Software Engineering

Prof. Dr. Bertrand Meyer
Dr. Manuel Oriol
Dr. Bernd Schoeller

Lecture 2: Software Engineering Fundamentals
Today

- We try to put Software Engineering in an historical perspective

- We present several methods and ideas that can help you build software in a practical way

- We show what most people software engineers remember of Software Engineering (sic!)
Software Engineering

Two Notions are Important:

- **Software**
  - Programs
  - Achievements: Internet, Personal Computers, Information Society...

- **Engineering**:
  - Building Process
  - Achievements: Pyramids, Eiffel Tower, Bridges, Cars...
Where it all started

Augusta Ada King, Countess of Lovelace (1815 - 1852)
“First Computer Programmer”
“...an analyzing process must equally have been performed in order to furnish the Analytical Engine with the necessary operative data; and that herein may also lie a possible source of error. Granted that the actual mechanism is unerring in its processes, the cards may give it wrong orders.”

in

Sketch of The Analytical Engine Invented by Charles Babbage by L. F. Menabrea

with notes upon the Memoir by the translator Ada Augusta, Countess of Lovelace
“It has been just so in all of my inventions. The first step is an intuition, and comes with a burst, then difficulties arise—this thing gives out and [it is] then that "Bugs"—as such little faults and difficulties are called—show themselves and months of intense watching, study and labor are requisite before commercial success or failure is certainly reached.”

- Thomas Eddison, in a letter, 1878 (wikipedia, Software Bugs)
More bugs...

Mark II operator William "Bill" Burke, 1947,
Pasted into log book by Grace Hopper -> coined term Debug
At that time...

- Harvard Architecture separates data and program:
  - Program on punched tape
  - Data in memory
How to make programs that do not bug?

- That’s the real question...
- Idea: When your actual code is too close to the machine, it is hard to debug

How to read: (x86 asm)

```
cseg   segment para public 'CODE'
    assume  cs:cseg,ds:cseg
start:
    jmp   start1
msgstr db      'Enter Fahrenheit '
crlf   db      13,10,'$
```
Main Idea

- Change the Programming Language!
The example of FORTRAN

- FORTRAN 53 (32 instructions)
- FORTRAN 58
  - added procedures!
- FORTRAN IV (1962)
  - Added logical expressions and logical IF (before only arithmetical IF)
- FORTRAN 66
  - ANSI standard
- FORTRAN 77
  - ELSE, DO WHILE (before GOTO was used)...
- FORTRAN 90 and 95
  - Modules, abstract data types,
- FORTRAN 2003, 2008
  - Objects, procedure pointers
First idea

- In structured programming (Böhm & Jacopini'66, Dijkstra'69), a program is always expressible as the control-flow instructions:
  - Concatenation (blocks)
  - Selection (if, switch...)
  - Repetition (loops)
  - One entry-point

What is missing?

GOTO, Multiple entry points
Why does it help?

• It is easier to understand the code
  
  ➢ Easier to prove that it works (Dijkstra thought that code should show a proof)
  
  ➢ Easier to maintain
  
  ➢ Easier to write

• Top-down design and programming
This translate into...

- Procedural programming
- The unit is the procedure and it groups individual statements
- Programmers do not use destructuring instructions (goto)
- Top-down approach for designing
Top-Down approach

- Also called "Divide and Conquer"
- Best example: Stepwise refinements

- The idea is to consider the problem in its entirety and state it simply
- Then decompose into smaller units in a recursive manner
- When not possible to decompose anymore, code the smaller units
Example

- A program that removes the comments of source code
Example (2)

- A program that removes the comments of source code
  - Read the file
  - Remove comments
  - Store the modified file back
A program that removes the comments of source code

- **Read the file**
  - Open the file
  - Read line by line and put in an array of strings

- **Remove comments**
  - Iterate through the array, in each line, remove the comment

- **Store the modified file back**
  - Iterate through the array, store each line
Example (4)

- A program that removes the comments of source code
  - **Read the file**
    - Open the file
    - Read line by line and put in an array of strings
  - **Remove comments**
    - Iterate through the array,
      - in each line look for first sequence "--"
      - remove the rest of the line
  - **Store the modified file back**
    - Iterate through the array, store each line
Example (5)

- Write the code!
Evaluation

• Advantages:
  - Code is modular
  - Skeletons illustrate the use
  - Programmers do not miss a part of the implementation

• Disadvantage:
  - The program works at the end only
Bottom-up Approach

- The idea is to make the small parts first and let the environment assemble them.

