin when required.
- The system shall perform reliably and deliver the correct amount of insulin to counteract the current level of blood sugar.
- The system must therefore be designed and implemented to ensure that the system always meets these requirements.

Mentcare: *A patient information system for mental health care*

- A patient information system to support mental health care is a medical information system that maintains information about patients suffering from mental health problems and the treatments that they have received.
- Most mental health patients do not require dedicated hospital treatment but need to attend specialist clinics regularly where they can meet a doctor who has detailed knowledge of their problems.
- To make it easier for patients to attend, these clinics are not just run in hospitals. They may also be held in local medical practices or community centres.

Mentcare

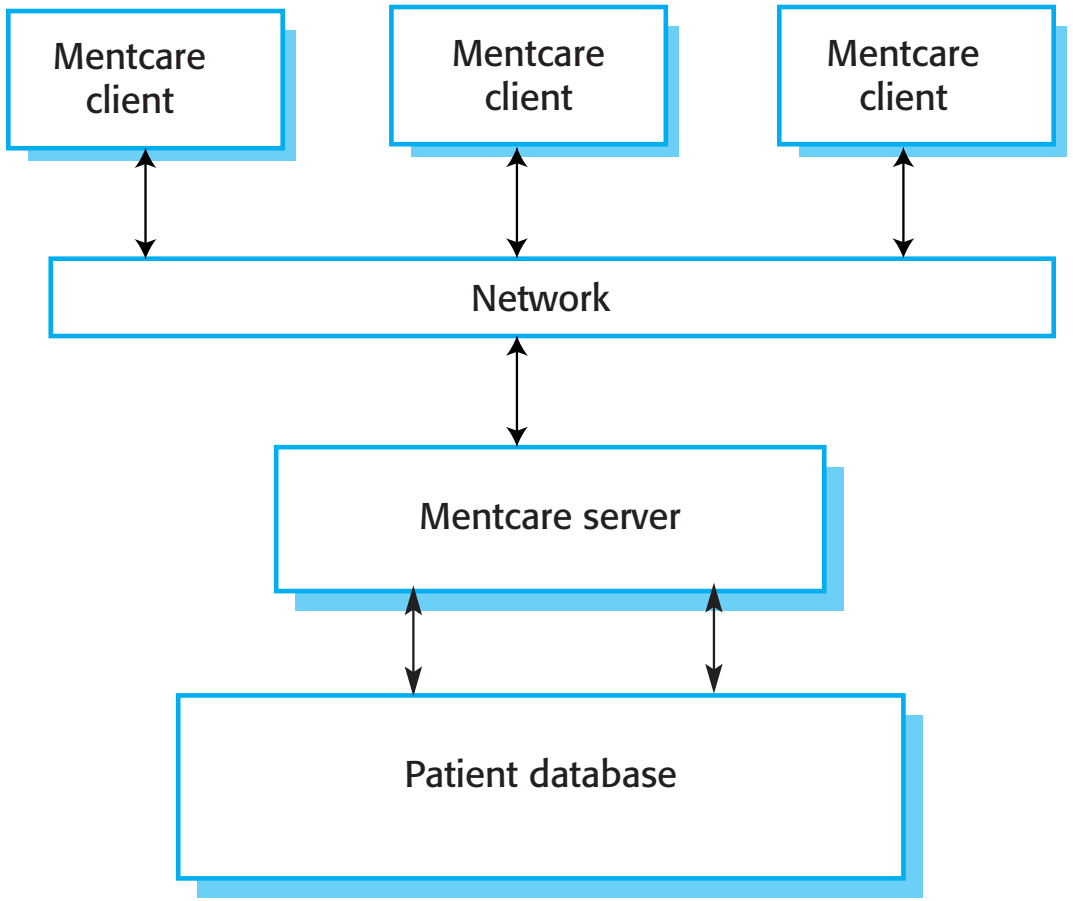
- Mentcare is an information system that is intended for use in clinics.
- It makes use of a **centralized database** of patient information but has also been designed to run on a PC, so that it may be accessed and used from sites that do not have secure network connectivity.
- When the local systems have secure network access, they use patient information in the database but they can download and use local copies of patient records when they are disconnected.



