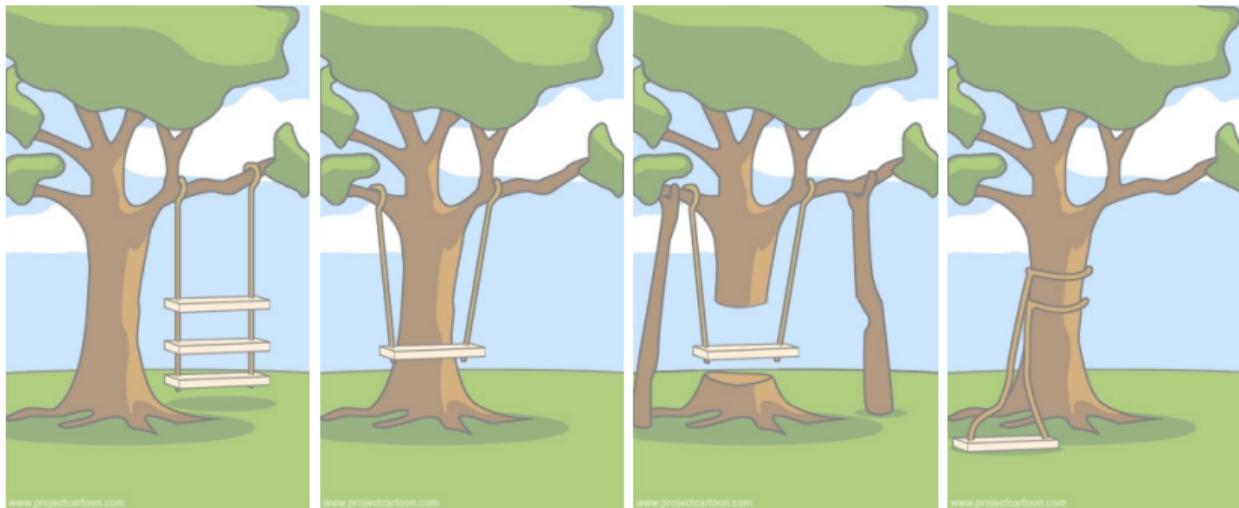




# Software Engineering

6. Implementation | Thomas Thüm | December 10, 2020

# Implementation in Software Projects



how the customer  
explained it

how the project  
leader understood it

how the analyst  
designed it

how the programmer  
implemented it

# Lecture Overview

1. Programming Languages
2. Coding Conventions
3. Tools and Environments

## Programming Languages

# Questionnaire Results

1

Warst du jemals betroffen von einer Softwarepanne?

| Antworten | relative Häufigkeit | absolute Häufigkeit |
|-----------|---------------------|---------------------|
| Ja        | 83%                 | 86                  |
| Nein      | 17%                 | 17                  |

2

Hast du dich im letzten Semester über Softwarefehler geärgert?

| Antworten | relative Häufigkeit | absolute Häufigkeit |
|-----------|---------------------|---------------------|
| Ja        | 78%                 | 85                  |
| Nein      | 22%                 | 24                  |

3

Hast du selbst Programmiererfahrung?

| Antworten | relative Häufigkeit | absolute Häufigkeit |
|-----------|---------------------|---------------------|
| Ja        | 96%                 | 108                 |
| Nein      | 4%                  | 4                   |

4

Hast du bereits Software getestet?

