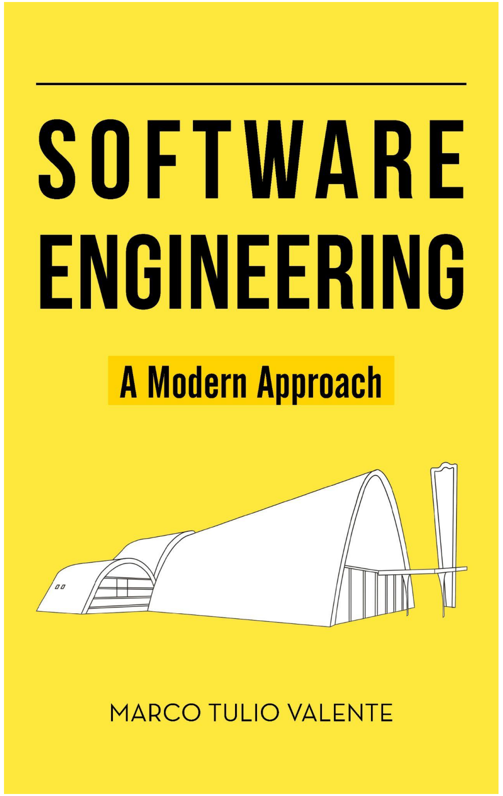


Chapter 1 - Introduction

Prof. Marco Tulio Valente

https://softengbook.org

NATO Conference (Germany, 1968)

- First time the term Software Engineering was used

Working Conference on Software Engineering

Comment from a participant

"Certain systems are presenting demands beyond our capabilities... We are having difficulties with large applications."

What is studied in SE?

SWEBOK®
V3.0
Guide to the Software Engineering Body of Knowledge

1. Requirements Engineering
2. Software Design
3. Software Construction
4. Software Testing
5. Software Maintenance
6. Configuration Management
7. Project Management

What is studied in SE?

SWEBOK® V3.0

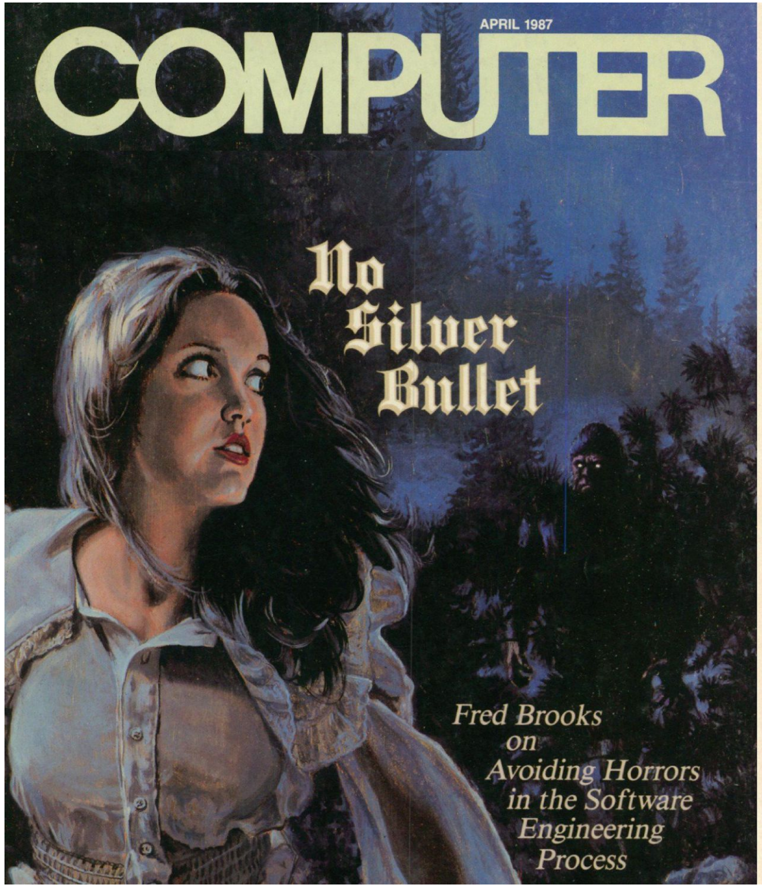
Guide to the Software Engineering Body of Knowledge

