Software Engineering Basics

JD Kilgallin

CPSC:480

08/24/22

Pressman Ch 1

Quality is the ally of schedule and cost, not their adversary. -James Ward

Notes

- Textbook is strongly encouraged.
  - I *will* use questions from the book.
  - You can get points back for anything marked wrong if you can defend your answer based on the book.
  - It's likely to be one of the most relevant books to your early career.
- Unannounced quizzes may occur, but will usually be announced.
- "How often do you run into a problem you don't know how to solve?"
  - At least a couple times a week. This is the great thing about teams, and the Internet!
- Mac vs Windows for class?
  - C, C++, Python: Mac (or Linux). C#, SQL, web: Windows. Java: tossup, lean Windows.
- What are the main fields for local companies?
  - Insurance (Progressive), Healthcare (CC), Business software (Hyland), Banking (Key), General (First Energy, Goodyear, Sherwin Williams, Joann Fabrics, Smuckers) I.M.E.

Survey Statistics

- Almost everyone is a CS major, with a laptop, seeking job opportunities.
- Mixed graduation dates, hobbies, prior GitHub experience.
- Areas of interest: Game dev (9), web (3), data (3), security (3), IoT (2), IT/support (2), Geographic Information Systems, Streaming, Apple, Cloud
- Languages (0.1="a little", "learning", or "not proficient")
  - 27.0 C/C++
  - 10.4 Web (HTML, JavaScript, TypeScript, PHP)
  - 08.1 C#
  - 07.3 Python
  - 05.2 Java
  - 02.1 SQL
  - 02.0 Assembly
  - 00.2 Visual BASIC

"Is it hard to get software jobs in the area?"

- Northeast Ohio is not known for its software industry, but many major companies here hire software engineers for internal/external development and there are many smaller software vendors.
- Job placement agencies make it much easier to get a good offer. They get *paid on commission* to find you a job you'll stay in. I recommend Robert Half. UA career center might help too.
- Remote software work is on the rise, and this is a great spot for cost-of-living!
- More on jobs in lecture 4.

Learning Objectives

- History of software and software engineering
- Role of software in the world
- Classification of software systems
- Role of a Software Engineer in an organization
- Core concepts of the software engineering profession

What is software?

- A set of instructions and data to execute specific tasks on a computer.
- "Software is: (1) instructions (computer programs) that when executed provide desired features, function, and performance; (2) data structures that enable the programs to adequately manipulate information; and (3) descriptive information in both hard copy and virtual forms that describes the operation and use of the programs."
- "Computer software is a work product that software professionals build and then support over many years. These work products include programs that execute within computers of any size and architecture"
- The practice of storing computer instructions as code in the same form as program input data.

Invention of software – Pre-industrial era

- I Ching, divination text ca. 1000 BC, influenced Confucius & Taoism. Describes binary encoding (yin-yang), symbolic representation, and "virtual" reality as an infinite combination of grouped binary codes.
- 17th century German natural philosopher Gottfried Leibniz seeks systems of universal arithmetic, algebra of concepts, alphabet of thought. Invents differential and integral calculus, inspires Einstein.
- 1679-1705, influenced by I Ching, considers binary arithmetic as a fundamental concept from which everything else is derived. Describes a computer divining truth using rolling balls in a frame with holes.

Invention of software – 19th century concept

- Charles Babbage, English mathematician & mechanical engineer, develops *Difference Engine*, a mechanical calculator, in 1823.
- Successor *Analytical Engine* includes memory & control flow as data.
- First design in 1837 and iterated through 1871, never* actually built.
- Babbage gives lecture on this Analytical Engine in Italian in 1840.
- Ada King, Countess of Lovelace and English mathematician, was commissioned in 1842 to translate lecture transcription to English.
- Expands translation with her own notes, including complete program design to compute *Bernoulli Numbers* (sums of powers of integers).
- Also described, but did not implement, programs for music composition.

