Department of Computer Science

CS411 SE Defined

@perrydBUCS

What is Software?

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) documentation that describes the operation and use of the programs.

What is Software?

- *Software is developed or engineered, it is not manufactured in the classical sense.*
- *Software doesn't "wear out" in the sense that material things do, but it can become obsolete.*
- *Although the industry is moving toward component-based construction, most software continues to be built by hand, one program at a time.*

Everyone wants to use software

- In the modern world, we are obsessed with software, and everyone always wants to be running the latest version
- True or false?

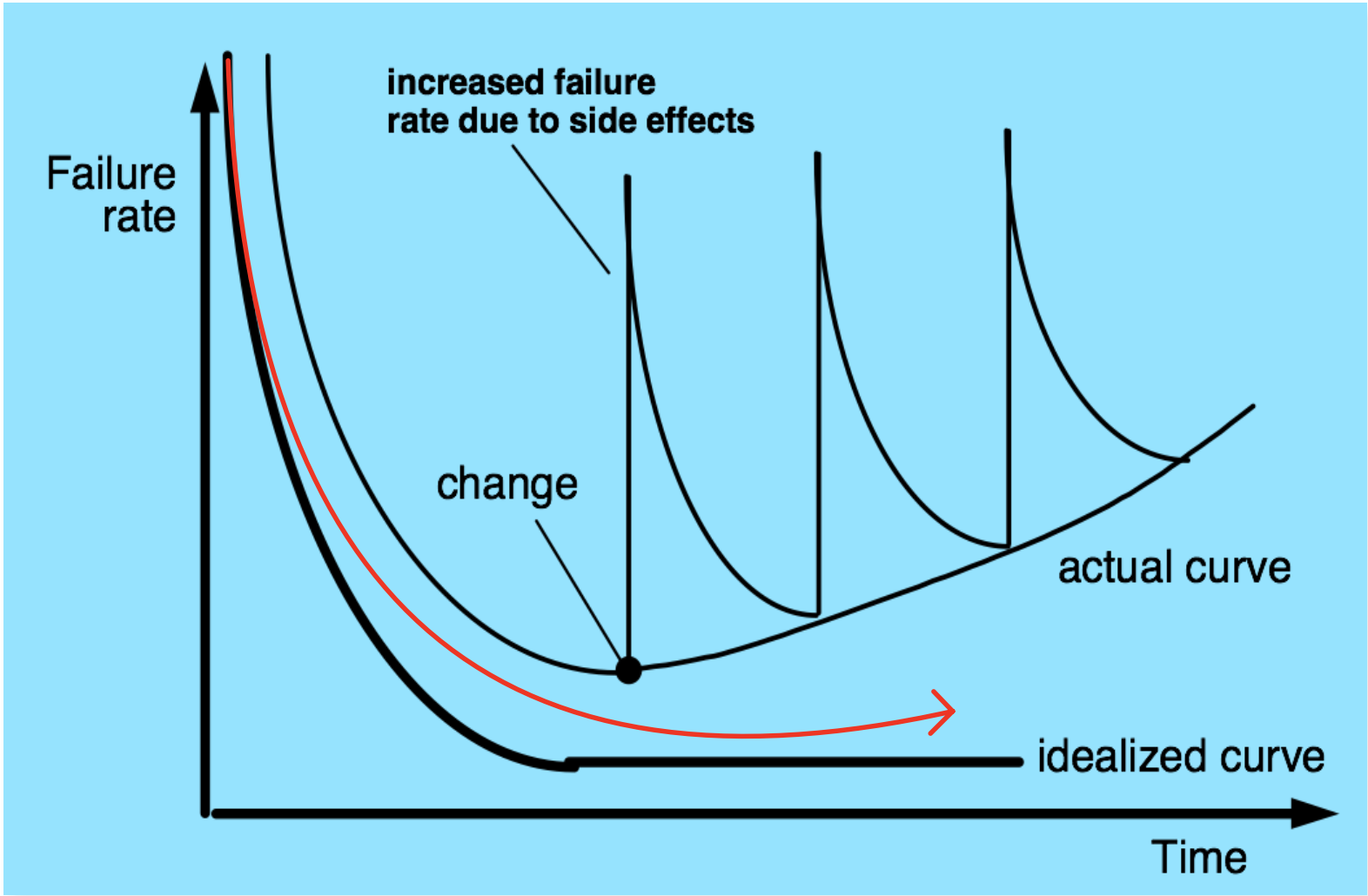
NOBODY wants to use software

- The reality is that we just want to DO things, not run software
- From an engineering standpoint this means that the software must be robust enough to fade into the background
- This is the classic feature-vs-benefit problem
- Compare the approaches of
  - Sony
  - Apple

