Announcements

- Last Lecture Today
- Project 2 Report (GitHub page link) **due November 29** at 11:59 pm
- Share your private GitHub repository with the instruction team for “Project Codes Deliverable”.
  - Nimisha-Roy (nimisha.roy9@gmail.com); tuanhuynh18 (minhtuan3396@gmail.com); tanujbohra (tanuj.bohra97@gmail.com)
- Next Class we will have presentation of 5 teams randomly. **MAKE SURE TO HAVE A DEMO**. 14–15 minute presentation explaining your software.ALL TEAMS SHOULD SUBMIT THEIR PRESENTATION BY NOV 29 9 AM EST.
- A **final extra-credit opportunity** for upto **2% of the final grade**- in the form of an **online quiz**
  - Available between Dec 1 and Dec 4 on Canvas. Comprises MCQ type questions.
  - A set of 15 questions with a 10-minute time limit
  - Can be attempted ONLY ONCE, so have a stable internet connection before attempting the quiz. Its open book/computer/notes

CS3300 Introduction to Software Engineering

Lecture 21: No Silver Bullet (1986)

Nimisha Roy ▸ nroy9@gatech.edu

Slides adapted from Mahdi Roozbahani

No Silver Bullet - Essence and Accident in Software Engineering

Widely Discussed article written by Turing Award winner **Fred Brooks** in 1987 discussing “Why is SE so hard?”

Reference: http://worrydream.com/refs/Brooks-NoSilverBullet.pdf

No Silver Bullet - Essence and Accident in Software Engineering

“There is no single development, in either technology or management technique, which by itself promises even one order-of-magnitude improvement within a decade in productivity, in reliability, in simplicity ”

No Silver Bullet - Essence and Accident in Software Engineering

Silver Bullet: A single technique or technology that by itself can deliver one order-of magnitude improvement to some aspect of software development