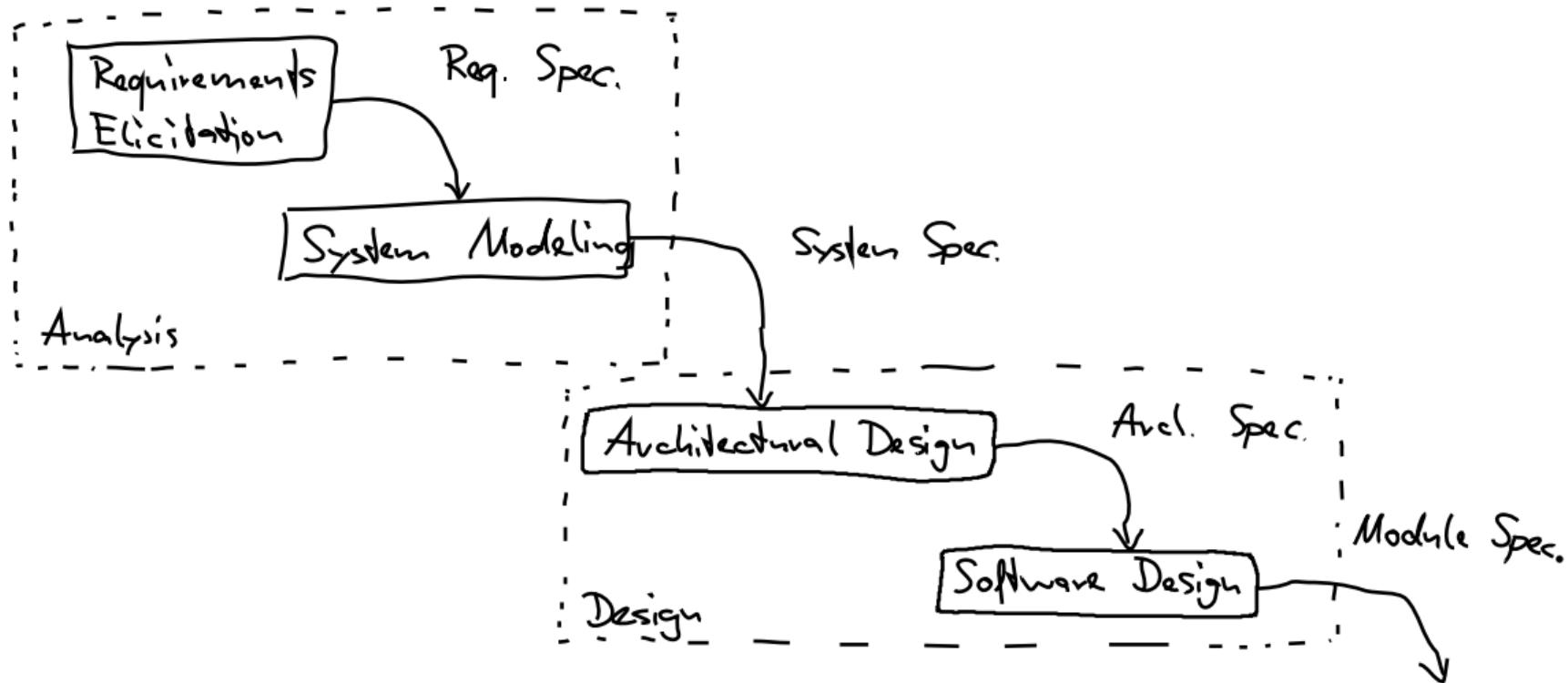
| Antworten | relative Häufigkeit | absolute Häufigkeit |
|-----------|---------------------|---------------------|
| Ja        | 56%                 | 60                  |
| Nein      | 44%                 | 48                  |

5

Hast du schon mal Programmierfehler verursacht?

| Antworten | relative Häufigkeit | absolute Häufigkeit |
|-----------|---------------------|---------------------|
| Ja        | 94%                 | 103                 |
| Nein      | 6%                  | 6                   |

# Analysis and Design



# History of Programming Languages

## Languages

[Jones + Krypczyk/Bochkor]

- 1945: first high-level language Plankalkül by Konrad Zuse (compiler written in 1998)
- 1954: first professional high-level language FORTRAN (Formula Translator) by IBM
- 1963: Basic as general-purpose language
- 1959: functional language Lisp
- 1970: first object-oriented lang. Smalltalk-80
- 1970: declarative language SQL
- 1971: Pascal by Niklaus Wirth for teaching
- 1974: very common procedural language C
- 1977: logical language Prolog
- 1980: C++ as object-oriented extension of C
- 1990: object-oriented language Java
- 1990: functional language Haskell
- 1991: multi-paradigm language Python
- 1995: scripting language JavaScript

## Milestones

[Jones]

