
[CS3704] Intermediate Software Design and Engineering

Dr. Chris Brown

Virginia Tech

8/23/2023

Introduction (cont.)

What is software and its engineering?

SE History

SE Processes Intro

Software Development Lifecycle

Learning Outcomes

By the end of the course, students should be able to:

- **Understand software engineering processes, methods, and tools used in the software development life cycle (SDLC)**
- Use techniques and processes to create and analyze requirements for an application
- Use techniques and processes to design a software system
- Identify processes, methods, and tools related to phases of the SDLC
- Explain the differences between software engineering processes
- Discuss research questions and current topics related to software engineering
- Create and communicate about the requirements and design of a software application

Announcements


HW0 due Friday at 11:59pm!

- Slack introductions
- Syllabus/Course review questions

Syllabus Review Questions so far...

- Exam date typo March 23 ☐ November 3
- Incorrect Fall Break date in schedule Oct 9 ☐ Oct 6
- Structure of workshops Varies based on the workshop
- Certain amount of days before missing participation points? How will attendance be recorded? Based on percentage of classes attended, participation in class activities
- Will you provide enough Ut Prosim events? Yes
- Will the slides be available? Yes, on Canvas and GitHub
- Do CS-related service opportunities for other classes count for Ut Prosim? Yes

Syllabus Review (cont.)

- How will project groups be formed? TBD
- How long, in terms of hours per team, does the project take on average? Depends on project, group,...
- Speaking louder in class Will try to speak louder or find 
- Can we have a group of 4 or do we need to have 6 people in our groups as per the syllabus? **Five*** for discussion presentations (was originally six, but due to error with fall break and students dropping there is space to add one more), 4 or 5 for project

What is software?

- Software is a set of instructions, data, or programs used to operate a computer and execute specific tasks. In simpler terms, software tells a computer how to function. [[1](#)]
- Software encompasses:
 - Executable programs
 - Data associated with these programs
 - Documents: user requirements, design documents, user/programmer guides, etc.

Why is software important?

Software is eating the world! [Andreessen, 2011]

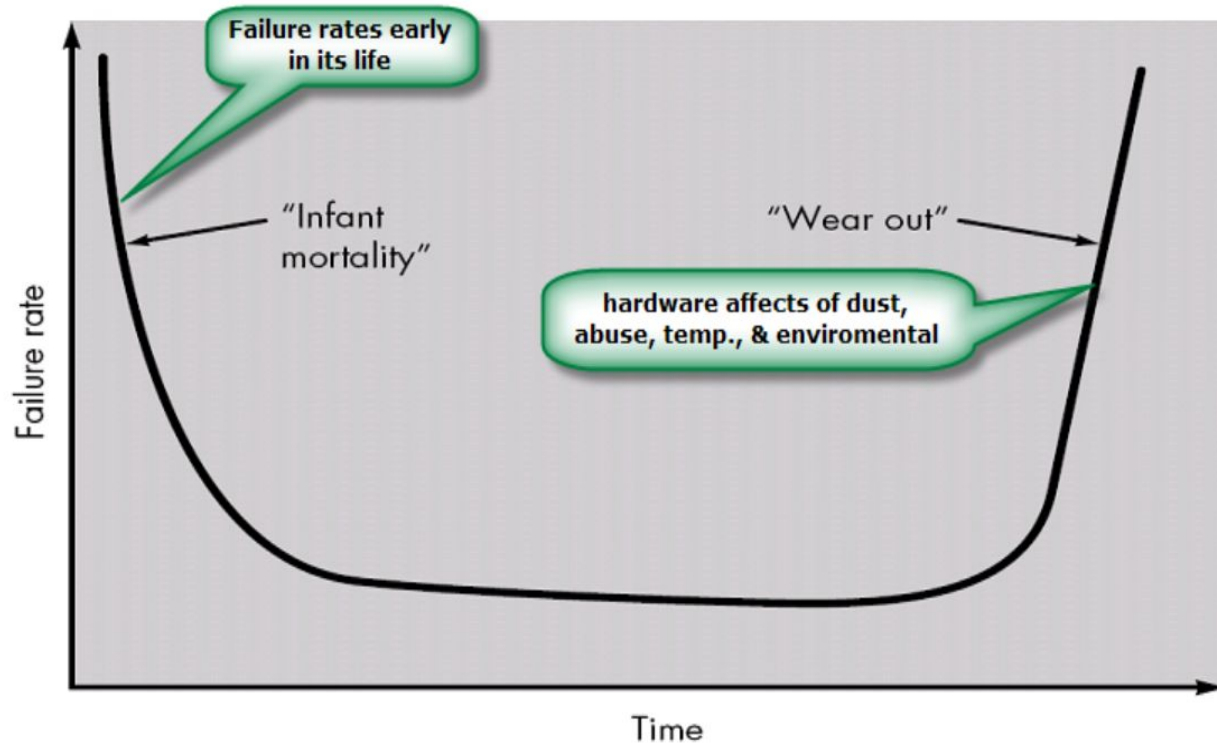
“More and more major businesses and industries are being run on software and delivered as online services...”



