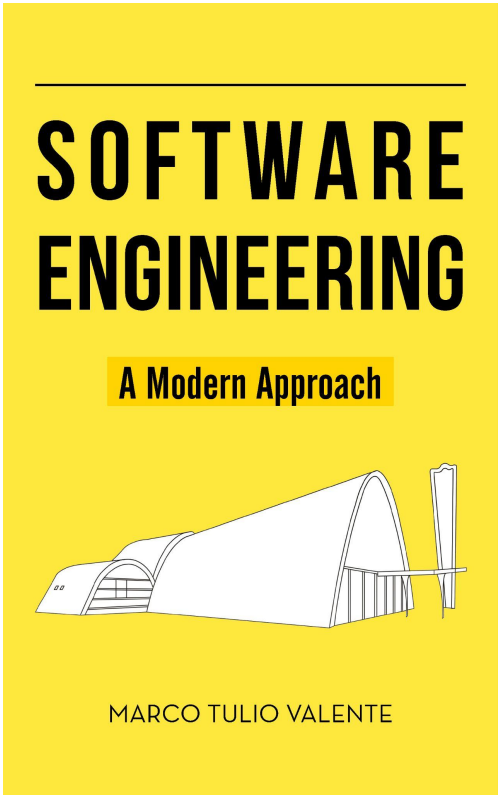


Chapter 1 - Introduction

Prof. Marco Tulio Valente

https://softengbook.org

NATO Conference (Germany, 1968)

Often regarded as the event that established the field of Software Engineering

Working Conference on Software Engineering

Comment from a participant

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

Areas of Software Engineering

SWEBOK®

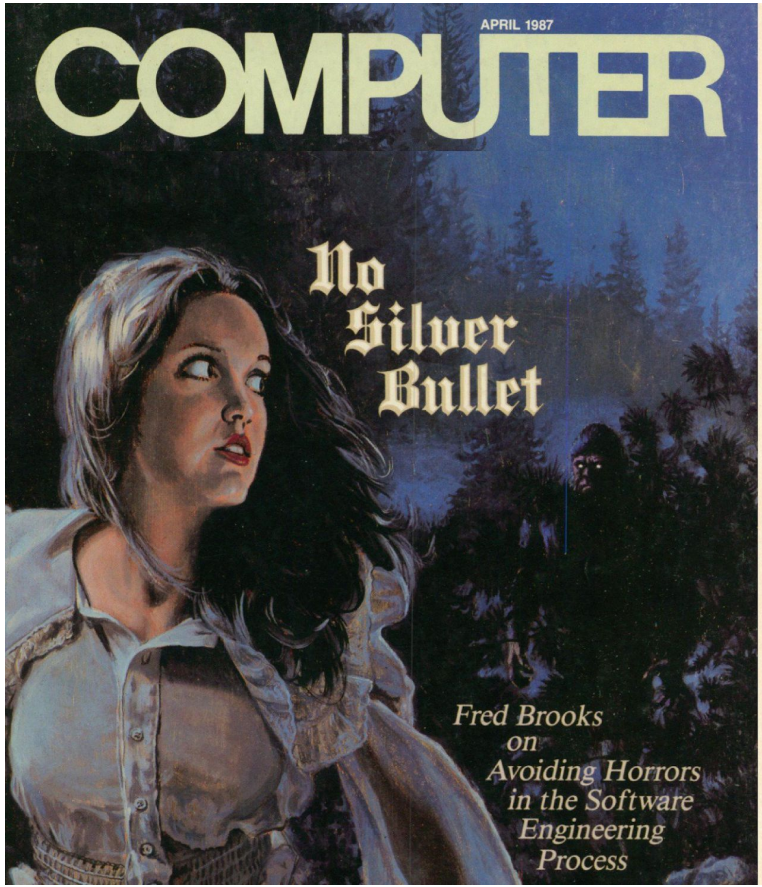
Guide to the Software Engineering Body of Knowledge