- controlling behavior of mechanical devices by wiring or with punchcards ([Lochkarten](#))
- machine languages used during World War II
- assembly languages: distinction between human-readable instructions (source code) and executable instructions (object code)
- birth of compilers and interpreters having a one-to-many mapping between source and object code (opposed to one-to-one mapping in assemblers)
- structured programming pioneered by David Parnas and Edsger Dijkstra
- high-level programming languages: high number of executable for each human-readable instruction
- domain-specific languages, later general-purpose programming languages

# Programming Languages Today

Today

[Jones + Krypczyk/Bochkor]

- 2002: C# by Microsoft
- 2009: Go by Google
- 2010: Rust by Mozilla Research
- 2014: Swift by Apple
- thousands of programming languages
- very few programming languages used for more than 10 years
- languages used for more than 25 years: Ada, C, C++, COBOL, Java, Objective C, PL/I, SQL, Visual Basic, ...

Many Languages

[Jones]

- good: fit for every use case
- bad: developer training for new and dead languages, expensive tool support

# Choice of Programming Languages

## Desired Properties

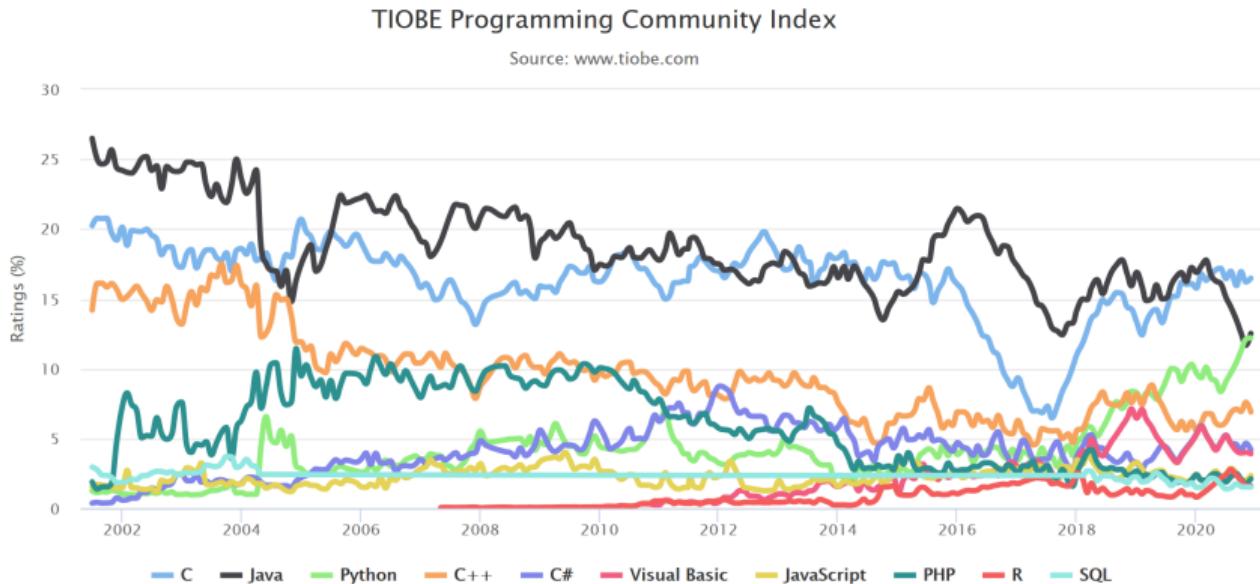
- modular implementation
- separation of interfaces and implementations
- type system: strongly/weakly typed languages
- readable syntax (FORTRAN vs ALGOL60)
- automatic pointer management (C vs Java)
- exception handling

## [Ludewig and Licher]

## Criteria in Practice

- language required by the company or customer?
- existing infrastructure?
- domain-specific languages available?
- language known/liked by developers?
- available libraries?
- available tool support?
- language popularity?
- what may change in the future?