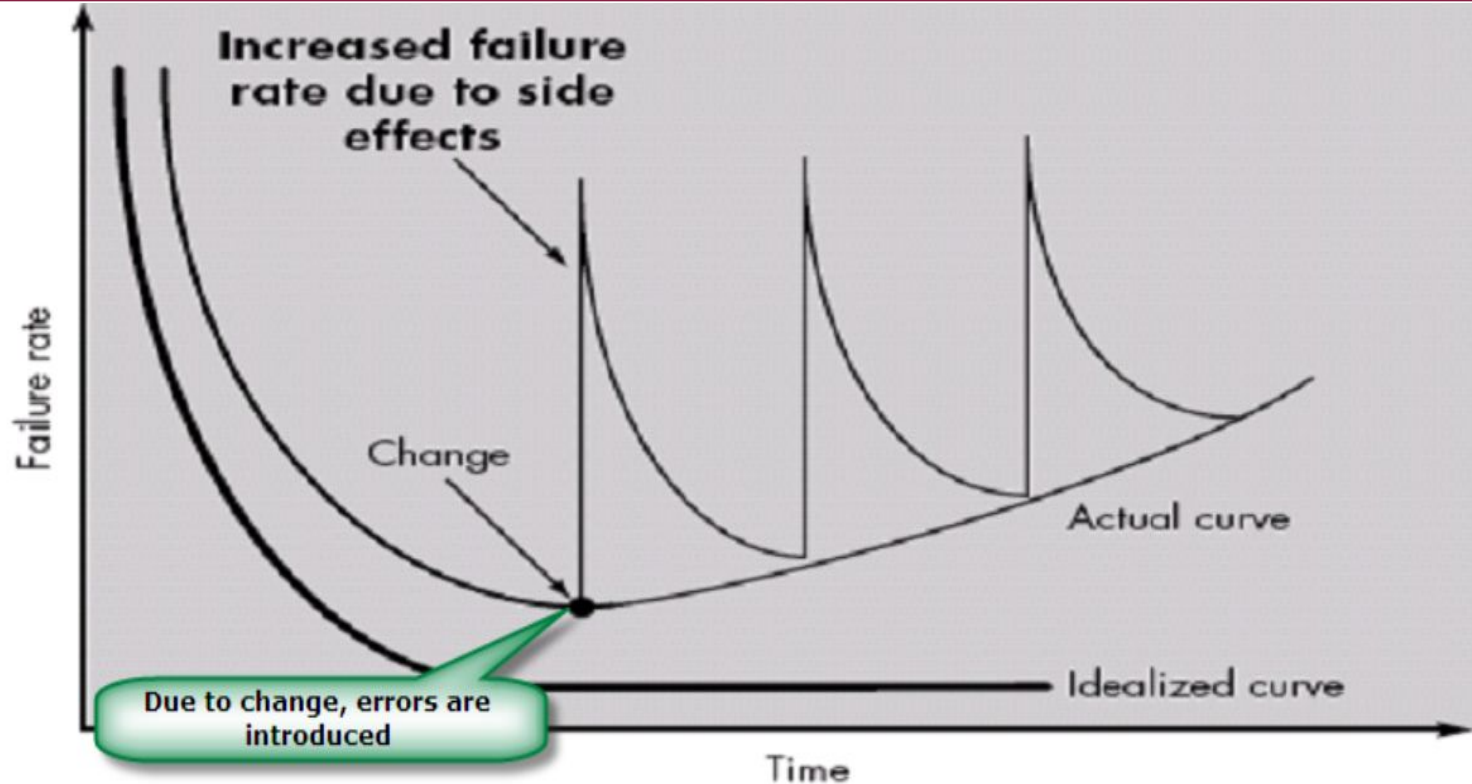
Warm-Up

Discuss: What are some of your favorite software applications? Why?

Hardware “wears out” over time



So does software!