V3 (2014)	V4 (2024)
Introduction	Introduction
1. Software Requirements	1. Software Requirements
	2. Software Architecture
2. Software Design	3. Software Design
3. Software Construction	4. Software Construction
4. Software Testing	5. Software Testing
	6. Software Engineering Operations
5. Software Maintenance	7. Software Maintenance
6. Software Configuration Management	8. Software Configuration Management
7. Software Engineering Management	9. Software Engineering Management
8. Software Engineering Process	10. Software Engineering Process
9. Software Engineering Models and Methods	11. Software Engineering Models and Methods
10. Software Quality	12. Software Quality
	13. Software Security
11. Software Engineering Professional Practice	14. Software Engineering Professional Practice
12. Software Engineering Economics	15. Software Engineering Economics
13. Computing Foundations	16. Computing Foundations
14. Mathematical Foundations	17. Mathematical Foundations
15. Engineering Foundations	18. Engineering Foundations
Appendix A. Knowledge Area Specifications	Appendix A. Knowledge Area Specifications
Appendix B. Standards	Appendix B. Standards
Appendix C. Consolidated Reference List	Appendix C. Consolidated Reference List

In this class

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

But first a disclaimer



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

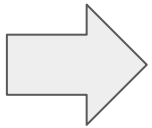
Reason: Essential Difficulties

Complexity

Conformity

Ease of Changes

Invisibility



These factors make SE different from other engineering fields

Now, let's return to some SWEBOK areas

Requirements

- What a system must do to meet customers' needs, including quality of service attributes

Functional vs Non-Functional Requirements

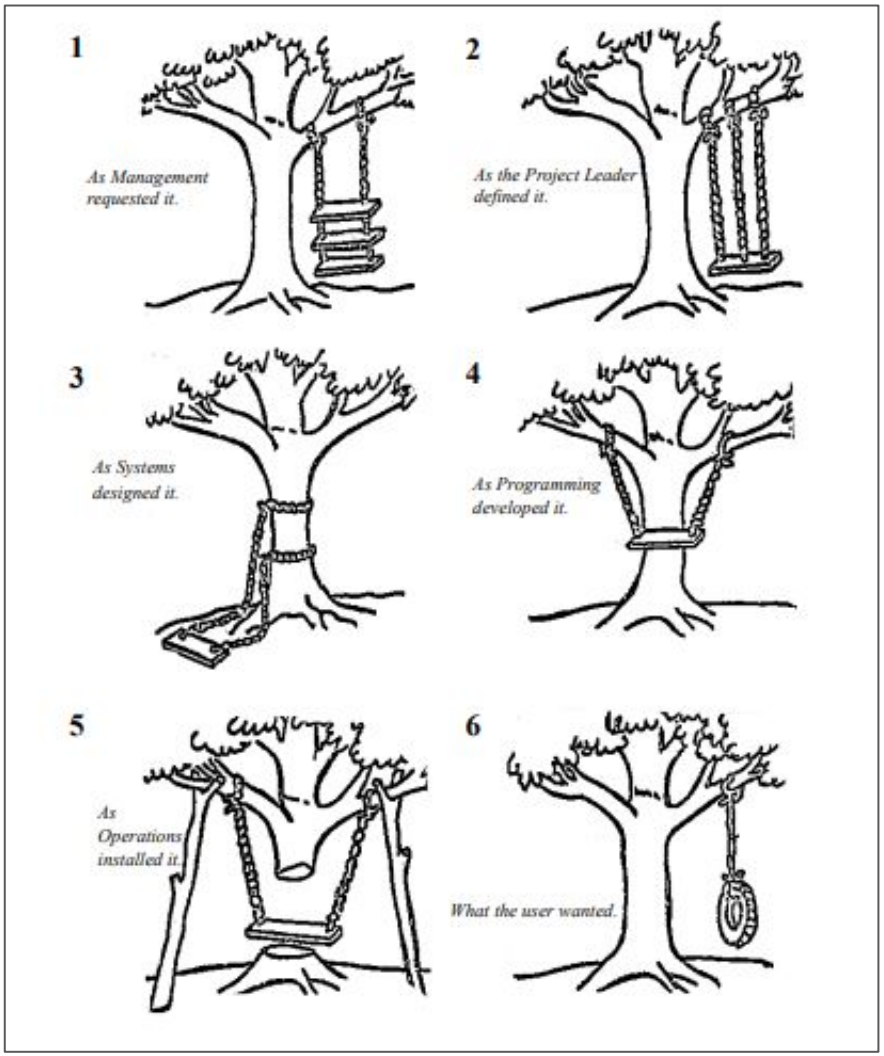
- Functional: what a system should do
  - Features or services
- Non-functional: how a system should operate
  - Under what constraints and with what quality of service