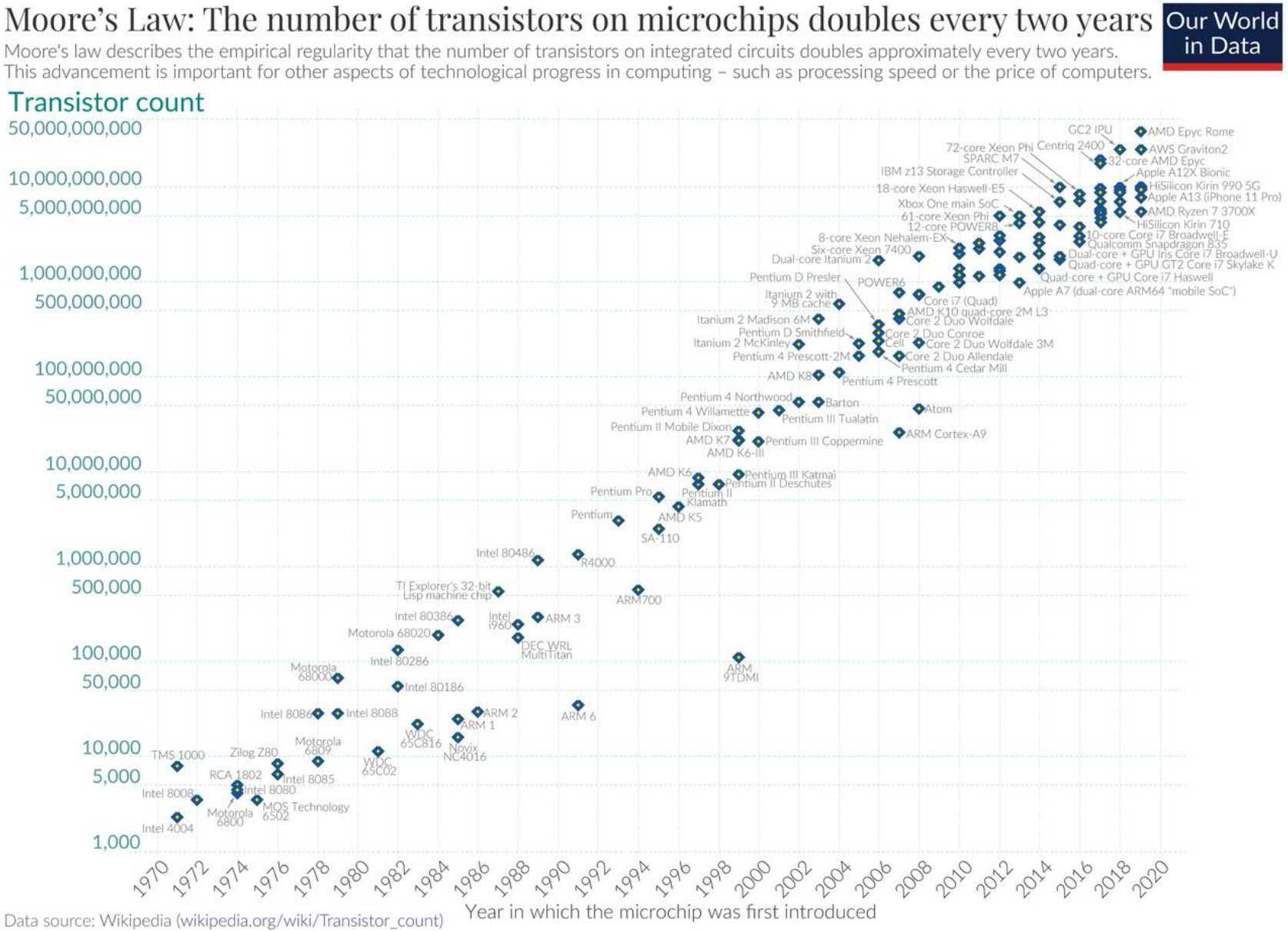
One order of magnitude means 10 times

The phrase typically appears with an expectation that some new technology or practice will easily cure a major prevailing problem

No Silver Bullet - Essence and Accident in Software Engineering

“We cannot expect ever to see two-fold gains every two years in software development, as there is in hardware development” (Moore's law)

No Silver Bullet - Moore's Law



- Started with an observation by Moore (co-founder of Intel) that number of transistors on microchips doubles every 2 years.
- Now describes a driving force of **technological and social change, productivity, and economic growth**. (hard drive cost per GB, cost per base of DNA sequencing…..)

No Silver Bullet - Comparison with Hardware

- The anomaly is not that software progress is so slow, but that computer hardware progress is so fast
- No other technology since civilization began has seen six orders of magnitude price-performance gain in 30 years [written in 1986]
- In no other technology can one choose to take the gain in *either* improved performance *or* in reduced costs.

No Silver Bullet - Why?

- Brooks divides the problems facing Software Engineering into 2 categories
  - **Essence**: Difficulties inherent, or intrinsic, in the nature of the software
  - **Accidents**: Difficulties related to the production of software
- Brooks argues that most techniques attack the accidents of software engineering, whereas essence constitute 90% of the problems.

Essential and Accidental Complexity

Essence:

Domain Complexity

- Accounting Software – the complexity involved in accounting by nature
- Rocketry managing software – complex functionality related to the functioning of a rocket

Accident:

Implementation Complexity

- Bugs
- Constructs that don’t exactly fit the way the domain wants them to/ Corner cases
- Threads, complexity of using AJAX requests

Why addressing “essential” problems are so difficult?

Like physical hardware limits (e.g., speed of light, heat dissipation), there are SE problems that will never be solved

4 Issues of essential difficulty:

1. Complexity
2. Conformity
3. Changeability
4. Invisibility

Complexity

- SW is far more complex for their size than any other human construct because no 2 parts are alike
- SW is far more complex than HW (computers, buildings, or automobiles, repeated elements abound)
  - 16-bit word in HW -> 216 states
  - State of SW is virtually infinite => problems verifying it

Complexity (cont'd)

- Scaling up not merely a repetition of the same elements in larger sizes, but an increase in the number of *different* elements

- Elements interact with each other in some non-linear fashion, so the complexity of the whole increases more than linearly

Complexity (cont'd)

- Complexity is an essential property: descriptions of a software entity that abstract away its complexity often abstract away its essence

- For three centuries, mathematics and the physical sciences made great strides by constructing simplified models of complex phenomena, deriving properties from the models, and verifying those properties by experiment. This paradigm worked because the complexities ignored in the models were not the essential properties of the phenomena. It does not work when the complexities are the essence.

Complexity (cont'd)

- Increased complexity => increased communication difficulty => SW flaws, schedule delays, costs, …
- Complexity => difficulty of enumerating and understanding, all possible SW states => unreliability
- Complexity of functions => difficulty of invoking functions => SW hard to use
- Complexity of structure => increased difficulty in adding functionality without introducing side effects
- Complexity of structure => unvisualized states that constitute reliability/security trapdoors
- …

Complexity (cont'd)

- Management, communication, and personnel turnover exacerbate these problems:
  - Difficult to overview, understand whole product, impeding conceptual integrity
  - How can you estimate without understanding?
  - How can you maintain without understanding?

Conformity

- Physics deals with terribly complex objects even at the "fundamental" particle level. The physicist labors on, however, in a firm faith that there are unifying principles to be found. Einstein argued that there must be simplified explanations of nature, because God is not capricious or arbitrary. No such faith comforts the software engineer.
- To the many (arbitrary) human institutions and systems to which it interfaces, SW is embedded in a mix of applications, users, laws, and machines. Conceived as most conformable
- Consider designing a software system to support an existing business process when a new VP arrives at the company. The VP decides to “make a mark” on the company and changes the business process. Our system must now conform to the (from our perspective) arbitrary changes imposed by the VP

Conformity (cont'd)