Software Crisis

“The major cause of the software crisis is that the machines have become several orders of magnitude more powerful! To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem.” [Dijkstra]

Software Crisis

- Projects running over-budget
- Projects running over-time
- Software was very inefficient
- Software was of low quality
- Software often did not meet requirements
- Projects were unmanageable and code is difficult to maintain
- Software was never delivered

What is software engineering?

A discipline that encompasses:

- the ***process*** of software development;
- ***methods*** for software analysis, design, construction, testing, and maintenance; and
- ***tools*** that support the processes and the methods.

Software Engineering

“... a new subdiscipline, software engineering, has arisen. The development of a large piece of software is perceived as an engineering task, to be approached with the same care as the construction of a skyscraper, for example, and with the same attention to cost, reliability, and maintainability of the final product... Even with such an engineering discipline in place, the software-development process is expensive and time-consuming.”

Software Engineering?

**“ ‘It’s Engineering... but not as we know it’...
Software Engineering - solution to the
software crisis, or part of the problem? ”**

Processes, Methods, and Tools

- Various tasks required to build and maintain software
 - e.g. design, testing, etc.
- *SE process*: the organization and management of these tasks
 - various process models
- *SE methods*: ways to perform the tasks
- *SE tools*: assist in perform the tasks
 - UML tools, IDEs, issue tracking tools

SE History

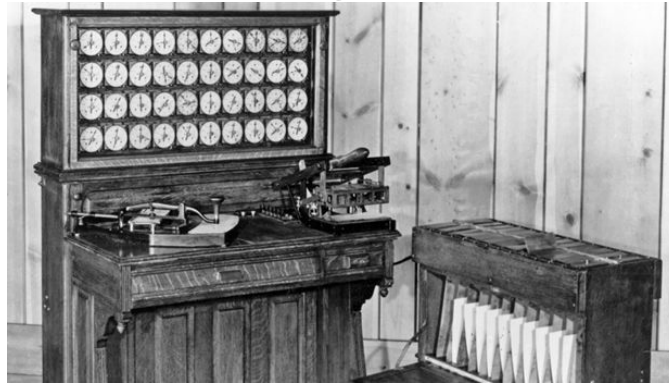
- Why study software engineering history?
 - To learn what has been done before and how we got here.
 - CS/SE is not that old, compared to other disciplines.
- Don't remember failures? Likely to repeat them
- Don't remember successes? Unlikely to repeat them

“Those who cannot remember the past are condemned to repeat it.”

– Santayana [<https://liberalarts.vt.edu/magazine/2017/history-repeating.html>]

History of Computer Programming

- **1843:** Sequence of steps from Ada Lovelace to Charles Babbage considered first computer program
- **1889:** Hollerith tabulating machine



- **1940s:** First electronic computers
- **1956:** First programming language - FORTRAN

History of Software Engineering


1950s: Engineer software like hardware

- First programming languages
 - **1956:** FORTRAN (Formula Translation);
 - **1958:** LISP (List Processor);
 - **1959:** COBOL (Common Business Oriented Language);
- Punch cards



History of SE (cont.)

1960s: Software is NOT like hardware

- Different properties:
 - invisible, complex, difficult to change, unconstrained by physical laws of nature,...
- Demand for programmers exceeded supply
- First college Computer Science departments formed 
 - **1962:** Purdue University
- First use of the term “***software engineering***”!

“Software Engineering”

1963/1964: “While developing the guidance and navigations systems for the Apollo missions, computer scientist and systems engineer Margaret Hamilton coins the term ‘*software engineering*’. Hamilton felt that software developers earned the right to be called engineers.” [Juhasz]



History of SE (cont.)

1960s (cont.): Software is NOT like hardware

- Still disorganized, but with better infrastructure
 - Operating systems, compilers, utilities, etc.
- Some successes:
 - Apollo
 - Electronic switching systems (ESS)
- Problems:
 - Unmaintainable spaghetti code
 - Unreliable, undiagnosable systems
 - Software development is too expensive