Examples of NFRs (for a banking app)

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

Examples of NFRs

- 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



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

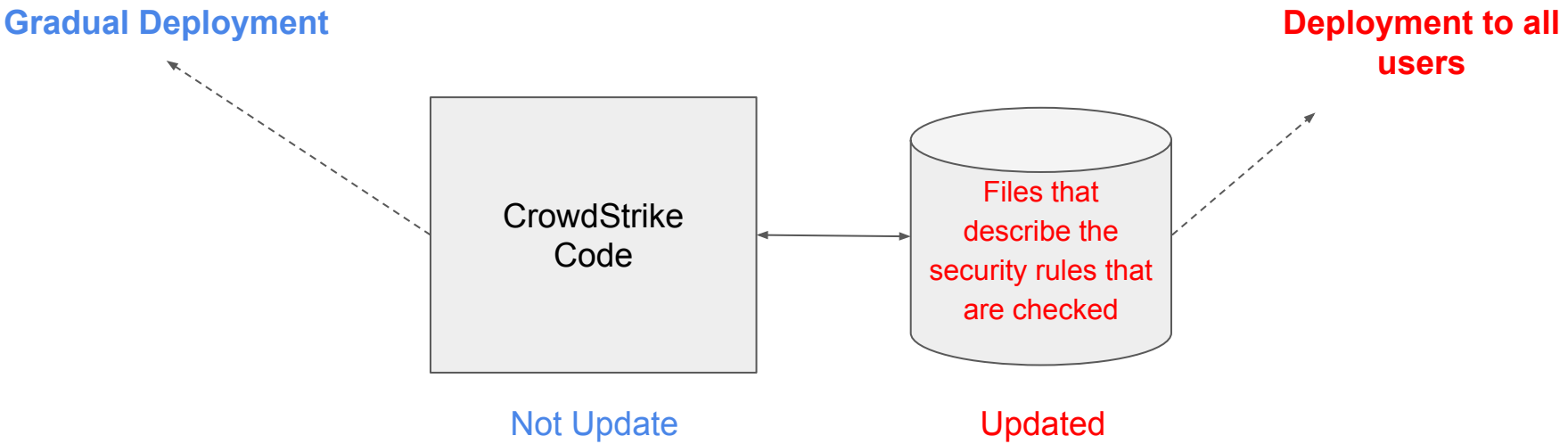
- Explosion was caused by a software failure
- Conversion 64-bit float ⇒16-bit integer
- Overflow: float didn't fit into 16 bits
- This issue had never occurred before!

CrowdStrike (July 2024)

8 million Windows machines

CrowdStrike

- Detects security threats ("more sophisticated anti-virus")
- Bug was caused by a version update


Gradual Deployment
Deployment to all users
CrowdStrike Code
Files that describe the security rules that are checked
Not Update
Updated

Postmortem

- On Friday, July 19, 2024 at 04:09 UTC, CrowdStrike released a content configuration update for Windows
- Systems in scope were online between Friday, July 19, 2024 04:09 UTC and Friday, July 19, 2024 05:27 UTC.