Software Engineer

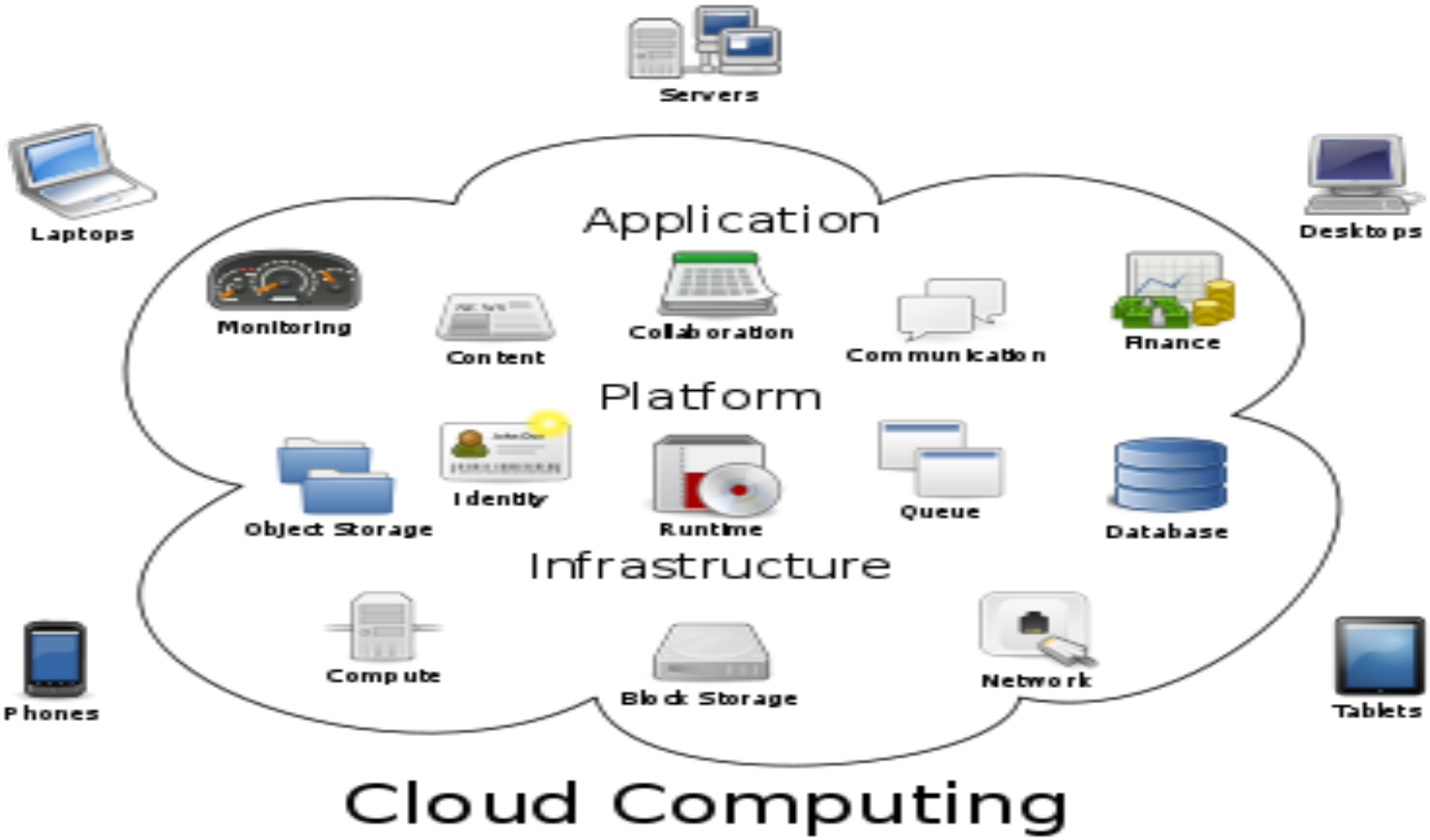
Ux designer

Wear vs. Deterioration



Cloud Computing

Cloud : OTher people's Computers



Software Engineering

- Some goals:
  - *a concerted effort should be made to understand the problem before a software solution is developed*
  - *design becomes a pivotal activity*
  - *software should exhibit high quality*
  - *software should be maintainable* – ilities
- The seminal definition:
  - *[Software engineering is] the establishment and use of sound engineering principles in order to obtain software that is reliable and works efficiently on real machines.*

Software Engineering