Invention of software–Entscheidungsproblem

- A *Diophantine equation* is a polynomial with integer solutions (roots).
- David Hilbert, German mathematician, presented 23 unsolved problems in 1900 that would influence 20th century mathematics.
- Problem 10 ("Entscheidungsproblem") asks for an algorithm to determine if an arbitrary Diophantine Equation has a solution.
- Impossibility was proven in 1935 by Alonzo Church (Church's Theorem) using a formal logic system called λ-calculus with concept of "calculability". Influenced by Kurt Gödel's "Incompleteness" theorems.

Invention of software – Turing Machines

- Independent computation-oriented proof on the Entscheidungsproblem published in Alan Turing's 1936 seminal paper.
- Turing's proof:
  - Introduced a mathematical model of computation (Turing Machine)
  - Introduced a "Universal Turing Machine" that could simulate others
  - Proved that no algorithm could determine if an arbitrary Turing Machine would run forever – the "Halting Problem"
  - Proved equivalence of this to the Entscheidungsproblem (in 64 words!)
  - Proved equivalence of Turing Machines and λ-calculus.
  - Formed the basis of modern theory and practice of computer science

History of computing – WWI computation

- Firearms technology improved by WWI to the point that trajectory could be predicted, taking into account factors down to air density and rotation of the earth.
- In 1917, the US Army Ordnance Corps commissioned Aberdeen Proving Ground in Maryland to perform ballistics testing and compile tables of figures needed to aim projectiles at a desired target.
- Most calculations were done by hand using formula sheets to formalize algorithms and sometimes simple mechanical calculators. *Computer* was the profession of performing these calculations.

Invention of software – WWII computation

- Team was re-mobilized for Aberdeen Proving Ground in 1937.
- John Von Neumann focused on modeling explosions & shockwaves.
- In February 1943 Von Neumann was introduced to an NCR machine and principles of computing used for military purposes in England.
- On his return he was enlisted on the Manhattan Project at Los Alamos using IBM electrical machines to model detonation of nuclear bombs. "Fat Man" bomb detonated in Nagasaki used his design.
- Working also in Aberdeen, helped develop the first general-purpose digital computer, the "Electronic Numerical Integrator And Calculator" (ENIAC). Used for artillery table and nuclear detonation calculation and unveiled after the war. Programmed by hard-wiring circuits.

Invention of software – Post-war conditions

- Electronic machines had progressed from vacuum tubes operating in a "pressure" model, to cathode ray tubes with directed electron motion, to controlled "bunching" of electrons for signaling.
- Claude Shannon of Bell Laboratories had developed information theory to a point where logical circuits could function indefinitely even on unreliable hardware using the concept of a *bit* of data.
- Dynamic memory in the form of self-punching tape or acoustic signals in a dense medium had been successfully implemented. Notably these did not allow random access and performed *very* slowly.
- After WWII, military funding and Von Neumann's expertise was available for technological projects.

Invention of software – von Neumann model

- Von Neumann, Shannon, and Radio Corporation of America (RCA) engineers launched the Princeton IAS Electronic Computing Project.
- Von Neumann's vision was to realize Turing's principle of a Universal Computation Machine. Theory meets practice.
- Functional components were defined to include hierarchical memory, a control unit, an arithmetic unit, and input/output channels.
- Hierarchical memory includes, at the least, a fast internal memory, a larger secondary memory, and an unbounded external storage (originally using stacks of punch cards).
- *Memory stores program in the same manner as other data - software*

Invention of Software – First software

- Von Neumann design implemented in 1952 Mathematical Analyzer Numerical Integrator and Automatic Computer (MANIAC).
- Prototype system "Small-Scale Experimental Machine" (SSEM) built with test program by Tom Kilburn to find the largest factor of 262,144 (=2^{18}). Considered the first piece of software.
- John's wife Klári largely wrote the first "real" application, a Monte Carlo simulation (analysis technique by sampling a large number of data points) of neutrons in a hydrogen bomb. She was also an author on the first peaceful application: predicting weather.
- These were first executed in April 1948 by modifying the ENIAC to include MANIAC-style program storage while MANIAC was still under construction.