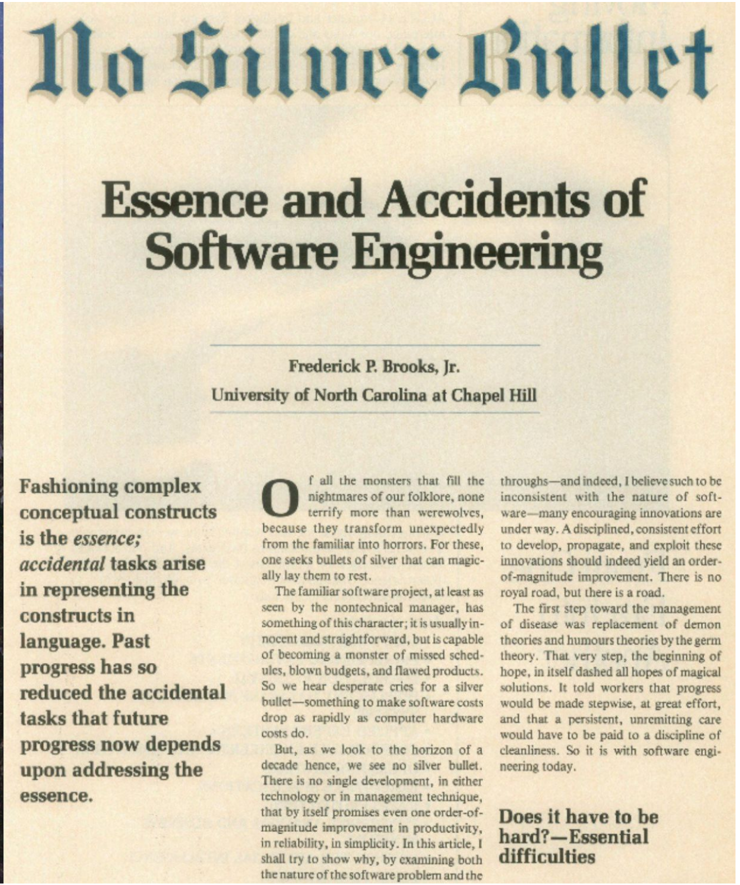
8. Software Processes
9. Software Models
10. Software Quality
11. Professional Practice
12. Economic Aspects

In this class

- We will give an overview of these areas
- To provide a broad understanding of what is SE
- In the rest of the course, we will study these topics in detail

But first a disclaimer


APRIL 1987
COMPUTER
No Silver Bullet
Fred Brooks on Avoiding Horrors in the Software Engineering Process



No Silver Bullet

Essence and Accidents of Software Engineering

Frederick P. Brooks, Jr.
University of North Carolina at Chapel Hill

Fashioning complex conceptual constructs is the *essence;* *accidental* tasks arise in representing the constructs in language. Past progress has so reduced the accidental tasks that future progress now depends upon addressing the essence.

Of all the monsters that fill the nightmares of our folklore, none terrify more than werewolves, because they transform unexpectedly from the familiar into horrors. For these, one seeks bullets of silver that can magically lay them to rest.

The familiar software project, at least as seen by the nontechnical manager, has something of this character; it is usually innocent and straightforward, but is capable of becoming a monster of missed schedules, blown budgets, and flawed products. So we hear desperate cries for a silver bullet—something to make software costs drop as rapidly as computer hardware costs do.

But, as we look to the horizon of a decade hence, we see no silver bullet. There is no single development, in either technology or in management technique, that by itself promises even one order-of-magnitude improvement in productivity, in reliability, in simplicity. In this article, I shall try to show why, by examining both the nature of the software problem and the

throughs—and indeed, I believe such to be inconsistent with the nature of software—many encouraging innovations are under way. A disciplined, consistent effort to develop, propagate, and exploit these innovations should indeed yield an order-of-magnitude improvement. There is no royal road, but there is a road.

The first step toward the management of disease was replacement of demon theories and humours theories by the germ theory. That very step, the beginning of hope, in itself dashed all hopes of magical solutions. It told workers that progress would be made stepwise, at great effort, and that a persistent, unremitting care would have to be paid to a discipline of cleanliness. So it is with software engineering today.

Does it have to be hard?—Essential difficulties

Frederick Brooks. No Silver Bullet - Essence and Accidents of Software Engineering. IEEE Computer, 1987.
Image from: https://twitter.com/zeljko_obren/status/909014656802574336

Reason: Essential Difficulties

Complexity

Conformity

Ease of Changes

Invisibility

⇨

They make SE different from other engineering fields

Now, let's return to some SWEBOK areas

Requirements

- What a system must do to meet clients needs
- Including quality of service attributes