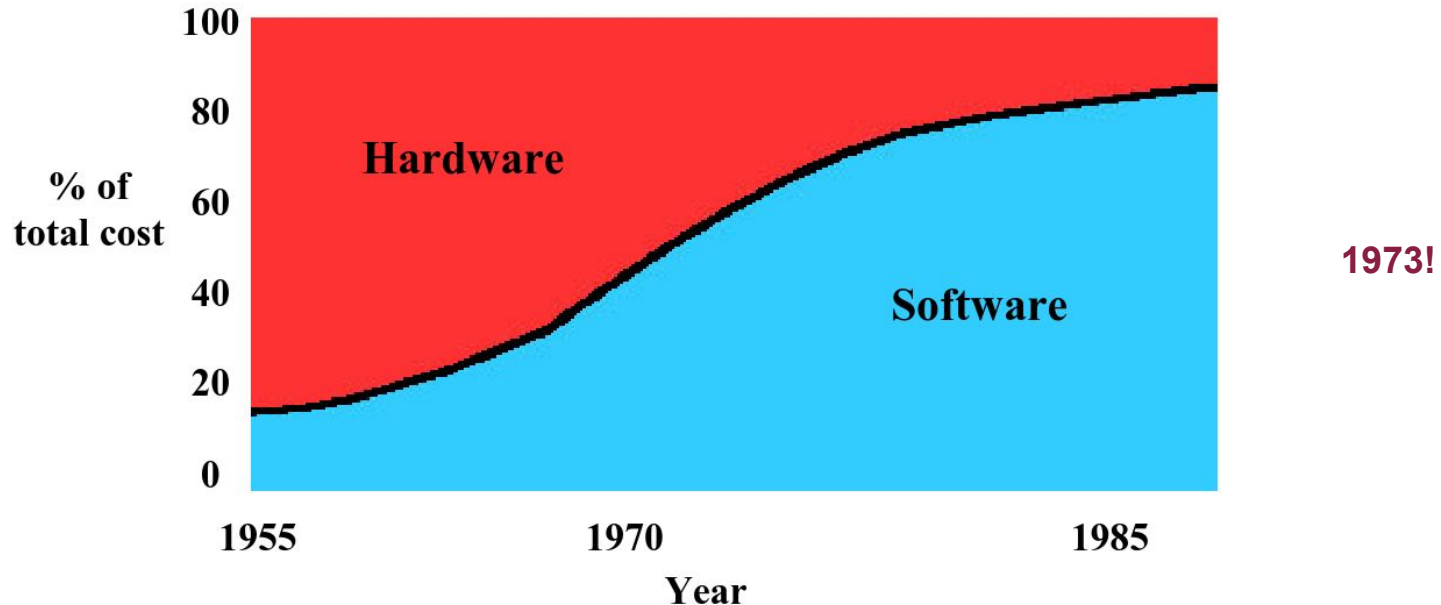
History of SE (cont.)

1970s: Formal Approaches and Waterfall

- Structural programming processes
- Formal methods begin to form
 - Specification
 - Development
 - Verification
- Punch cards starting to become obsolete...
- **Plan-Driven process models**

History of SE (cont.)

1970s: Software costs surpass hardware



History of SE (cont.)

1980s: More Productivity, Less Plan-Driven

- Major productivity enhancements
 - Working faster: tools and environments
 - Working smarter: processes and methods
 - Work avoidance: code reuse, simplicity, objects
- **Iterative process models**
- Other trends: Iterations, Code reuse libraries, codes of ethics, licenses, Object oriented...
 - Smalltalk, Eiffel, Ada, C++