Invention of software – Key players recap

- Charles Babbage and Ada Lovelace credited for first program design
- Gottfried Leibniz, David Hilbert, Alonzo Church, Kurt Gödel, Claude Shannon (and others) developed mathematical/statistical foundations
- Church's student Alan Turing developed a formal theory of computing
- Electrical computing hardware developed for Allied artillery and corporate business processes in first half of 20th century.
- Hilbert's student John von Neumann applied Turing's theory to build a general-purpose digital computer following World War II.
- The von Neumann model, including program-as-data, persists to modern computing systems, and forms the basis for *software*.

History of software - 1950s

- Most applications are still built on custom, single-purpose hardware
- Most stored-program software is for scientific and military simulation.
- By 1952, tic-tac-toe and checkers software had been implemented on multiple systems. Rudimentary pool and tennis by 1958.
- First chess AI to beat a human written for MANIAC in 1956.
- First recreational puzzle-solving engine, for *Polyominoes* (similar to Tetris), in 1958, written for MANIAC by Dana Scott.
- First modern programming language and compiler, FORTRAN, by John Backus at IBM in 1957. Supported on virtually all systems by 1963.

History of software – 1960s

- Additional programming languages like COBOL and ALGOL.
- First Turing Award in 1966 goes to Alan Perlis for programming techniques and compilers.
- First textbooks on software programming and programming languages, and first departments/colleges of computer science begin to train professional software developers.
- Apollo program leads to first computer made entirely of silicon integrated circuits and brings software to public consciousness.
- Term "Software Engineering" coined by Margaret Hamilton on Apollo program and established as a professional discipline.

History of software – 1970s

- Standalone software packaged and sold to consumers without involvement of hardware vendors. Companies specifically in the business of producing software.
- Applications typically cost over $50k (about $250k in 2022 dollars).
- First legal cases on software ownership and licensing.
- C programming language and UNIX OS released.
- First open-source software still in current use, such as TeX typesetter.
- Apple II released to mass market with color graphics & spreadsheets.
- Network protocols and global computer networks created (notably ARPANET, sponsored by US DoD), now grown to the modern Internet.

History of software – 1980s

- Mass market hardware e.g. IBM Personal Computer & Commodore 64
- Mass market software, with applications down to $50.
- Software consistently distributed in digital form rather than print.
- Rise of Microsoft with BASIC language and then Windows OS.
- Popularization of arcades and second/third-generation gaming consoles including Nintendo Entertainment System and Game Boy.
- Many modern software products including Office, GNU, AutoCAD
- World Wide Web, Domain Name System, Internet Service Providers.
- First college students who never experienced a pre-software world.

History of software - 1990s

- HTTP, HTML, JavaScript, WiFi, browsers, search engines, and mass adoption of Internet connectivity. 5% of world has access by 1999. Enables online software sales and downloads as well as webapps.
- Visual Studio, Python, Java, rise of Object-Oriented Programming.
- 3-D graphics and games (e.g. Doom) and computer animation.
- AI software developed to the point that IBM's Deep Blue beats world champion Gary Kasparov in chess.
- First degrees in software engineering (rather than computer science).
- Software inspires cyberpunk literary genre.
- Unified process model and unified modelling language developed.

History of software – 21st century

- Smartphones and portable devices create new software requirements
- Robotics proliferate with unique software requirements.
- Modern development and runtime environments and frameworks.
- Software-as-a-Service, Cloud computing and cloud-native applications
- Online gaming, social networking, & chat software become common.
- Personal computer and Internet use breaks 90% in US.
- Wikipedia and Google allow users to access vast amounts of information through software applications.
- 27 million professional software engineers today.

Why is software *important*?

- Every individual in the world benefits from software to some extent, and in the US the average person spends more than 8 hours a day using software.
- Allows individuals to perform tasks & acquire info they otherwise couldn't
- Used in lifesaving equipment, e.g. medical, emergency response, military
- Provides endless entertainment, with 72% of US playing video games.
- Enables businesses to better compete in an open market.
- Contributes trillions of dollars to the economy (Internet alone >$1 trillion)
- Easy to proliferate – single application can serve billions of users without mass production logistics or products wearing down. Allows large impact.
- Study of software structure provides insights to other areas of science.

What is software engineering?