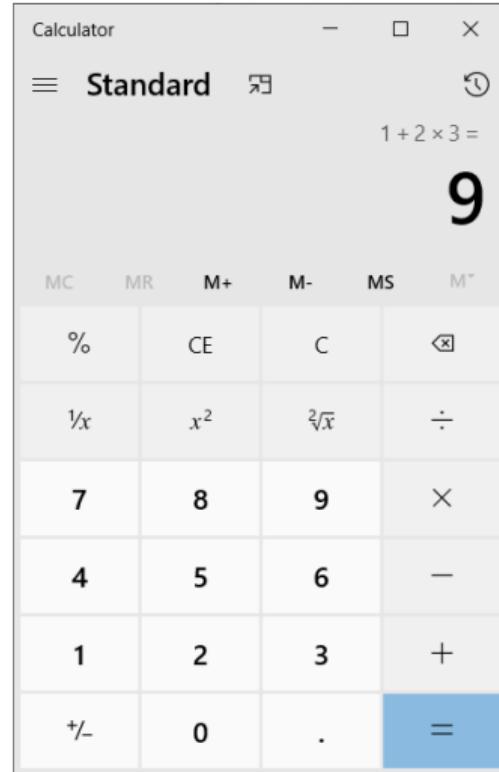
# Popularity of Programming Languages



# Popularity of Programming Languages

| Programming Language | 2020 | 2015 | 2010 | 2005 | 2000 | 1995 | 1990 | 1985 |
|----------------------|------|------|------|------|------|------|------|------|
| C                    | 1    | 2    | 2    | 1    | 1    | 1    | 1    | 1    |
| Java                 | 2    | 1    | 1    | 2    | 3    | 29   | -    | -    |
| Python               | 3    | 5    | 6    | 7    | 22   | 13   | -    | -    |
| C++                  | 4    | 3    | 3    | 3    | 2    | 2    | 2    | 8    |
| C#                   | 5    | 4    | 5    | 6    | 10   | -    | -    | -    |
| JavaScript           | 6    | 8    | 10   | 10   | 7    | -    | -    | -    |
| PHP                  | 7    | 6    | 4    | 4    | 19   | -    | -    | -    |
| SQL                  | 8    | -    | -    | -    | -    | -    | -    | -    |
| R                    | 9    | 14   | 46   | -    | -    | -    | -    | -    |
| Swift                | 10   | 15   | -    | -    | -    | -    | -    | -    |
| Lisp                 | 29   | 26   | 14   | 13   | 9    | 6    | 4    | 2    |
| Fortran              | 31   | 21   | 24   | 14   | 13   | 14   | 3    | 5    |
| Ada                  | 34   | 23   | 21   | 16   | 17   | 3    | 9    | 3    |

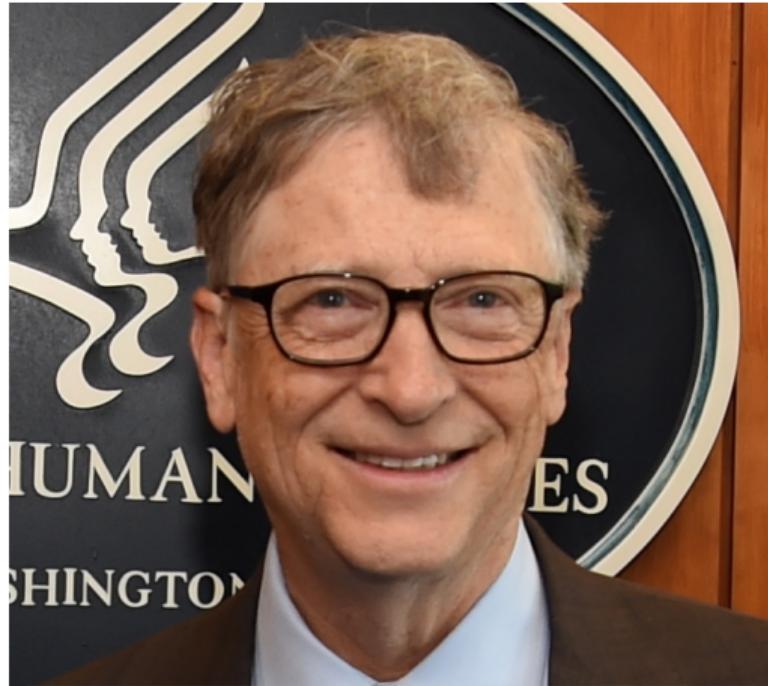
# Excursion: Windows Calc





**Patrick McKenzie**

“Every great developer you know got there by solving problems they were unqualified to solve until they actually did it.”