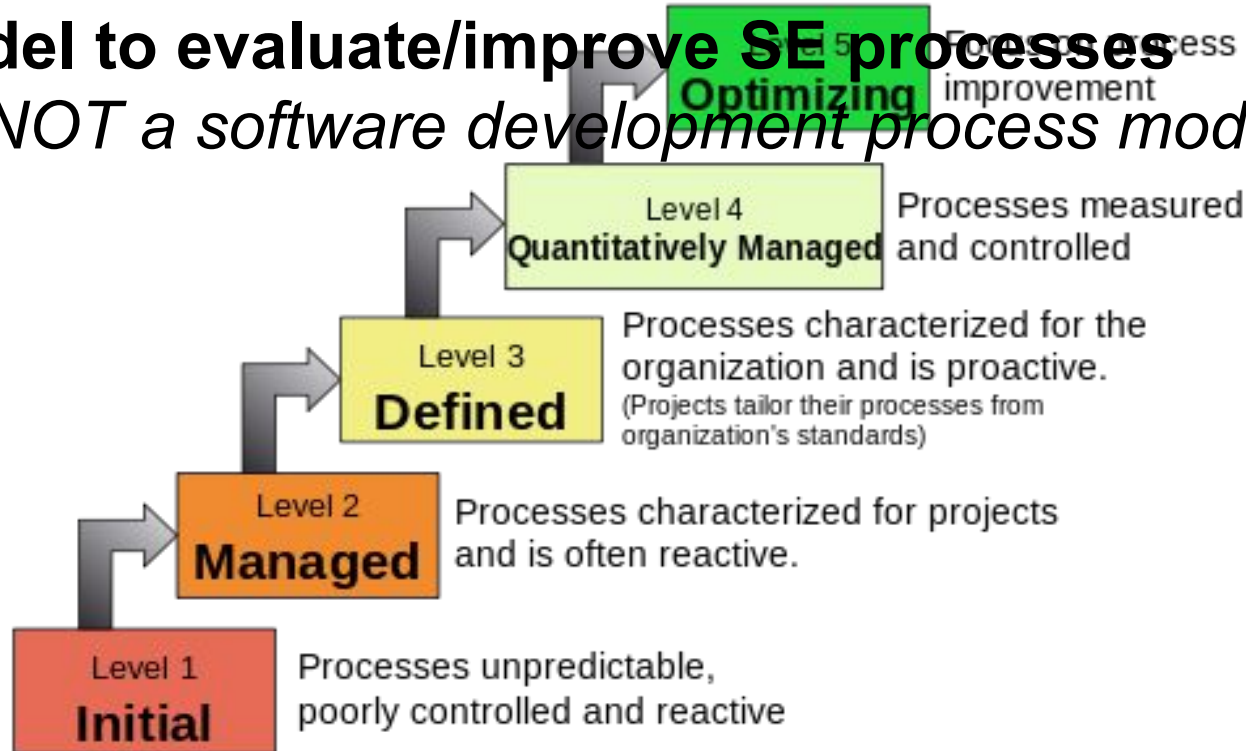
History of SE (cont.)

1990s: CMM and Iterative Models

- Capacity Maturity Models (CMM)
- Growing divide between *plan-driven* and *iterative* models (Standish Group)
- Other trends: Open source software, reverse engineering, computer viruses, etc.
- Modern programming languages
 - Java, Python, JavaScript, Ruby, etc.
 - End-user programming: HTML, CSS, R, etc.

Capacity Maturity Model

- **Model to evaluate/improve SE processes**
 - *NOT a software development process model!*



Data by the Standish Group (1995)

- \$81B on canceled software projects
- \$59B for budget overruns
- Only 1/6 projects were completed on time and within budget
- Nearly 1/3 projects were canceled
- Over half projects were considered “challenged”
- Among canceled and challenged projects
 - Budget overrun: 189% of original estimate
 - Time overrun: 222% of original estimate
 - Only 61% of the originally specified features

The CHAOS Report 1995

“In the United States, we spend more than \$250 billion each year on IT application development...A great many of these projects will fail. Software development projects are in chaos, and we can no longer imitate the three monkeys -- hear no failures, see no failures, speak no failures. The Standish Group research shows a staggering 31.1% of projects will be canceled before they ever get completed. Further results indicate 52.7% of projects will cost 189% of their original estimates. The cost of these failures and overruns are just the tip of the proverbial iceberg.”

[Standish Group]

History of SE (cont.)

2000s-present: Process Synthesis

- **2001:** Agile manifesto
- **2004:** Software Engineering Body of Knowledge ([SWEBOK](#))
- Model, feature, and test-driven development
- Service-oriented software
- Hybrid agile/plan-driven processes
- **Discuss with a partner: Anything else???**

[Review] What is SE?

A discipline that encompasses:

- the *process* of software development
- *methods* for software analysis, design, construction, testing, and maintenance
- *tools* that support the process and the methods

What is a SE process?

- a framework for the tasks that are required to build high-quality software to provide stability, control and organization to an otherwise chaotic activity

[Pressman]

- a software development process defines **who** does **what**, **when**, in order to build a piece of software.

[Wilson]

SE Myths

Management

- “If we get behind schedule, we can just add more people and catch up”
- **Fact:** Adding more people to a late project will make it later
 - The people working now must spend time educating the newcomers


SE Myths (cont.)

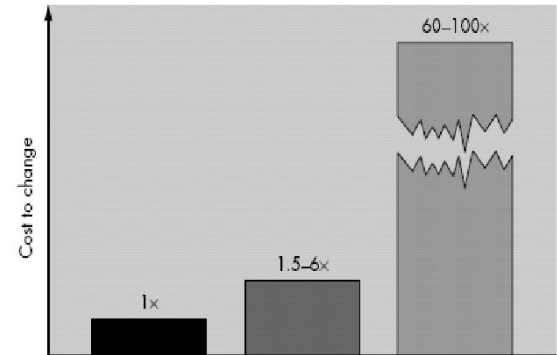
Customers/Clients

- “A general statement of objectives is enough to start programming”
- **Fact:** An ambiguous statement of objectives leads to project failures
 - Unambiguous requirements need effective and continuous communication between customer and developer

SE Myths (cont.)