- Engineering: The branch of science and technology concerned with the design, building, and use of engines, machines, and structures.
- Software engineering: "The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software"
- "The practical application of scientific knowledge to the creative design and building of computer programs."

Why is software *engineered?*

- The past 70 years of software development has led to identification of effective principles, techniques, and processes for software systems.
- Well-engineered software is easier to enhance and adapt, and therefore *cheaper* to develop and maintain.
- Poorly engineered software will ultimately fail to meet user expectations in the presence of competing/disrupting products.
- Software is ubiquitous and present in important settings including medical, automotive and financial applications. Faulty software causes real physical and economic harm.

What types of software are there?

- Form factor – desktop, mobile, web, embedded, distributed, cloud
- Purpose – application, utility, system, drivers, etc.
- Context – standalone executable, script, plugin, service, library
- Consumer – individual, enterprise, advertisers, developers
- Framework – Java, .NET, Ruby on Rails, etc.
- Architecture – Monolith, microservices, layers, Model-View-Controller
- Source language is *not* a distinguishing feature of software. Most software contains components written in many different languages, and it is not generally possible to tell what language was used.

Software application domains

- System – Written to support other software (OS, compiler, editor)
- Application – Standalone software that meets a specific need
- Scientific/engineering – Data processing software for computation
- Embedded – Control software in an appliance (e.g. car, microwave)
- Product-line – Software systems built from reusable components
- Mobile/web – Software designed for portability and remote access
- Artificial Intelligence – Uses large datasets to solve problems that are intractable by other means, e.g. language processing.
- *Gaming – Uses computer graphics for interactive entertainment