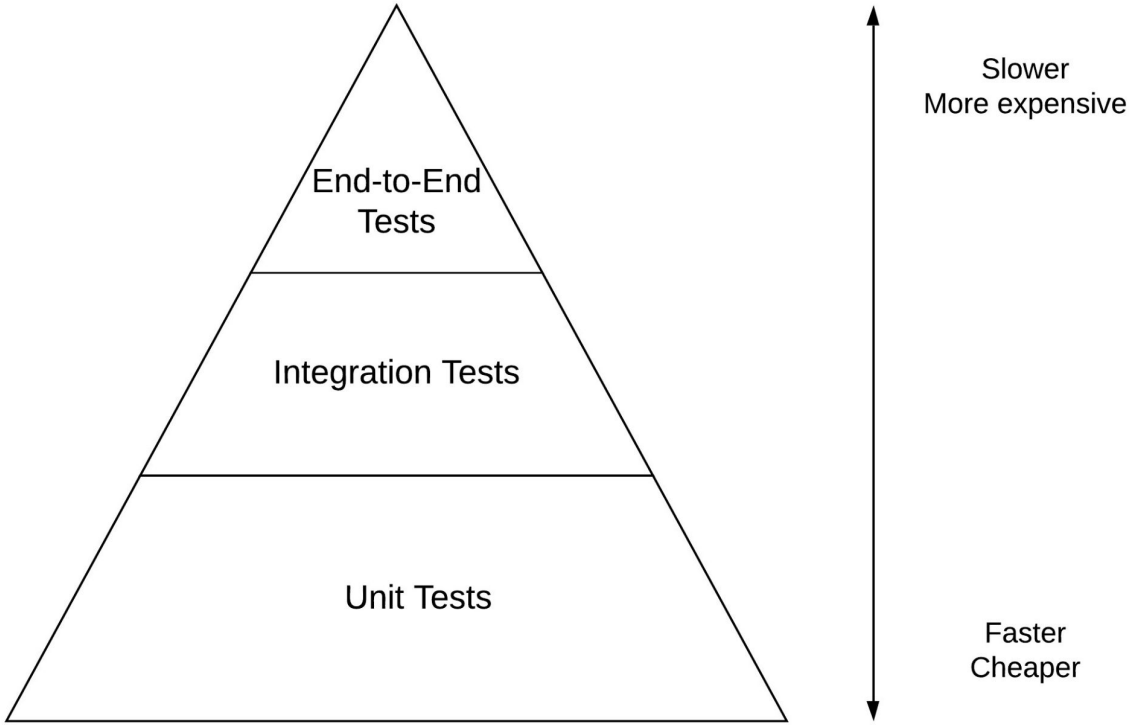
https://www.crowdstrike.com/blog/falcon-content-update-preliminary-post-incident-report/

https://www.crowdstrike.com/wp-content/uploads/2024/08/Channel-File-291-Incident-Root-Cause-Analysis-08.06.2024.pdf

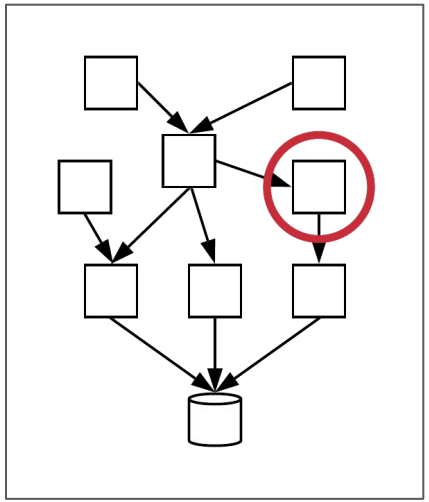
Postmortem

- Two additional IPC Template Instances were deployed.
- Due to a bug in the Content Validator, one of the two Template Instances passed validation despite containing problematic data.
- When loaded into the Content Interpreter, problematic content resulted in an out-of-bounds memory read triggering an exception.
- This exception resulted in a Windows crash (BSOD).