Customers/Clients

- “Changes in requirements are easy to deal with because software is flexible”
- **Fact:** Changes are harder and more expensive over a project 



SE Myths (cont.)

Practitioners

- “Once we get the program running, we are done”
- **Fact:** 60-80% effort comes after the software is delivered for the first time
 - Bug fixes, feature enhancements, software reengineering, migration

SE Myths (cont.)

Practitioners

- “Until I get the program running, I cannot assess quality”
- **Fact:** Software assessment methods can be applied once code is written and are very effective

SE Myths (cont.)

Practitioners

- “The only deliverable work product is the running program”
- **Fact:** Need the entire configuration
 - Documentation of system requirements, design, programming, and usage

SE Myths (cont.)

Practitioners

- “SE will slow us down by requiring unnecessary documentation”
- **Fact:** SE is about creating quality
Better quality ☐ reduced rework ☐ faster delivery time

SE Myths (cont.)

Practitioners

- “I need to be an exceptional coder to be in software engineering.
- **Fact:** There are a variety of roles and skills needed to successfully develop and maintain software applications.
 - Managers, QA/testers, scrum masters, UI/UX designers, business analysts, IT, customer support, data analysts, database administrators (DBA), architects, security specialists, deployment engineers,...

Software Development Life Cycle

1. Requirements
2. Design
3. Implementation
4. Testing
5. Deployment/Maintenance

→ *As defined in this class. Specific terms to describe phases will vary based on company, team, process, etc., but basic ideas will apply.*

Requirements

Goal: Understand customer requirements for the software system

- The *what* of the project
- Very difficult to “get right” the first time and evolve over the course of development
 - Remember 2 of the Top 3 reasons for project failure:
(2) Incomplete and (3) Changing Requirements
- *Software Artifacts*: requirements documents, use cases, user stories,...

Goal: decide the software structure and enable programmers to implement requirements by designating projected parts of the implementation

- The *how* of the project
- design: a representation or model of the software to be built
- How individual classes and software components work together (Programs can have 1000s+ of classes/methods)
- *Software Artifacts*: design documents, class diagrams,...

Implementation



Goal: translating design into a concrete system (i.e. code)

- Can use any language, but some languages are better suited to certain types of programs than others
- *Software Artifacts*: source code, documentation, configuration files, media, executables, bug database, source code repository, issue trackers...

Testing



Goal: Execute software with intent of finding errors

- While you can't test until there is code to run, you can start planning testing when you're analyzing the requirements
- Includes Unit (Ut) and System (St) tests to verify code and functionality
- *Software Artifacts*: test code, bug database, test database, test inputs and outputs, documentation,...

Deployment/Maintenance



Goal: release, upgrade, and fix the software

- deployment: delivery of software to users
 - When software is completed, it must be **deployed** to customers for usage.
- *Just because you deliver your software doesn't mean you're done with it:* Software must be **maintained** such that user problems are addressed after operation (next version, debugging, increased testing, refactoring, updates to requirements, etc.)
- *Software Artifacts: All!*

The First Law

*“No matter where you are in the system life cycle, **the system will change, and the desire to change it will persist throughout the life cycle.**”*

[Bersoff, 1980]

Next Class

- **SE Basics Workshop**
 - Overview of software engineering topics
 - Activity will be led by GTAs Xiaoxiao and Hunter
- **HW0 due Friday at 11:59pm**

References

- “What is Software?” <<https://www.webopedia.com/definitions/software/>>
- RS Pressman, “Software engineering: a practitioner's approach”.
- Greg Wilson, “Building Software Together”.
- Edsger Dijkstra, “The Humble Programmer”. ACM
- Kevin Juhasz, “The history of coding and software engineering”. <<https://www.hackreactor.com/blog/the-history-of-coding-and-software-engineering>>
- An extract from the entry on software engineering, a subsection of the entry on Computer Science: Software, in the CDROM version of the Encyclopaedia Britannica, as quoted by: A Bryant, ACM.
- Computer Hope, “Computer Programming History”. <<https://www.computerhope.com/history/programming.html>>
- Barry Boehm, ICSE’06 Keynote
- Na Meng and Barbara Ryder