Functional vs Non-Functional Requirements

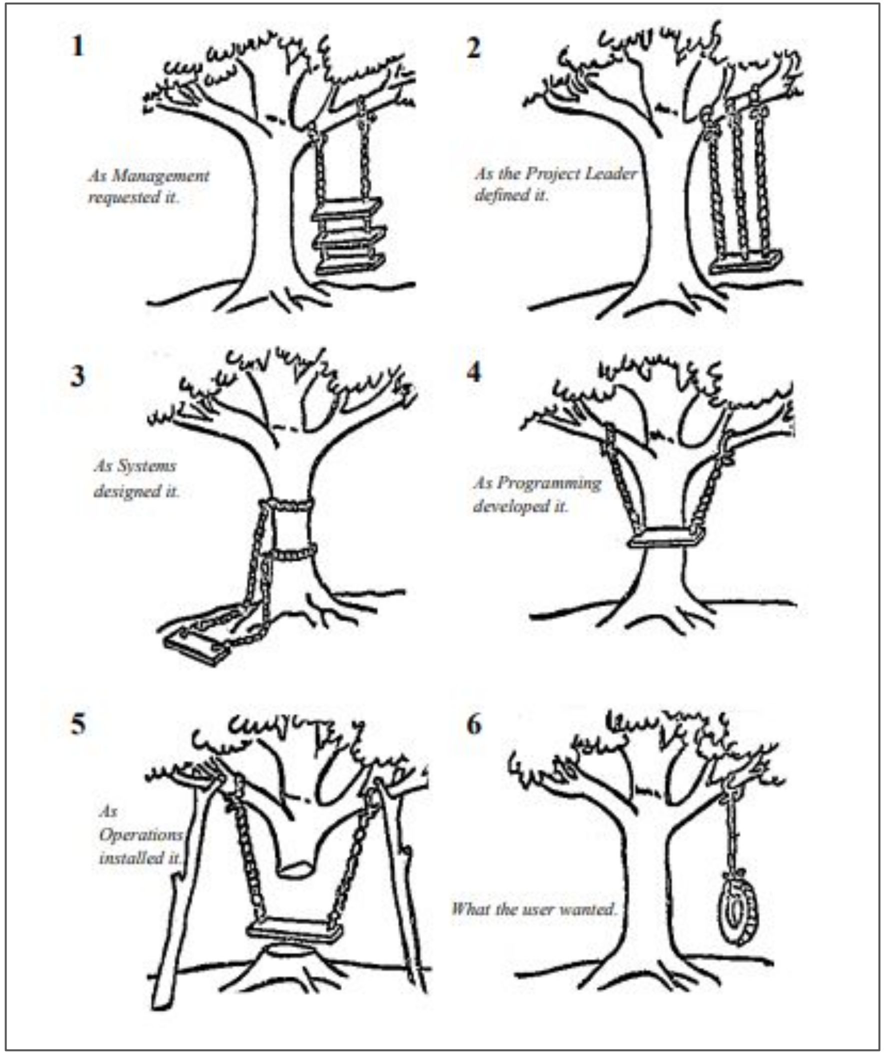
- Functional:
  - What a system should do
  - Features or services
- Non-functional:
  - How a system should operate
  - Under what constraints and with what quality of service

Examples of NFR (for a banking app)

- Performance: provide account balance in 5 seconds
- Availability: be online 99.99% of the time
- Capacity: store data for 1M customers
- Fault tolerance: continue operating if a datacenter goes down
- Security: encrypt data exchanges with branches

Examples of NFR

- Privacy: do not store user locations
- Interoperability: integrate with Central Bank systems
- Maintainability: bugs should be fixed in 24 hours
- Usability: version for cellphones and tablets


1
As Management requested it.
2
As the Project Leader defined it.
3
As Systems designed it.
4
As Programming developed it.
5
As Operations installed it.
6
What the user wanted.

Pre-1970 cartoon; origin unknown
Source: Bertrand Meyer. Object Success, 1995.

Testing

- Checks if a program has the expected results when executed with some test cases
- Two types:
  - Manual
  - Automated

Famous Software Failure: Explosion of Ariane 5 (1996)

