# Security Engineering

## Software Development for Safe and Secure Systems

Stefan Lucks

Bauhaus-Universität Weimar

Winter 2018/19

# Information on the WWW

- Slides
- Homework problems, and reading tasks $(*)$, $(**)$
- Code-example
- Links to download further information and tools.

---

$(*)$ It is very important for you to **solve** your homework problems, or at least, **try hard to solve them**! You will mostly learn from doing, rather than from listening!

$(**)$ When you are asked to **read** something on your own (typically available on the web), **please do** so before the next lecture/tutorial!

# Web Resources

- A place to download the **gnat Ada Compiler**, the **SPARK toolset** and **gps**, an IDE for gnat and SPARK: ⟨https://www.adacore.com/community⟩
- The **Ada Programming Wikibook** ⟨https://en.wikibooks.org/wiki/Ada_Programming⟩
- The **Ada Information Clearinghouse** ⟨www.Adaic.org⟩, ⟨www.Adaic.org/learn⟩
- The **Ada 2012 Rationale** ⟨http://www.ada-auth.org/standards/rationale12.html⟩
- **Ada Distilled** ⟨http://www.adaic.org/resources/add_content/docs/distilled/adadistilled.pdf⟩

# What Are Safe and Secure Systems?

$$\text{Sicherheit}_{\text{German}} \approx \text{Safety}_{\text{Engl.}} + \text{Security}_{\text{Engl.}}.$$

**Safety:** The system must run reliably under "normal" and "exceptional" circumstances.

**Security:** The system defends itself against malicious manipulations and attacks.

**Exceptionally high assurance required:**

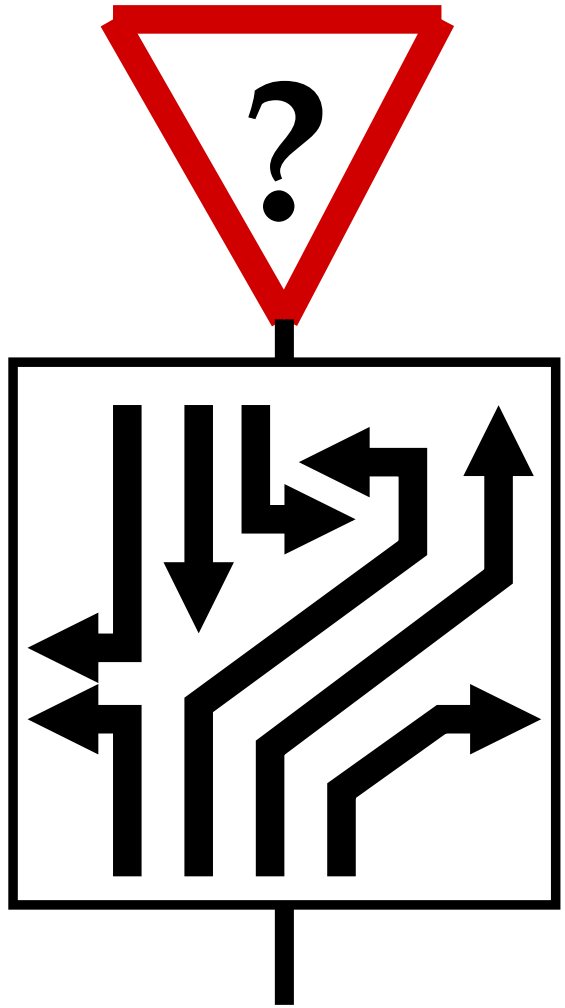
- If the system fails, people will be killed. (Boeing, Airbus,...).
- A system failure could be *very* expensive. (Electronic banking, unmanned satellites, ...)

However, safety and security concerns are becoming omnipresent, today. Even for standard internet tools, such as a **provably secure Domain Name Server**:
⟨`http://ironsides.martincarlisle.com/`⟩.

# What Will You Learn in This Course?

## Methods and Tools to Develop Safe and Secure Systems

- the Ada programming language
- systematic tool-based testing
- design by contract
- the theory of static verification
- practical static verification with SPARK
- failure-tolerant distributed systems



# The sorid reality

If it's switched on
and stops working
probably
the fault is in the software.

If you switch it off and on again,
and it now works again,
certainly
the fault is in the software.

Whatever **it** is!

Tony Hoare

If it is connected to the outside,
someone will attempt
to break in
and probably succeed.

Even if withstands the attack,
it likely sends its data
to some place
where the attacker can break in.

Whatever **it** is!

Stefan Lucks

# Is it impossible to write secure software that does not need a security fix every other day?

# After the Course, You Should...

- ... understand that software development does not *need* to be as poor as it usually is,
- ...know methods and tools to develop safe and secure software (and the prize for their usage),
- ...and understand how to to use such methods and tools.

# Example: Binary Search

Given:

- array $A$ with $n$ items
- for all $i$, $i+1$ in range: $A[i] \leq A[i+1]$
- value $x$

Required: $i$ with $A[i] = x$ or "not found"

Time: $\Theta(\log n)$

Example:

- $A$ =[a b d d e f g h j m n n o p p p q x y z]
- search for c, q, r, ...

"Only 10% of all programmers can implement binary search properly."

# Can you find some of the Errors here?

**Read: Richard Patts, “Textbook Errors in Binary Searching”**

```
PROCEDURE BinarySearch (A          : anArray,
                        Size       : anArraySize,
                        Key        : INTEGER,
                        VAR Found  : BOOLEAN;
                        VAR Index  : anArrayIndex);
Var Low, High : anArrayIndex;
BEGIN
   LOW := 1;
    High := Size;

   REPEAT
       Index := (Low + High) DIV 2;
       If Key < A[Index]
          THEN High := Index - 1
          ELSE Low  := Index + 1
   UNTIL (Low > High) OR (Key = A[Index]);

   FOUND := (Low <= High)
END;
```

# Outlook

You will learn some methods and tools

- to *specify* what an implementation is supposed to do,
- to *discover* such errors, and
- to *verify* the correctness of an implementation.

Other topics include

- concurrency,
- failure tolerance and
- distributed systems.

# A bug found in the JDK, discovered after 9 years

```
public static int binarySearch(int[] a, int key) {
  int low = 0;
  int high = a.length - 1;

  while (low <= high) {
    int mid = (low + high) / 2;
    int midVal = a[mid];

    if (midVal < key)
      low = mid + 1
    else if (midVal > key)
      high = mid - 1;
    else
      return mid; // key found
  }
  return -(low + 1);  // key not found.
}
```