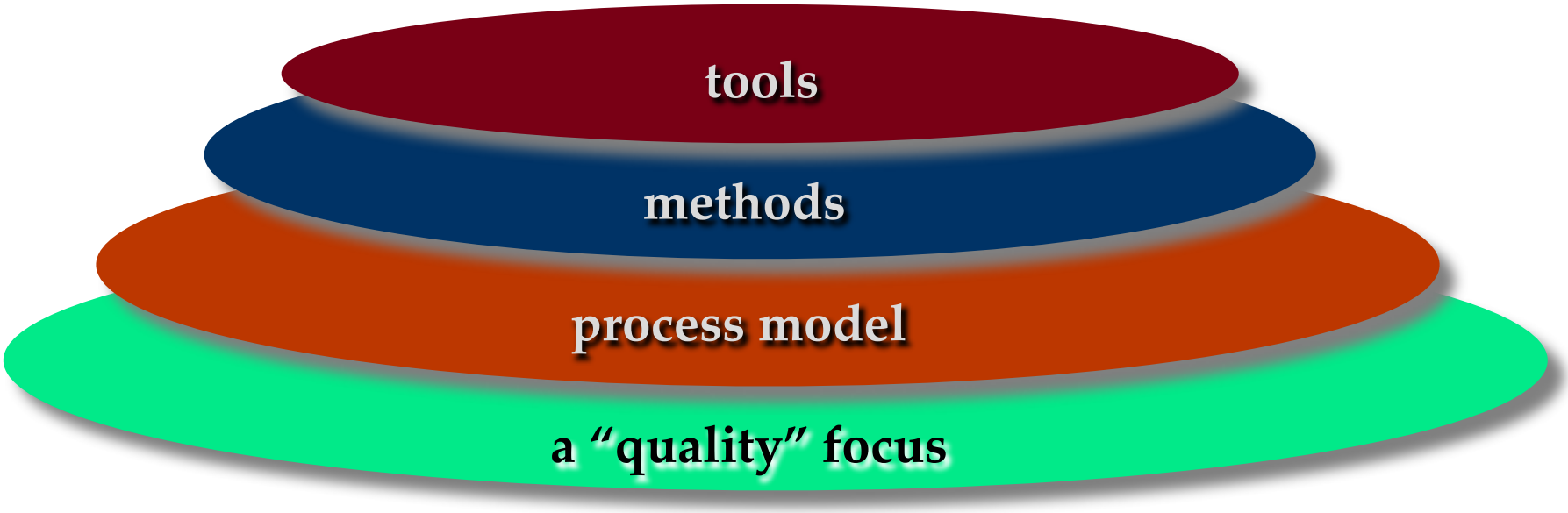
- The IEEE definition:
  - *Software Engineering:*
  - *(1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.*
  - *(2) The study of approaches as in (1).*

What is Software Engineering?

Software engineering encompasses at least three things:

- *A set of **processes***
- *A set of **methods** to implement the processes*
- *A set of **tools** to facilitate the methods*

A Layered Technology



Software Engineering

Quality

- Every text on software engineering claims that ‘quality’ is a primary goal.
- What does the word ‘quality’ mean?
- Example: Ford Motor Company Touch screen
- Question: Is quality destroying the economy?