Mentcare Goals

- To generate management information that allows health service managers to assess performance against local and government targets.
- To provide medical staff with timely information to support the treatment of patients.

Organization of the Mentcare System



Key Features of the Mentcare System

- **Individual care management**

  Clinicians can create records for patients, edit the information in the system, view patient history, etc. The system supports data summaries so that doctors can quickly learn about the key problems and treatments that have been prescribed.

- **Patient monitoring**

  The system monitors the records of patients that are involved in treatment and issues warnings if possible problems are detected.

- **Administrative reporting**

  The system generates monthly management reports showing the number of patients treated at each clinic, the number of patients who have entered and left the care system, number of patients sectioned, the drugs prescribed and their costs, etc.

Mentcare System Concerns

- **Privacy**
  - It is essential that patient information is confidential and is never disclosed to anyone apart from authorised medical staff and the patient themselves.
- **Safety**
  - Some mental illnesses cause patients to become suicidal or a danger to other people. Wherever possible, the system should warn medical staff about potentially suicidal or dangerous patients.
  - The system must be available when needed otherwise safety may be compromised and it may be impossible to prescribe the correct medication to patients.

Wilderness Weather Station

- The government of a country with large areas of wilderness decides to deploy several hundred weather stations in remote areas.
- Weather stations collect data from a set of instruments that measure temperature and pressure, sunshine, rainfall, wind speed and wind direction.
  - The weather station includes a number of instruments that measure weather parameters such as the wind speed and direction, the ground and air temperatures, the barometric pressure and the rainfall over a 24-hour period.
  - Each of these instruments is controlled by a software system that takes parameter readings periodically and manages the data collected from the instruments.

Weather Information System

- **Weather Station System**

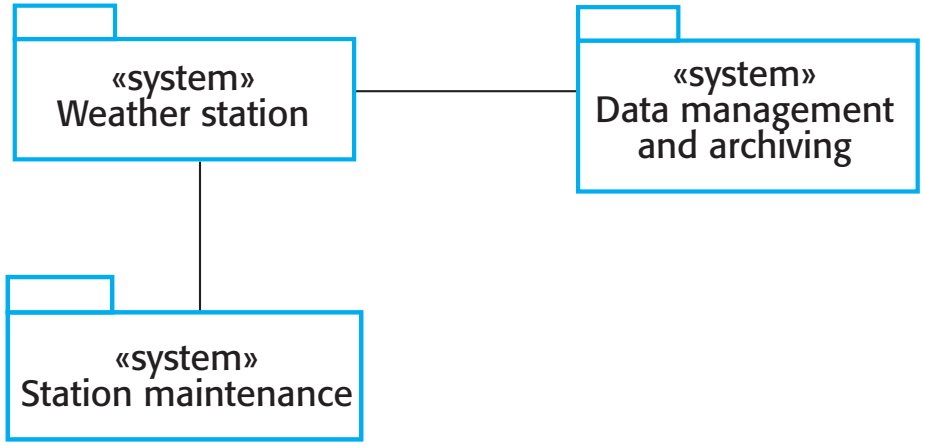
  This is responsible for collecting weather data, carrying out some initial data processing and transmitting it to the data management system.

- **Data Management and Archiving System**

  This system collects the data from all of the wilderness weather stations, carries out data processing and analysis and archives the data.

- **Station Maintenance System**

  This system can communicate by satellite with all wilderness weather stations to monitor the health of these systems and provide reports of problems.