30 seconds later

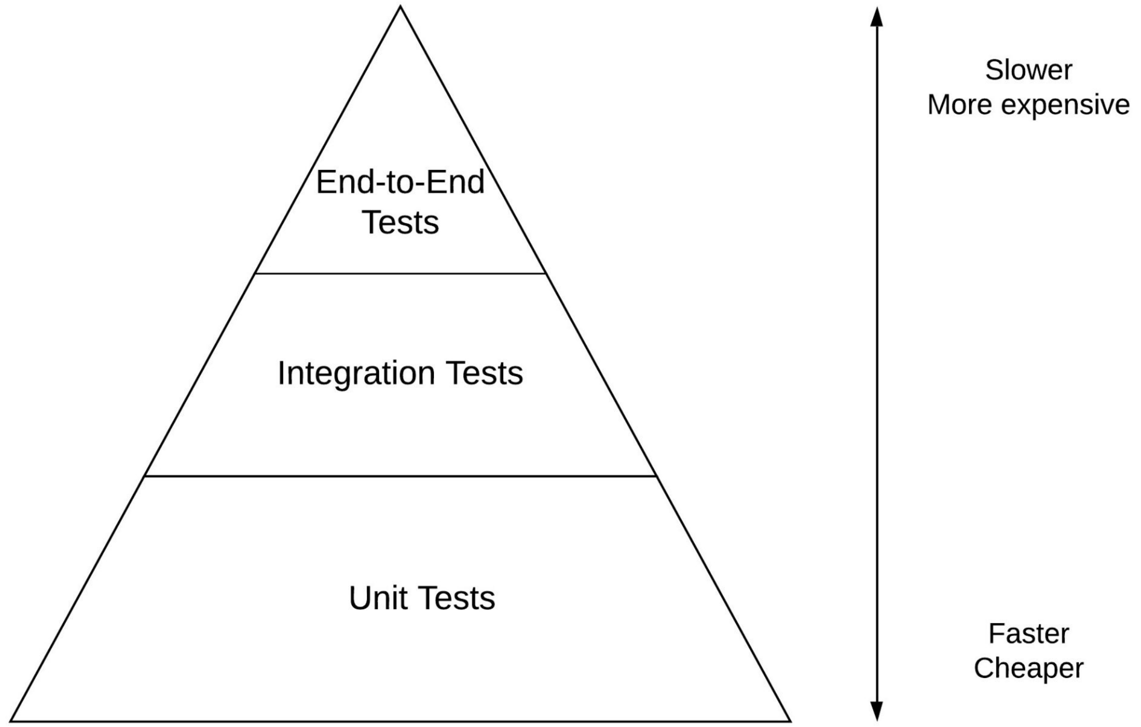
rocket + satellite: US$ 500M

Credits: ESA 1996

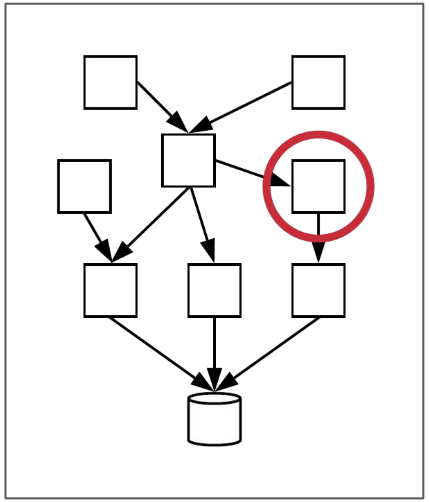
Explosion Investigation

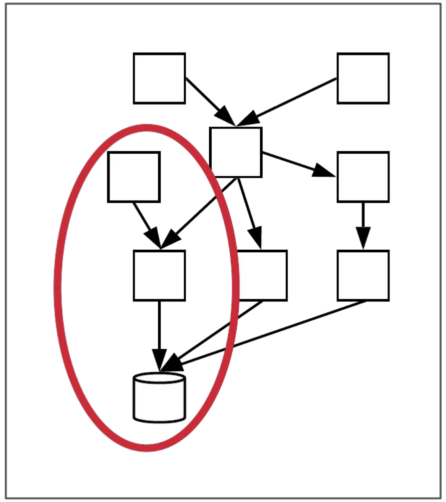
- Caused by a software failure
- Conversion 64-bit float ⇒16-bit integer
- Overflow: float didn't fit into 16 bits
- This overflow has never happened before