JD

Software isn't the goal

Not a goal

- Writing software isn't our core activity
- Really we're solving business problems
- Writing software -- coding -- is a skill
- Software engineering is a discipline
- The code is just an expression of our design and decisions

Engineering part

A Process Framework

- A *process framework* is a set of guidelines, work products, and tools that attempt to facilitate a process
- For software engineering in general, the framework comprises
  - Communication
  - Planning
  - Modeling
  - Construction
  - Deployment

Umbrella Activities

- These are applied across all phases of the product
  - Software project tracking and control
  - Risk management
  - Software quality assurance
  - Technical reviews
  - Measurement
  - Software configuration management
  - Reusability management
  - Work product preparation and production

Adapting a Process Model Involves…

- Determining an overall flow of work
- Identifying work products
- Establishing norms for consistency (naming conventions, etc)
- Applying tracking and other management oversight
- Defining roles and responsibilities
- Documenting and disseminating the process

All of that boils down to…

- Polya (*How to Solve It*) suggests:

  1. Define *Understand the problem* (communication and analysis).
  2. Design *Plan a solution* (modeling and software design).
  3. Develop *Carry out the plan* (code generation).
  4. Deliver *Examine the result for accuracy* (testing and quality assurance).

  \+
  manage / maintain

40 process

Polya 1: Understand the Problem

Define

- *Who has a stake in the solution to the problem?* That is, who are the stakeholders?
- *What are the unknowns?* What data, functions, and features are required to properly solve the problem?
- *Can the problem be compartmentalized?* Is it possible to represent smaller problems that may be easier to understand?
- *Can the problem be represented graphically?* Can an analysis model be created?

Polya 2: Plan the Solution

Design

- *Have you seen similar problems before?* Are there patterns that are recognizable in a potential solution? Is there existing software that implements the data, functions, and features that are required?
- *Has a similar problem been solved?* If so, are elements of the solution reusable?
- *Can subproblems be defined?* If so, are solutions readily apparent for the subproblems?
- *Can you represent a solution in a manner that leads to effective implementation?* Can a design model be created?

Polya 3: Carry Out the Plan

Develop

- *Does the solution conform to the plan?* Is source code traceable to the design model?
- *Is each component part of the solution provably correct?* Has the design and code been reviewed, or better, have correctness proofs been applied to algorithm?
- *Is there a process in place to facilitate development?*

Polya 4: Examine the Result

Deploy

- *Is it possible to test each component part of the solution?* Has a reasonable testing strategy been implemented?
- *Does the solution produce results that conform to the data, functions, and features that are required?* Has the software been validated against all stakeholder requirements?
- *What lessons can be learned?*