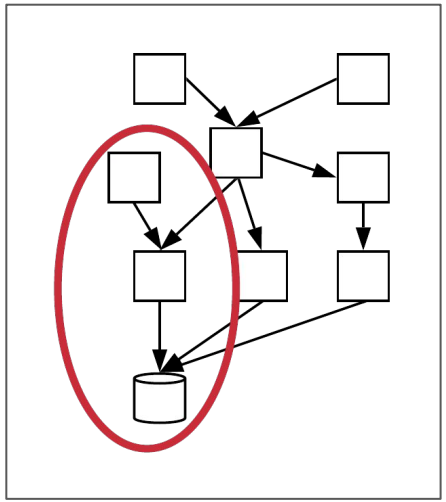
Test Pyramid



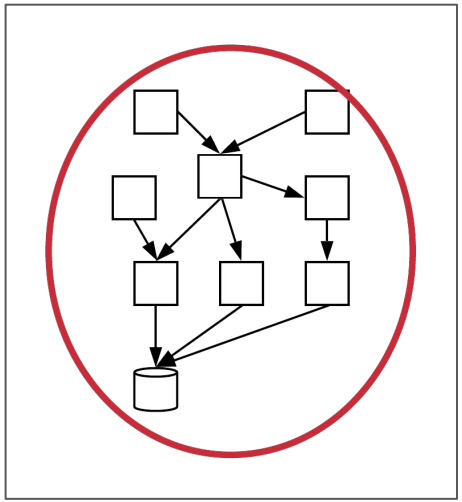
Types of Automated Tests



Unit



Integration

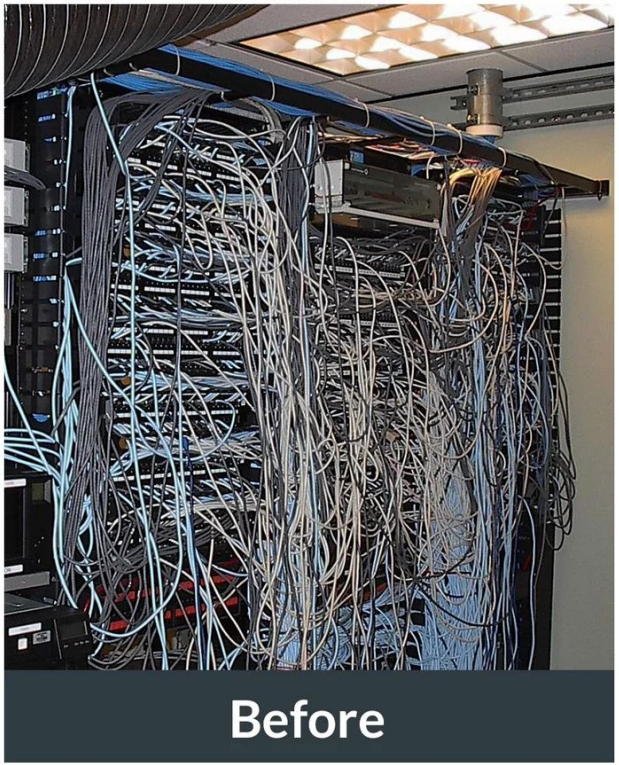


End-to-End

Maintenance

- Corrective
- Preventive
- Adaptive
- Evolutionary
- Refactoring

Refactoring in one slide



Legacy Systems