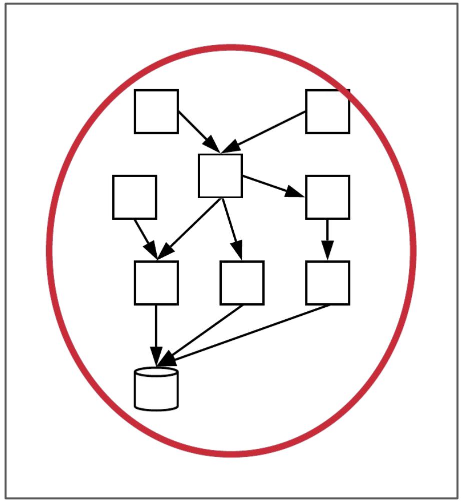
Test Pyramid



Types of Automated Tests


Unit

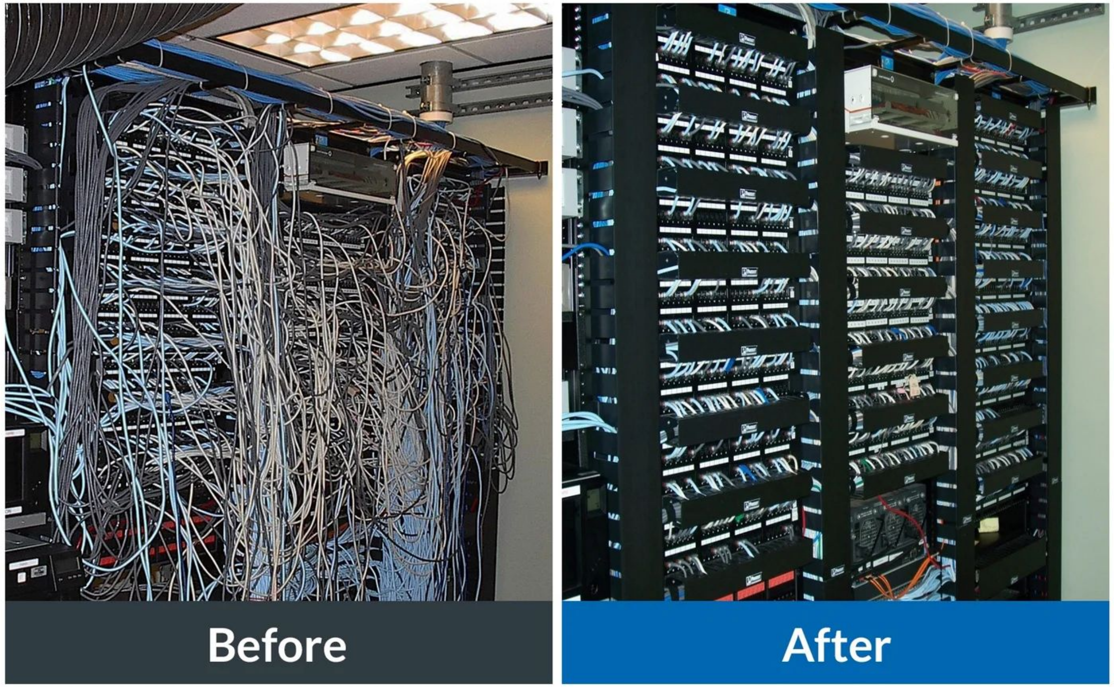

Integration


End-to-End

Maintenance

- Corrective
- Preventive
- Adaptive
- Evolutionary
- Refactoring

Refactoring in one slide



Legacy Systems

- Old systems, using old technologies (language, OS, DB)
- Maintenance is costly and risky
- But legacy ≠ irrelevant

COBOL lives…

- ~200 billion LOC in COBOL worldwide
- Most in banking systems
    - 95% of ATM transactions are in COBOL
    - Single European bank has 250 MLOC in COBOL

Source: Vadim Zaytsev's talk at SLE 2020 (https://youtu.be/sSkIUTdfDjs)

Cobol Example

```
PROGRAM-ID. CONDITIONALS.

DATA DIVISION.
  WORKING-STORAGE SECTION.
  *> setting up places to store values
  *> no values set yet
  01 NUM1 PIC 9(9).
  01 NUM2 PIC 9(9).
  01 NUM3 PIC 9(5).
  01 NUM4 PIC 9(6).
  *> create a positive and a negative
  *> number to check
  01 NEG-NUM PIC S9(9) VALUE -1234.
  *> create variables for testing classes
  01 CLASS1 PIC X(9) VALUE 'ABCD '.
  *> create statements that can be fed
  *> into a cobol conditional
  01 CHECK-VAL PIC 9(3).
    88 PASS VALUES ARE 041 THRU 100.
    88 FAIL VALUES ARE 000 THRU 40.
```