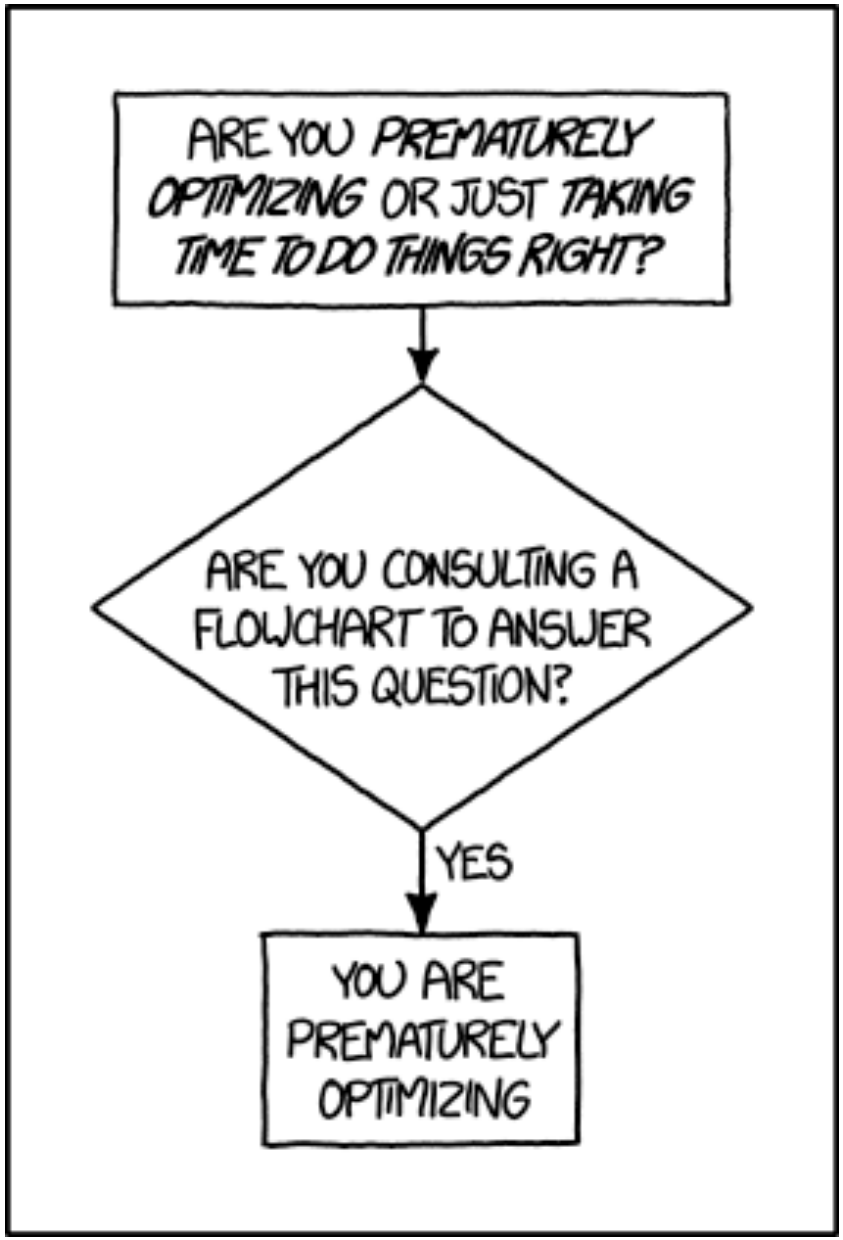
Hooker's General Principles

- David Hooker's seven principles focus on software engineering as a whole:
- **1: The Reason It All Exists**
  - *To provide value to users*
- **2: KISS (Keep It Simple, Stupid!)**
  - *Designs should be as simple as possible but no simpler*
- **3: Maintain the Vision**
  - *There should be a consistent, overlying architecture*

- **4: What You Produce, Others Will Consume**
  - *Those 'others' must be able to understand how to use your product*
- **5: Be Open to the Future**
  - *Never design for obsolescence…plan for expansion and change*
- **6: Plan Ahead for Reuse**
  - *Create software building blocks that can be combined to create new products*
- **7: Think!**

And one from Knuth via Randall Munroe

- https://xkcd.com/1691/



Advice from Chris Date

- What, not how.

Software Myths

- Affect managers, customers (and other non-technical stakeholders) and practitioners
- Are believable because they often have elements of truth,

but …

- Invariably lead to bad decisions,

therefore …

- Insist on reality as you navigate your way through software engineering

Common myths

- We've done it this way for years. We can succeed by not changing things.
- If we get behind schedule, we can just add more resources
- We can just outsource development and let them figure things out.
- We can fill in the details later.
- It's easy to change software to meet needs we forgot to articulate up front
- The only deliverable is the running software
- Software engineering will just slow the project down

How It all Starts

- Every software project is precipitated by some business need —
  - the need to correct a defect in an existing application;
  - the need to the need to adapt a 'legacy system' to a changing business environment;
  - the need to extend the functions and features of an existing application, or
  - the need to create a new product, service, or system.

Software isn't the goal

- One thing to keep in mind is that, for most companies, software projects are only a tool used to increase profits
- As software developers we sometimes lose sight of that
- Understanding how projects fit into a company's strategic goals is important in order to gain some context
- In this context, failure or success of a project has nearly nothing to do whether the code is correct or passes all of its test

Bottom line

- Software is a product designed and built
- Typically a professional engineer works on software
- Benefits are more important than features
- It doesn't make sense for every software product to be created using its own process
- What we really want is a single process that we can repeat over and over for each project
- What does this buy us?