- Old systems, using old technologies (PL, OS, DB)
- Maintenance is expensive and risky
- But legacy does not mean 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.
```

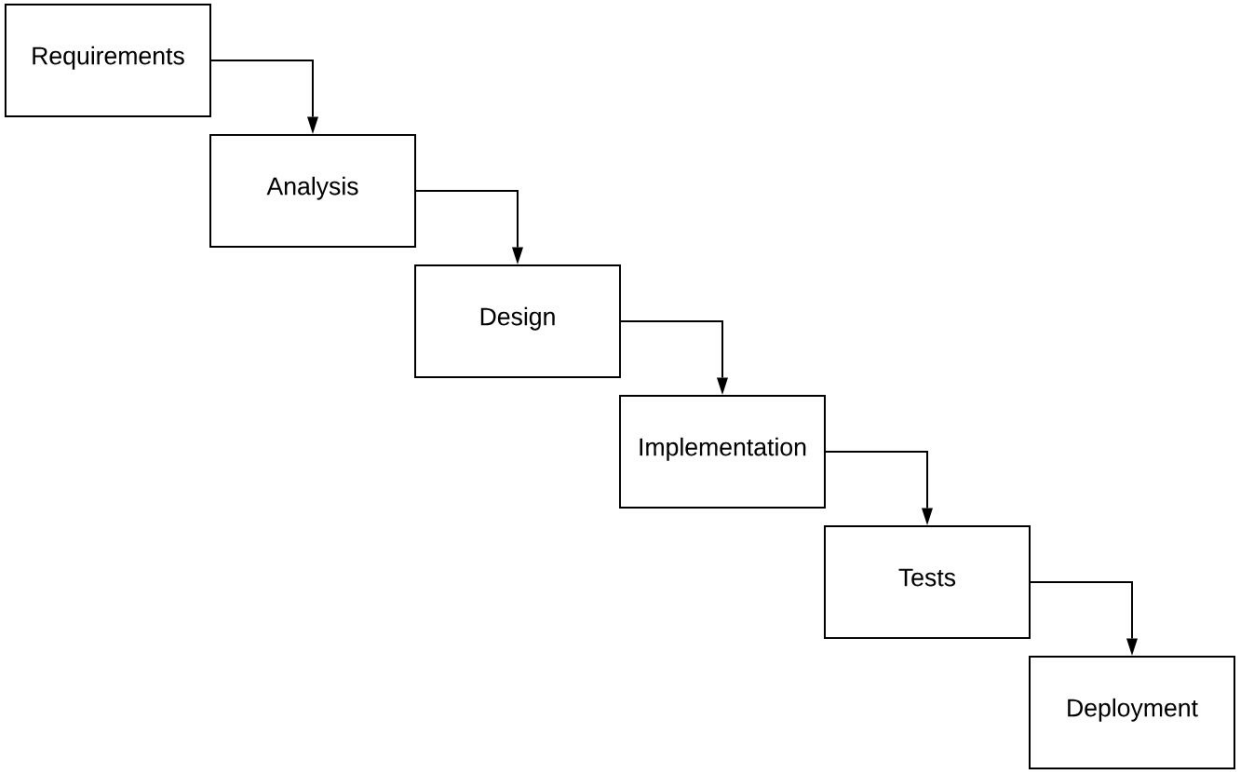
```
IDENTIFICATION DIVISION.
PROCEDURE DIVISION.
  *> set 25 into num1 and num3
  *> set 15 into num2 and num4
  MOVE 25 TO NUM1 NUM3.
  MOVE 15 TO NUM2 NUM4.