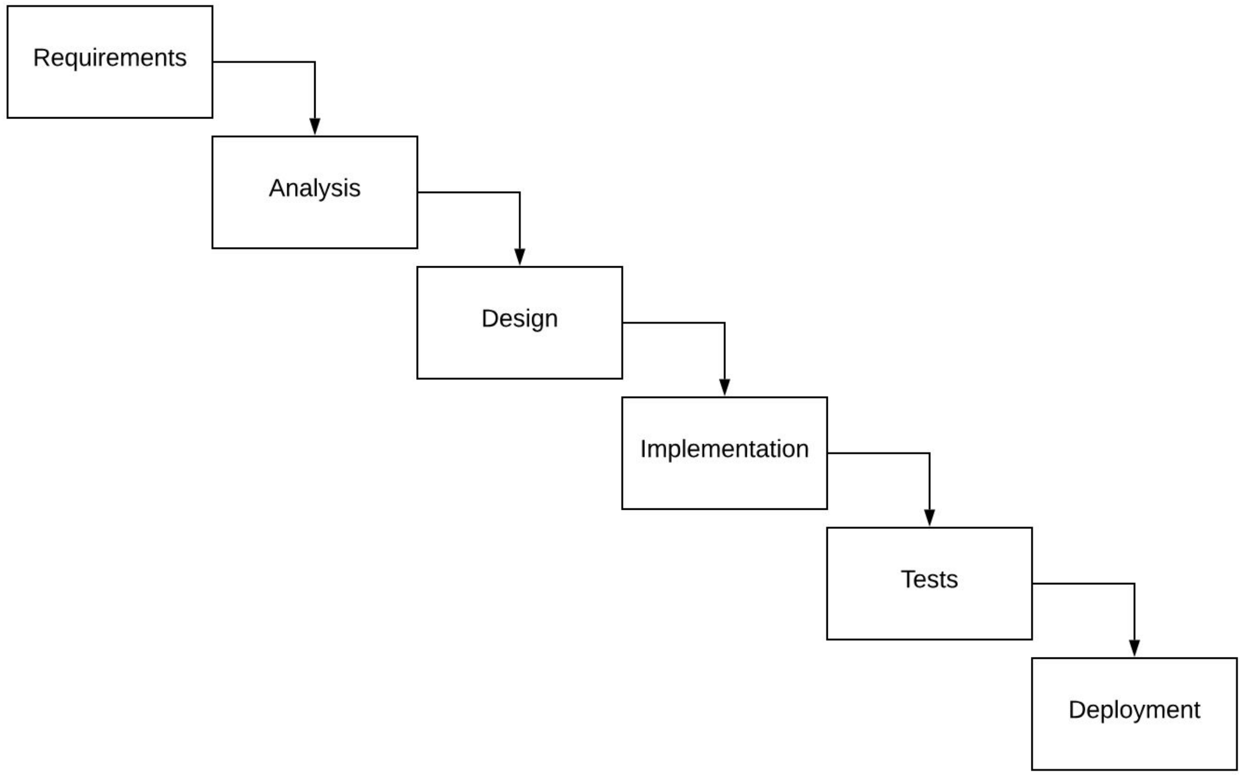
Source: GitHub gist

```
IDENTIFICATION DIVISION.
PROCEDURE DIVISION.
  *> set 25 into num1 and num3
  *> set 15 into num2 and num4
  MOVE 25 TO NUM1 NUM3.
  MOVE 15 TO NUM2 NUM4.

  *> comparing two numbers and checking for equality
  IF NUM1 > NUM2 THEN
    DISPLAY 'IN LOOP 1 - IF BLOCK'
    IF NUM3 = NUM4 THEN
      DISPLAY 'IN LOOP 2 - IF BLOCK'
    ELSE
      DISPLAY 'IN LOOP 2 - ELSE BLOCK'
    END-IF
  ELSE
    DISPLAY 'IN LOOP 1 -ELSE BLOCK'
  END-IF

  *> use a custom pre-defined condition
  *> which checks CHECK-VAL
  MOVE 65 TO CHECK-VAL.
  IF PASS
    DISPLAY 'PASSED WITH 'CHECK-VAL' MARKS.'.
  IF FAIL
    DISPLAY 'FAILED WITH 'CHECK-VAL' MARKS.'.

  *> a switch statment
  EVALUATE TRUE
    WHEN NUM1 < 2
      DISPLAY 'NUM1 LESS THAN 2'
    WHEN NUM1 < 19
      DISPLAY 'NUM1 LESS THAN 19'
    WHEN NUM1 < 1000
      DISPLAY 'NUM1 LESS THAN 1000'
  END-EVALUATE.
STOP RUN.
```

Processes

- Activities whe should follow to build a software system
- Two types:
    - Waterfall
    - Agile

Waterfall Model

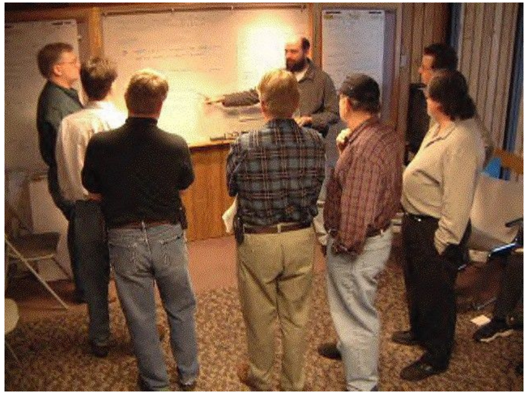


Problems with Waterfall

- Requirements often change
  - Complete requirements specification takes time
  - When it's finished, the world changed
- Moreover, customers usually don't know what they want
- Documentation is verbose and quickly becomes outdated