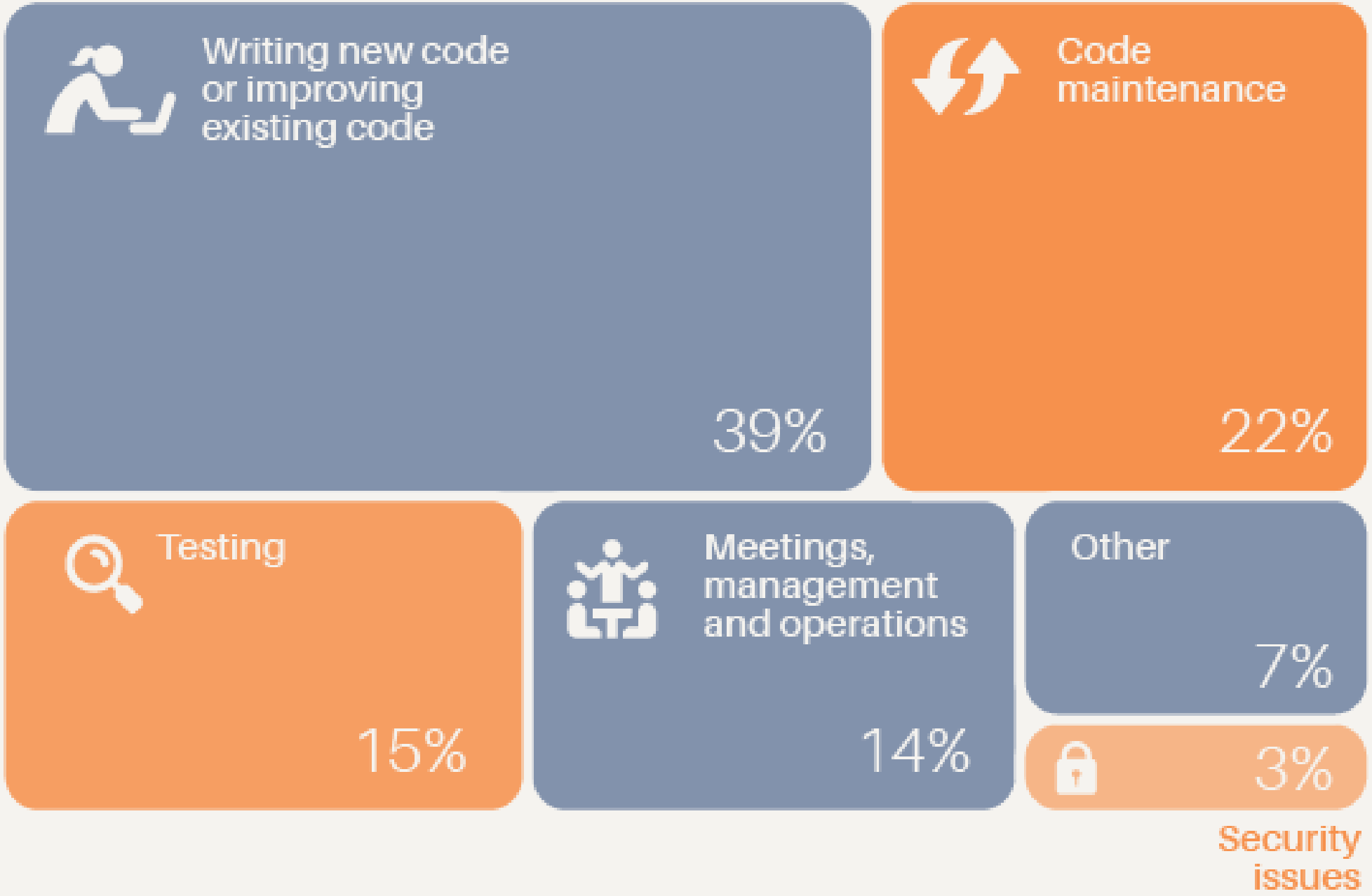
Classifying by specification

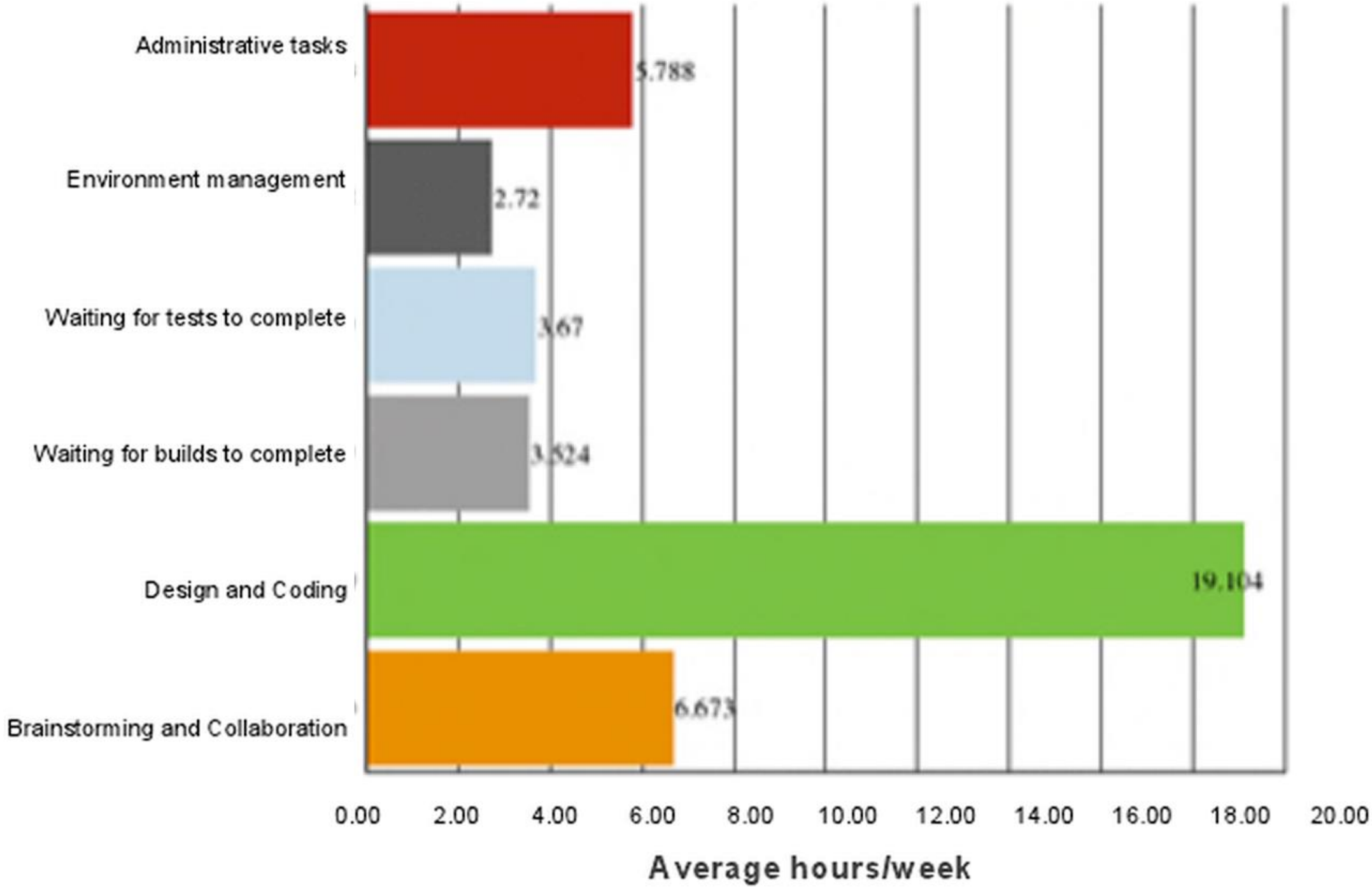
- An *S*-program is written according to an exact specification of what that program can do.
- A *P*-program is written to implement certain procedures that completely determine what the program can do (e.g. a program to play chess).
- An *E*-program is written to perform some real-world activity and needs to adapt to varying requirements and circumstances in the environment in which it runs.
- Almost all modern software of interest is E-type.