- Other instances of conformity
  - Adapting to a pre-existing environment
    - Such as integrating with legacy systems
    - And if the environment changes (for whatever reason), you can bet that the software will be asked to change in response
  - Implementing regulations or rules that may change from year to year
  - Dealing with a change in vendor imposed by your customer
- **Main Point: It is almost impossible to plan for arbitrary change;**
  - Instead, you just have to wait for it to occur and deal with it when it happens

Changeability

- Software is constantly asked to change
  - Other things are too, however, manufactured things are rarely changed after they have been created
  - Instead, changes appear in later models
    - Automobiles are recalled only infrequently
    - Buildings are expensive to remodel

Changeability (cont'd)

- Pressure to change is greater
  - Reality changes
  - Useful SW will encourage new requests
  - Long lifetime (~15 yrs) vs. HW (~4 yrs)
  - SW changes viewed as "free"
- Contrast with tangible domains
  - Imagine asking for a new layout of a house after the foundation has been poured.
  - Buildings can be changed. But change understood by all to be time-consuming and expensive (and messy)

Invisibility

- Software is, by its nature, invisible and intangible; it is difficult to design graphical displays of software that convey meaning to developers
  - Contrast to blueprints: here geometry can be used to identify problems and help optimize the use of space
- But with software, its difficult to reduce it to diagrams
  - UML contains 13 different diagram types (!)
    - to model class structure, object relationships, activities, event handling, software architecture, deployment, packages, etc.
  - The notations of the different types almost never appear in the same diagram
    - they really do document 13 different aspects of the software system!

Invisibility (cont'd)

- Hard to get both a "big picture" view as well as details
  - Hard to convey just one issue on a single diagram
  - instead multiple concerns crowd and/or clutter the diagram hindering understanding
- This lack of visualization deprives the engineer from using the brain's powerful visual skills

Essence or Accident?

[**E**] A bug in a financial system is discovered that came from a conflict in state/federal regulations on one type of transaction

[**A**] A program developed in two weeks using a whiz bang new application framework is unable to handle multiple threads since the framework is not thread safe

[**A**] A new version of a compiler generates code that crashes on 32-bit architectures; the previous version did not

[**E**] A fickle customer submits 10 change requests per week after receiving the first usable version of a software system

Past Breakthroughs Solved Accidental Difficulties (from Brooks)

- **High-level languages** (vs. bits, registers, conditions, branches)
  - enhance/ease representation, improve vocabulary and how to think about problems

- **Time-sharing** (vs. batch programming)
  - provided quick turnaround benefits, immediacy, less context switch

- **Unified programming environments**
  - help us better manage conceptual constructs, use programs together, but not figure out what they should be

=> **All addressed accident, not essence difficulties**

Hopes for the Silver [in 1986]

- Several "hopes for the silver", but they only made small improvements (not 10x)
- **Ada**
  - In the end, "just" a high-level language
  - Good for retraining programmers in modern design and modularization techniques
  - Does not eliminate essential complexity
- **OOP**
  - *Abstract data types* (information hiding)
  - *Hierarchical interfaces and refined subtypes (*more information*)*
  - Higher order of accidental difficulty removed by both. Same as above.
- **PC's increasing power**
  - Speeds up machine-bound activities, but does not simplify the tasks

Hopes for the Silver [in 2012]

- **AI** - two meanings
  - Use of computers to solve problems previously solved by humans (not really AI)
  - Expert systems (most advanced AI)
- **Expert Systems**
  - A computer system emulating the decision-making ability of a human expert. Designed to solve complex problems by reasoning through bodies of knowledge, represented mainly as if–then rules rather than through conventional procedural code. Application independent
  - Could help by suggesting designs, testing strategies, etc.
  - Needs good knowledge base
  - May help novice programmers benefit from the accumulated wisdom of experts
- **Automatic programming**
  - No black magic, but somehow successful today
  - Examples?

  Domain Specific Languages; Program Generator

Hopes for the Silver (cont'd)

- **Graphical programming**
  - Helpful, but SW is hard to visualize
  - Lots of research in this area. New interesting articles published.
- **Program verification**
  - Great, but does not mean error free (e.g., errors in proofs or specifications)
  - Plus, it is complex
- **Environments and tools**
  - Hierarchical file systems, uniform file formats, generalized tools, language-based editors, integrated database systems to track details, …
  - Very useful, but still attacking the accidental complexity
- **Must attack essential complexity, which is 90% of SW**

Promising Attacks on the Essence

- **Buy vs. build**
  - Hard part is requirements, specifications and design, not implementation; amount and quality of off-the-shelf SW is increasing
- **Iterative requirements refinement and rapid prototyping**
  - Clients do not know what they want
- **Incremental development**
  - Grow, not build
- **Cultivate great designers** (focus on *people*)
  - Better SE training
  - Career mentors and career development paths
- **Did they work?**

A New Silver Bullet?

- The World Wide Web? – some claim productivity increased 10-fold due to WWW.
- Automated testing?- some claim most regression errors are avoided.
- Open-source development?
- Agile Software Development paired with Git.
- Something else?