Agile Manifesto (2001)

- Meeting of 17 software engineers in Utah
- New model: incremental and iterative



https://siamchamnankit.co.th/history-some-pictures-and-pdfs-of-the-agile-manifesto-meeting-on-2001-a33c40bcc2b

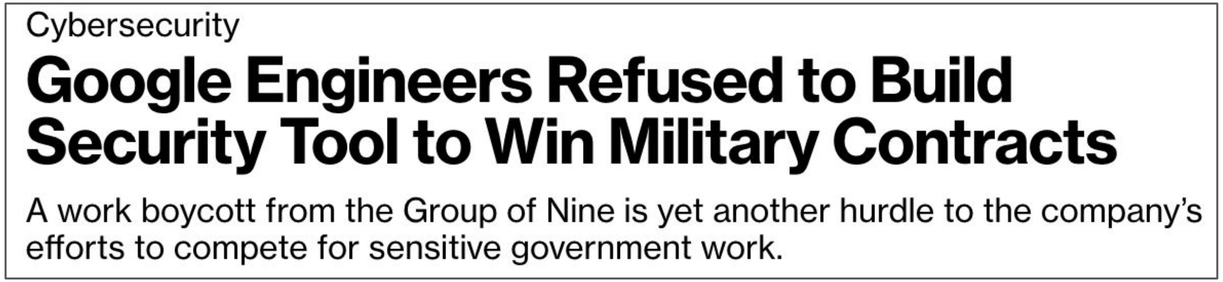
Major impact on the software industry (and beyond)

Harvard Business Review
134 Marketing Meets Mission
147 The Strategic Side Gig
44 Why Sexual Harassment Programs Backfire
HBR.ORG May–June 2020
The Agile Executive
How to balance efficiency and innovation (even in tough times like these) 64

May 2020

Ethical Aspects

- Devs are questioning the use of the software they create



Cybersecurity

Google Engineers Refused to Build Security Tool to Win Military Contracts

A work boycott from the Group of Nine is yet another hurdle to the company's efforts to compete for sensitive government work.

https://www.bloomberg.com/news/articles/2018-06-21/google-engineers-refused-to-build-security-tool-to-win-military-contracts

Types of Software Systems

The ABC of Software Engineering

- Classification proposed by Bertrand Meyer
- Three types of software:
    - Type C (Casual)
    - Type B (Business)
    - Type A (Acute)

https://bertrandmeyer.com/2013/03/25/the-abc-of-software-engineering/

Casual Systems (Type C)

- Very common
- Small systems, not very important
- Can have bugs; sometimes, they are temporary systems
- Implemented by 1-2 devs
- They don't benefit much from what we'll study
- The risk is over-engineering

Business Systems (Type B)

- Vey important to an organization
- Systems that benefit from what we will study in this course
- Risk: if we do not use SE techniques, they may become a liability, rather than an asset for organizations

Acute Systems (Type A)

- Software where nothing can go wrong, as the cost is immense, in terms of human lives and/or $$$
- Mission-critical systems

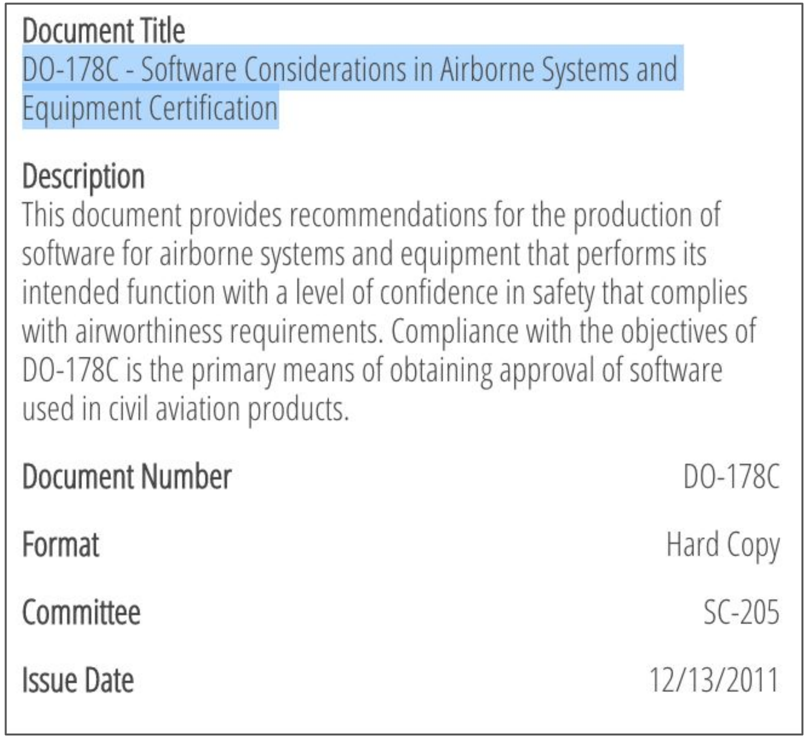
Subway

Aviation

Medicine

Acute Systems

- May require certifications
- They are beyond the scope of our course



Document Title
DO-178C - Software Considerations in Airborne Systems and Equipment Certification