**Bill Gates**

“Learning to write programs stretches your mind, and helps you think better, creates a way of thinking about things that I think is helpful in all domains.”

# Programming Languages

## Lessons Learned

- Historical perspective on programming
- Criteria for choosing languages
- Popularity of programming languages
- Further Reading on Programming Languages: Jones, Chapter 8 Programming and Code Development + Krypczyk/Bochkor, Chapter 2.4 Programming Languages

## Practice

- Watch tutorial on how to change a C program into a C++ program:  
<https://twitter.com/peterc/status/1197963742266433536>
- Look at the code of my calculator and think about possible improvements to the code quality: <https://github.com/tthuem/2020WS-SE-Calculator/tree/2b93d004a76db27992d224864bbd41fea51bad65/Calculator/src/de/tubs/se1/CAL>

# Lecture Contents

## 1. Programming Languages

Questionnaire Results

Analysis and Design

History of Programming Languages

Programming Languages Today

Choice of Programming Languages

Popularity of Programming Languages

Excursion: Windows Calc

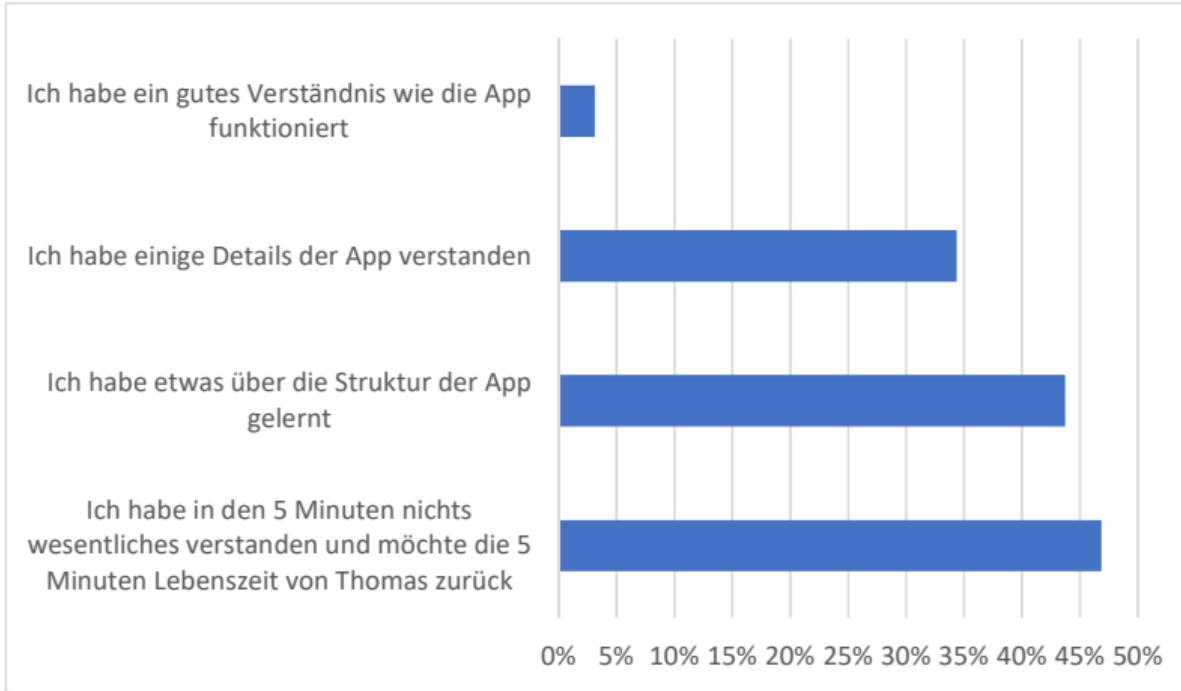
Lessons Learned

## 2. Coding Conventions

## 3. Tools and Environments

## Coding Conventions

# Understanding the Corona-Warn-App





**Douglas Crockford**

"It turns out that style matters in programming for the same reason that it matters in writing. It makes for better reading."