- As an example, PROLOG programs are a specification of the inferences. The system deduces the facts.
Top-down AND Bottom-up

- Most of the current approaches actually mix the two approaches:
  - Top-down for the design
  - Bottom-up, by using libraries or composing components
And the world became Objects

- Breakthrough with Simula 67
- Coupling data types and code
- Objects deal with reuse
- Objects deal with modularity
- Objects model naturally some paradigms (e.g. GUI, libraries)
Modularity

- Classes are whole units they allow the co-localization of code and relevant data
- Deferred (abstract) classes help enforce that
- Client relationship helps to encapsulate
- Easy graphical representation
Reuse

- Easy extensions with inheritance.
- Generics help reuse the code across types
- Strong types help with the possible mismatches
- Libraries shorten development time
Design by Contracts

- Design by contracts is the next step towards correctness of the code and the data (in Eiffel, Java/JML, Spec#)

- Invariants check data at each method call

- Preconditions ensure that the application is in a valid state before a call

- Postconditions ensure that the code proceeded in a good manner
Languages only are not the solution...

- The fact is that bugs are still found in most (if not all) programs, classes...

- As an example automated tools find bugs in (almost) all classes
Bugs found by AutoTest in Eiffel classes
Other factors

- The final programmer code is not the only parameter to consider when executing the code: the operating system, the libraries, the runtime system

- What is needed is not easily identifiable

- Programmers can make mistakes
How do you develop a project?

- Code and fix?

- Design, code and fix?

- Design, code, test, fix?

- Design, code, test, document and fix?
Suggestions to improve the situation?

- What would you do to improve the situation?
How to reduce the issues?

- By Describing: Specifying and Documenting
- By Implementing: Designing and *Coding*
- By Assessing: verifying, validating, analyzing, testing, measuring (both products and processes).
- By Managing: organizing the work, communicating, collaborating
- By Operating: deploying systems and overseeing their proper functioning.
A software development process is a process used to develop applications.

Proponents of the process hope that it reduces the discrepancies between what the program is supposed to do and what it actually does.

There are two main types of processes:

- Sequential
- Iterative
What is considered?

- Feasibility study
- Global design
- V & V
- Unit Testing

- Requirements
- Detailed design
- Acceptance Testing
- System Testing

- Specification
- Implementation
- Distribution
- Prototyping
Original waterfall model

Winston Royce, 1970, source: wikipedia, waterfall model
Waterfall risk profile

C. Larman Agile & Iterative Development *A Manager guide* Addison Wesley 2003 p. 58
Strength and Weaknesses

· Advantages:
  - Widely used...
  - Time spent early on helps later on to remove some unnecessary delays
  - Emphasis on documentation and source code
  - Simple

· Disadvantages
  - Difficult for big projects to have one phase only
  - Most steps are not easily finished
  - Even Royce advocated for an iterative model in the original paper!
The sashimi model?

- The steps are overlapping:

Peter DeGrace
Sequential Model: V-model

Feasibility study

Requirements

Global design

Specification design

Detailed design

Implementation

Unit Testing

System Testing

Acceptance Testing

Distribution

German Ministry of Defense, 1992
V-Model (2)

Source: V-model, wikipedia
Strength and Weaknesses

- Advantages:
  - The testing begins very early on in the project
  - The implementation and design are guided by testing

- Disadvantages
  - It is still a sequential model (requirements do not change etc...)
Sequential Model: Evolutionary Model

Brooks, 1995
Strength and Weaknesses

- **Advantages:**
  - The prototype permits an early assessment
  - User test the first prototype and give feedback

- **Disadvantages**
  - Code it twice! (sometimes not possible)
Iterative Model: Prototype Model

- Implementation
- Prototyping
- User Tests
- Distribution
Strength and Weaknesses

• Advantages:
  - The prototype permits an early assessment
  - User test the prototype and give feedback
  - Feedback integrated

• Disadvantages
  - You have to code it twice! (sometimes not possible)
Iterative Model: Spiral Model

Barry Boehm, 1988
Characteristics

- The iterations typically take 6 months to 2 years to proceed
- It creates successions of prototypes as well
Strength and Weaknesses

• Advantages:
  - Manages the risk
  - Estimates are more realistic with time
  - Manage changes
  - Software Engineers start working earlier

• Disadvantages
  - A bit of an overkill for small projects
Why did it evolve?

- Agencies need to control the development and have accountability for delays

- The quality of the software is at the core of the processes

- People can then rely on a more codified process
Today

- We saw how programming languages evolved to help programmers avoid writing bugs as much as possible.

- We talked about the emergence of processes for engineering software.

- We showed processes that are very used in companies.
Next week

- Requirements
  - Standards
  - Categories