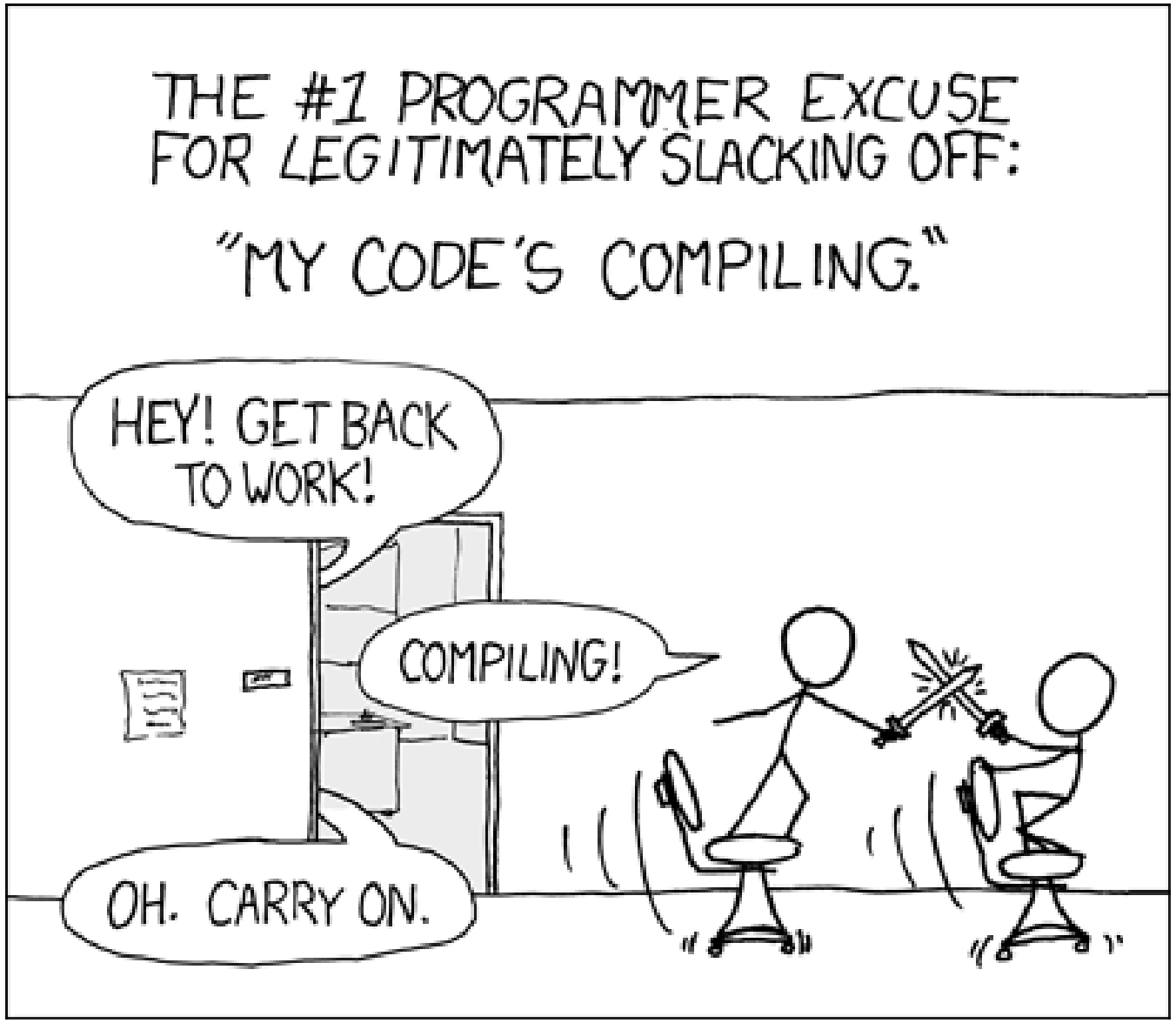
What do software engineers do?