**François Chollet**

"In software, naming matters, because names reflect how you think about a problem. Code is also communication, and naming is a big part of making it work."

# Code Formatting

## Code Formatting

- motivation: code is read much more often and by more developers than written
- avoid differences by each programmer
- indentation: typically 4 characters per level
- length of a line: often 80 or 100 characters
- extra indentation: typically 8 characters when breaking extra long lines
- empty lines between methods and attributes
- automated code formatters available (on demand or when saving the editor)
- typical formatting rules for each language
- automated code formatters are configurable (handle with care)

# Rules on Naming

## Unwanted Names

- single character as a name
- very long names
- names consisting only of special chars
- synonyms: delete, remove, clear
- abbreviations (unless very common)

## Wanted Names

- nouns for class names: Calculator
- nouns for attribute names: calculateButton
- verbs for method names: getCalculator(), evaluate(), isZero(), hasChildren(), setValue()
- CamelCaseNotation for classes, attributes, methods, local variables, parameters
- UPPER\_CASE\_NOTATION for constants
- lowercasenotation for package names



**Martin Fowler**

"Any fool can write code that a computer can understand. Good programmers write code that humans can understand."



**Cory House**

"Code is like humor. When you have to explain it, it's bad."

# Code Documentation

## Comments ...

- in source code are easier to maintain (than in external documents)
- should be written while editing the code
- can be used to generate documentation (e.g., JavaDoc, Doxygen)
- are used to specify classes and public methods (e.g., parameters, exceptions, dependencies)
- document hacks, side effects and unfinished parts (e.g., TODO)
- should not paraphrase the code

### Ryan Campbell

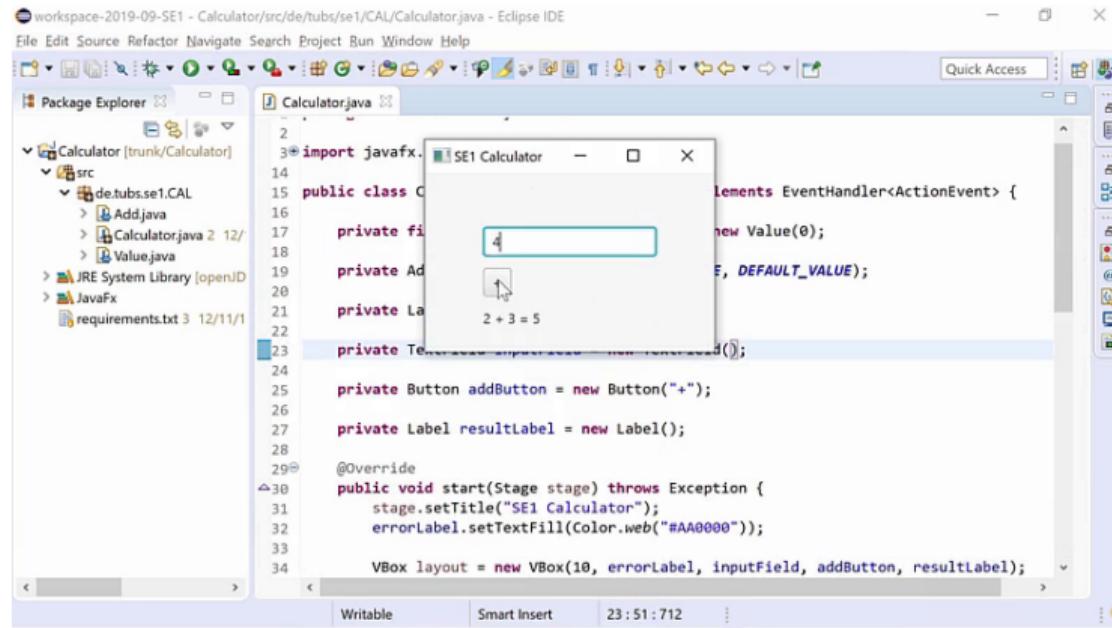
“Commenting your code is like cleaning your bathroom - you never want to do it, but it really does create a more pleasant experience for you and your guests.”