  *> 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.
```

Source: GitHub gist

Processes

- Activities that should be followed to build a software system
- Two main types:
  - Waterfall
  - Agile

Waterfall Model

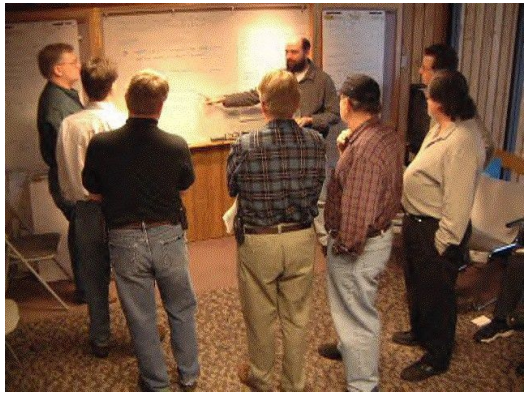


Problems with Waterfall

1. Requirements often change
    - Complete requirements specification takes time
    - By the time it's finished, the world has changed!
2. Customers usually don't know what they want
3. 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)

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
- Implemented by 1-2 devs
- They don't benefit much from what we'll study here
- The main risk is over-engineering

Business Systems (Type B)

- Vey important to an organization
- Systems that benefit from what we will study here
- 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

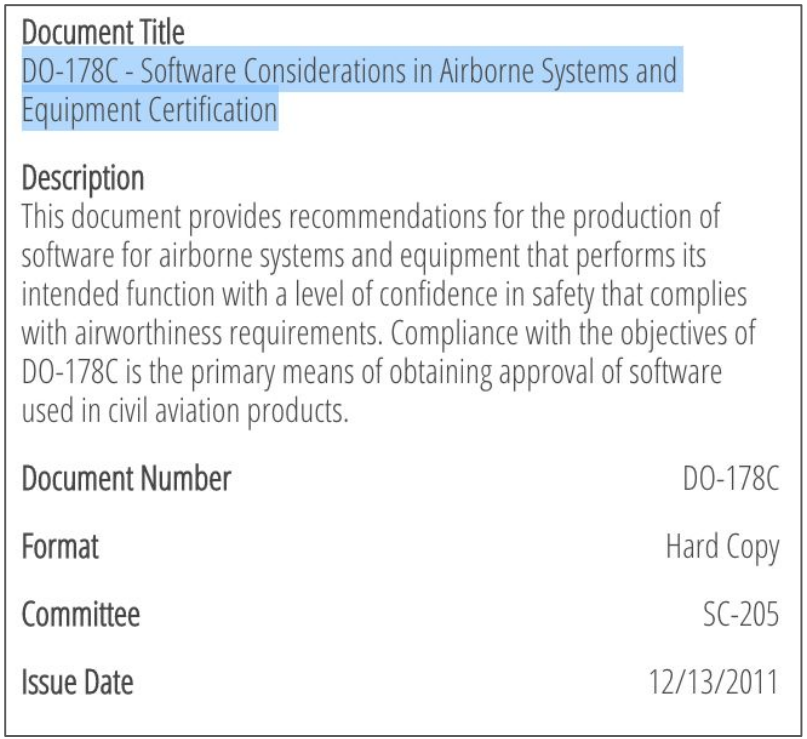
Subway

Aviation

Medicine

Acute Systems

- May require certifications
- 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

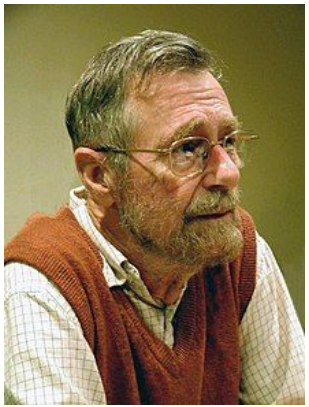
Exercises

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"

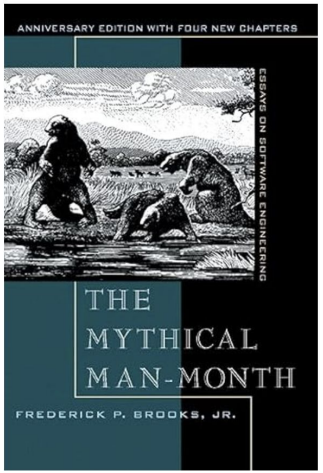
   Why are tests unable to show the absence of bugs?



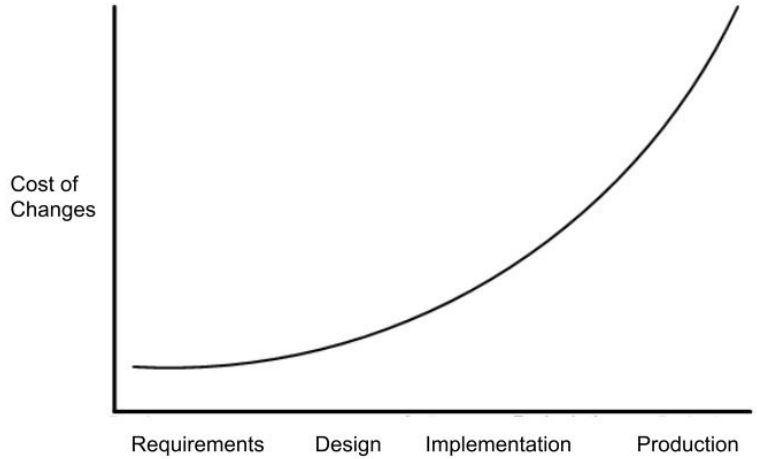
5. In software project management, there is an empirical law, called Brooks' Law, which says that:

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

Why does this phenomenon tend to occur?



6. This chart illustrates how the costs of changes vary according to the development phase of a given application. (a) Which development method would you recommend for this system, and why? (b) Provide 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. The code likely included a decision command similar to the following (only illustrative). What would you do if your manager asked you to write an "if" statement like the one below?

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

End