- Write code.
- Design and architect programs to meet requirements.
- Test, troubleshoot, and maintain code.
- Integrate new and modified code with other developers' code.
- Communicate with product managers, customers, and others to ensure software meets users' needs.
- Track tasks being worked on, complete, and still pending.
- Work with other departments to meet company objectives.

SOFTWARE DEVELOPER/ENGINEER




Administrative tasks
5.788
Environment management
2.72
Waiting for tests to complete
3.67
Waiting for builds to complete
3.524
Design and Coding
19.104
Brainstorming and Collaboration
6.673
0.00
2.00
4.00
6.00
8.00
10.00
12.00
14.00
16.00
18.00
20.00
Average hours/week


THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:
"MY CODE'S COMPILING."
HEY! GET BACK TO WORK!
COMPILING!
OH. CARRY ON.

Terminology Arcana

- "Software engineering" and "software development" are interchangeable.
- "Software Engineer(ing)" is abbreviated SWE, not SE, to avoid confusion.
- "Software" may not be written out everywhere that it would be in the most formal setting (e.g. "development", "testing", even "engineering")
- Programming, coding, implementation, construction may be used interchangeably. Sometimes this will include testing, but not always. Use of the term "development" to refer only to coding is discouraged.
- A software *project* refers to a planned iteration of software development. This differs from Free & Open-Source Software (FOSS) usage.
- A software *product* is the artifact of a commercial software project. May be interchanged with application or software system, but technically distinct.
- There are many pitfalls with inconsistent terminology throughout the profession. Seek clarification when you find ambiguity in course content.

Recommended References

- On Computable Numbers, with an Application to the Entscheidungsproblem. Alan Turing. Nov 1936. Proceedings of The London Mathematical Society.
- Turing's Cathedral. George Dyson. Dec 2012. Vintage Books.
- *Reading for next lecture: Pressman Ch 2-4, emphasis on Ch 4.*

Additional References

- What is Software? Definition, Types, and Examples. Linda Rosencrance. March 2021. TechTarget.
- Apollo Guidance Computer Explained: Everything You Need To Know. Dec 2021. History Computer.
- Framework for Evolving Software Product Line. Sami Ouali et al. May 2011. International Journal of Software Engineering & Applications.
- How Much Time Do Developers Spend Actually Writing Code? Chris Grams. Oct 2019. The New Stack.
- How Software Developers Really Spend Their Time. Lauren Orsini. Apr 2013. ReadWrite.
- Compiling. Randall Munroe. Aug 2007. xkcd.
- Software Engineering Terminology. Dec 2005. Free University of Brussels Software Languages Lab.