Additional Software Functionality

- Monitor the instruments, power and communication hardware and report faults to the management system.
- Manage the system power, ensuring that batteries are charged whenever the environmental conditions permit but also that generators are shut down in potentially damaging weather conditions, such as high wind.
- Support dynamic reconfiguration where parts of the software are replaced with new versions and where backup instruments are switched into the system in the event of system failure.

iLearn: *A digital Learning Environment*

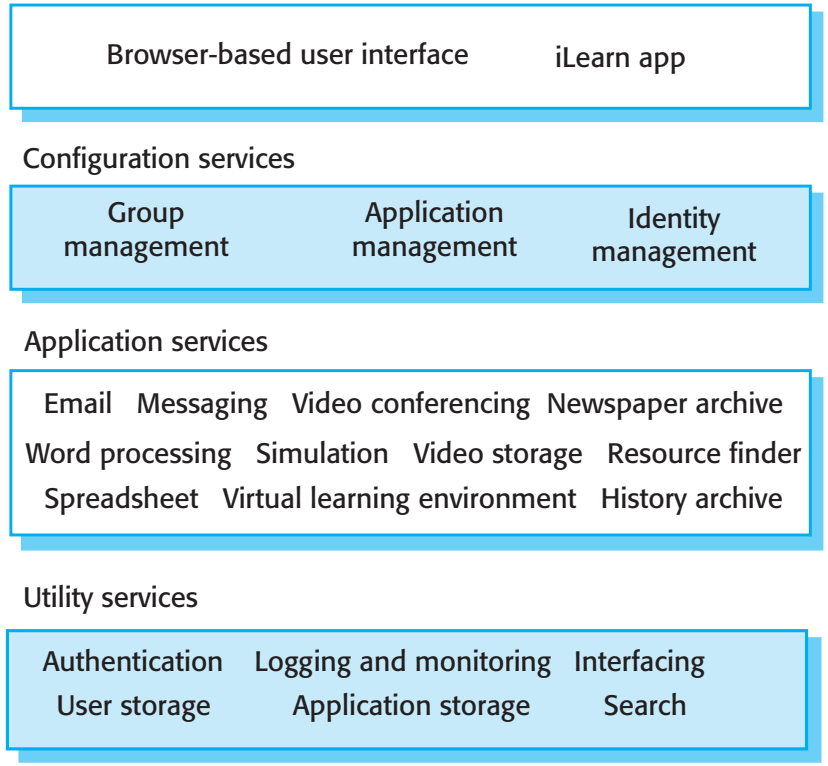
- A digital learning environment is a framework in which a set of general-purpose and specially designed tools for learning may be embedded plus a set of applications that are geared to the needs of the learners using the system.
- The tools included in each version of the environment are chosen by teachers and learners to suit their specific needs.
  - These can be general applications such as spread sheets, learning management applications such as a Virtual Learning Environment (VLE) to manage homework submission and assessment, games and simulations.

iLearn: *A Service-Oriented System*

- The system is a service-oriented system with all system components considered to be a replaceable service.
- This allows the system to be updated incrementally as new services become available.
- It also makes it possible to rapidly configure the system to create versions of the environment for different groups such as very young children who cannot read, senior students, etc.

iLearn Services

- ***Utility services*** that provide basic application-independent functionality and which may be used by other services in the system.
- ***Application services*** that provide specific applications such as email, conferencing, photo sharing and access to educational content such as scientific films or historical resources.
- ***Configuration services*** that are used to adapt the environment with a specific set of application services and do define how services are shared between students, teachers and their parents.



iLearn: *Service Integration*

- The environment has been designed so that services can be replaced as new services become available and to provide different versions of the system that are suited for the age of the users.
- This means that the system has to support two levels of service integration:
  - *Integrated services* are services which offer an API (application programming interface) and which can be accessed by other services through that API. Direct service-to-service communication is therefore possible.
  - *Independent services* are services which are simply accessed through a browser interface and which operate independently of other services. Information can only be shared with other services through explicit user actions such as copy and paste; re-authentication may be required for each independent service.

Key Points

- Software engineering is an engineering discipline that is concerned with all aspects of software production.
- Essential software product attributes are maintainability, dependability and security, efficiency and acceptability.
- The high-level activities of specification, development, validation and evolution are part of all software processes.
- The fundamental notions of software engineering are universally applicable to all types of system development.

Key Points

- There are many different types of system and each requires appropriate software engineering tools and techniques for their development.
- The fundamental ideas of software engineering are applicable to all types of software system.
- Software engineers have responsibilities to the engineering profession and society. They should not simply be concerned with technical issues.
- Professional societies publish codes of conduct which set out the standards of behaviour expected of their members.

つづく