### Steve McConnell

“Good code is its own best documentation. As you’re about to add a comment, ask yourself, “How can I improve the code so that this comment isn’t needed?” Improve the code and then document it to make it even clearer.”

# Excursion: Revising Thomas' Calculator



The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the project structure for "Calculator". It includes a package named "Calculator [trunk/Calculator]" containing a source folder "src" with files: "Add.java", "Calculator.java", and "Value.java". There are also references to "JRE System Library [openJDK]", "JavaFX", and "requirements.txt".
- Code Editor:** The file "Calculator.java" is open. The code implements a JavaFX application with a stage titled "SE1 Calculator". It features an input field ("inputField") and a result label ("resultLabel"). A button (" addButton") is added to a vertical box ("VBox"). The start method sets the stage title and adds the error label, input field, button, and result label to the layout.
- JavaFX Application Window:** A small window titled "SE1 Calculator" is displayed in the center of the screen. It contains an input field with the value "4", a button, and a result label showing "2 + 3 = 5".
- Status Bar:** At the bottom, it shows "Writable", "Smart Insert", the current time "23:51:712", and a help icon.

# Coding Conventions

## Lessons Learned

- Coding conventions ([Programmierrichtlinien](#))
- Formatting, naming, comments, and documentation
- Further Reading: Google 's Java Style Guide

## Practice

- Feel free to inspect the improvements on Github: <https://github.com/tthuem/2020WS-SE-Calculator/commit/28e87e89ff4f87fcc56ecbd5f9d3e310648fb954>
  - The code does not contain any comments.  
<https://github.com/tthuem/2020WS-SE-Calculator/tree/28e87e89ff4f87fcc56ecbd5f9d3e310648fb954/Calculator/src/de/tubs/se1/CAL>
- Where would you add a comment and why?  
Copy one file, add your comment, and upload it to Moodle:  
<https://moodle.uni-ulm.de/mod/moodleoverflow/discussion.php?d=2053>

# Lecture Contents

1. Programming Languages
2. Coding Conventions
  - Understanding the Corona-Warn-App
  - Code Formatting
  - Rules on Naming
  - Code Documentation
  - Excursion: Revising Thomas' Calculator
  - Lessons Learned
3. Tools and Environments

## Tools and Environments

# Computer-Aided Software Engineering

## Terms

A **tool** is an application that supports a particular activity. An **environment** is a collection of related tools. Tools and environments aim at automating some of the activities that are involved in software engineering. The generic term for this field of study is **computer-aided software engineering**.

[adapted from Ghezzi/Jazayeri/Mandrioli]

nano? REAL  
PROGRAMMERS  
USE emacs



HEY. REAL  
PROGRAMMERS  
USE vim.



WELL, REAL  
PROGRAMMERS  
USE ed.



NO, REAL  
PROGRAMMERS  
USE cat.



REAL PROGRAMMERS  
USE A MAGNETIZED  
NEEDLE AND A  
STEADY HAND.



EXCUSE ME, BUT  
REAL PROGRAMMERS  
USE BUTTERFLIES.



THEY OPEN THEIR  
HANDS AND LET THE  
DELICATE WINGS FLAP ONCE.

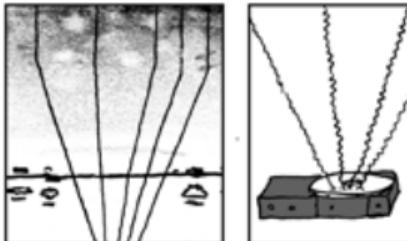


THE DISTURBANCE RIPPLES  
OUTWARD, CHANGING THE FLOW  
OF THE EDDY CURRENTS  
IN THE UPPER ATMOSPHERE.