Description
This document provides recommendations for the production of software for airborne systems and equipment that performs its intended function with a level of confidence in safety that complies with airworthiness requirements. Compliance with the objectives of DO-178C is the primary means of obtaining approval of software used in civil aviation products.

Document Number	DO-178C
Format	Hard Copy
Committee	SC-205
Issue Date	12/13/2011

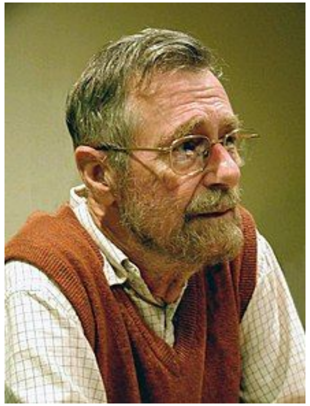
Exercises

1. Studies show that maintenance and evolution costs can reach 80% or more of a software’s total costs over its lifecycle. Explain why this value is so high.
2. Suppose that you have to build a bridge. Describe how a project for building this bridge would be assuming:
    a. Waterfall-based project
    b. Agile-based project
3. Refactoring is a code transformation that preserves behavior. What is the meaning of the expression preserve behavior? What restriction does it impose on refactoring activities?

4. In testing, there is this famous quote, by Edsger W. Dijkstra:

   "tests show the presence of bugs, but not their absence."

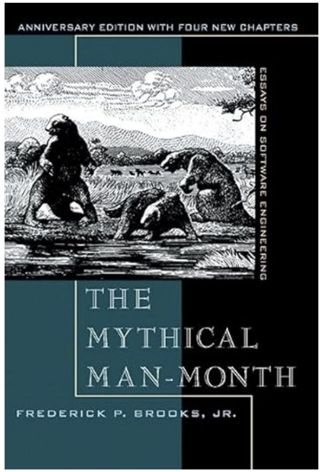
   Why are tests unable to show the absence of bugs?



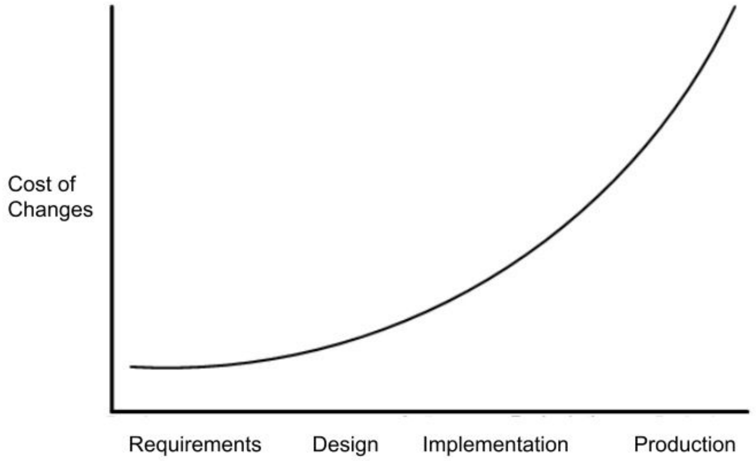
5. In software project management, there is an empirical law, called Brooks' Law, which says that:

   "adding new devs to a project that is late, makes it even later."

   Why does this fact tend to be true?


ANNIVERSARY EDITION WITH FOUR NEW CHAPTERS
ESSAYS ON SOFTWARE ENGINEERING
THE MYTHICAL MAN-MONTH
FREDERICK P. BROOKS, JR.

6. This chart illustrates how the costs of changes vary according to the development phase they occur for a given application. (a) Which development method would you recommend for this system, and why? (b) Give examples of systems that have a similar change cost curve.



7. In 2015, it was discovered that millions of cars manufactured by a major automobile company emitted pollutants within legal standards only during laboratory tests. Under normal usage conditions, the cars released higher levels of pollutants to enhance performance. Thus, the code possibly included a decision command like the following one (merely illustrative). What would you do if your manager asks you to write an if like the one above?

```
 if "car being tested in a laboratory"
    "comply with emission standards"
else
    "exceed emission standards"
```

End