THESE CAUSE MOMENTARY POCKETS  
OF HIGHER-PRESSURE AIR TO FORM,

WHICH ACT AS LENSES THAT  
DEFLECT INCOMING COSMIC  
RAYS, FOCUSING THEM TO  
STRIKE THE DRIVE PLATTER  
AND FLIP THE DESIRED BIT.



NICE.  
'COURSE, THERE'S AN EMACS  
COMMAND TO DO THAT.  
OH YEAH! GOOD OL'  
C-x M-c M-butterfly...



DAMMIT, EMACS.

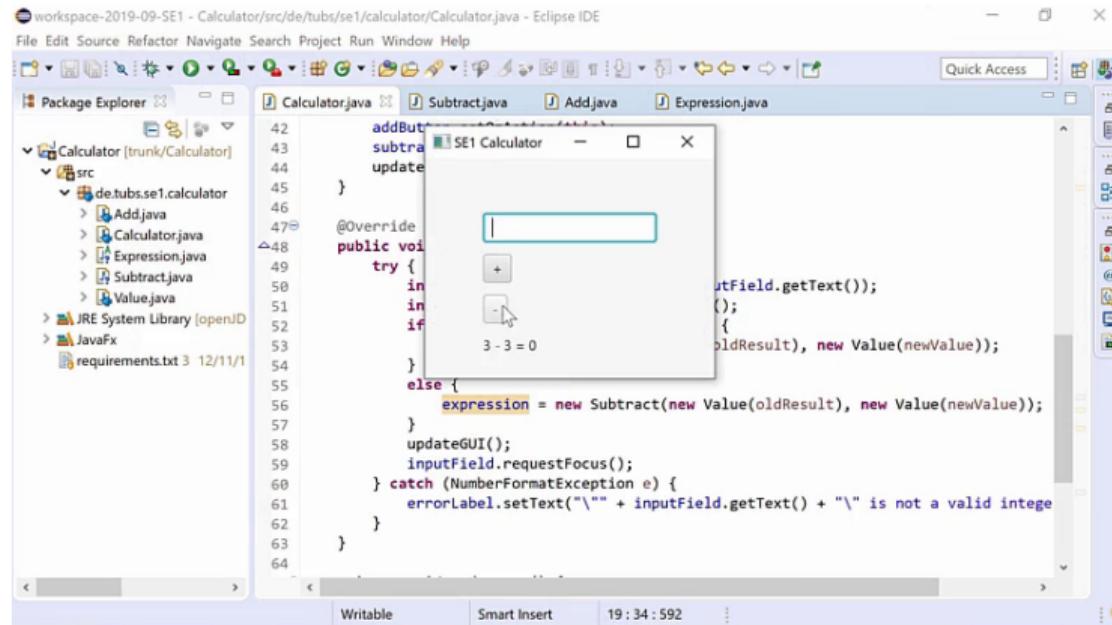
# Overview on Development Tools

## Variety of Tools

[Ghezzi/Jazayeri/Mandrioli]

- text(ual) editors: emacs, vim, ed, Word, ...
- graphical editors: UML editors, Powerpoint, ...
- assembler, compiler, interpreter
- configuration management tools: git, SVN, CVS, ...
- tracking tools (issue trackers): Github, Gitlab, ...
- tools for code navigation and refactoring
- tools for test specification, generation, execution, reporting
- tools for static and dynamic code analysis (e.g., debugger), reverse/reengineering, project management
- integrated development environments (IDEs): Eclipse, IntelliJ, Android Studio, Visual Studio

# Excursion: Extending Thomas' Calculator



# Tools and Environments

## Lessons Learned

- Tool supports by means of tools, environments, and IDEs
- Further Reading: Ghezzi/Jazayeri/Mandrioli, Chapter 9 Software Engineering Tools and Environments

## Practice

Try automated code formatting and rename refactorings in your favorite IDE

# Lecture Contents

1. Programming Languages
2. Coding Conventions
3. Tools and Environments
  - Computer-Aided Software Engineering
  - Overview on Development Tools
  - Excursion: Extending Thomas' Calculator
  - Lessons Learned