# Deductive Program Verification: Mature Enough to be Taught to Software Engineers?

**Marc Schoolderman**, Sjaak Smetsers, Marko van Eekelen
18 November 2019

Institute for Computing and Information Sciences
Radboud University

# A short exercise in epistemology

What is the correct answer?

$$
\begin{aligned}
14627333968688430831 \times 18369543843582177293 \\
&= 268697452652541021050625400954609320483\ ? \\
&= 268697426686324666285319976061674831483\ ?
\end{aligned}
$$

# Computers do make mistakes

**1994:** Intel Pentium: `FDIV` instruction infamously "flawed"

$$\frac{4195835}{3145727} = 1333820449136241002 \ldots \text{ or } 1333739068902037589 \text{ ?}$$

**1995:** AMD K5: `FDIV` instruction verified using formal methods

- Division microcode complies with IEEE-754 standard
  - *Proof by:* J Strother Moore, T. Lynch, M. Kaufmann
- ACL2 interactive theorem prover
  - *Developed by:* J Strother Moore, M. Kaufmann

# Does this compute $X_{64} \times Y_{64}$?

**2015:** High-performance multiplication for AVR microcontrollers (*M. Hutter, P. Schwabe,2015*)
**2017:** Verified using Why3 proof framework (*M. Schoolderman,2017*)

```
clr r20
clr r21
movw r16, r20
ld r2, X+
ld r3, X+
ld r4, X+
ld r5, X+
ldd r6 Y+0
ldd r7 Y+1
ldd r8 Y+2
ldd r9 Y+3
mul r2, r8
movw r12, r0
mul r2, r6
movw r10, r0
mul r2, r7
add r11, r0
adc r12, r1
adc r13, r21
mul r3, r9
movw r14, r0
mul r2, r9
movw r18, r0
mul r3, r6
add r11, r0
adc r12, r1
adc r13, r18
adc r19, r21
mul r3, r7
add r12, r0
adc r13, r1
adc r19, r21
mul r4, r9
add r14, r19
adc r15, r0
adc r16, r1
mul r4, r8
movw r18, r0
mul r4, r6
add r12, r0
adc r13, r1
adc r14, r18
adc r19, r21
mul r3, r8
add r13, r0
adc r14, r1
adc r19, r21
mul r5, r9
add r15, r19
adc r16, r0
adc r17, r1
mul r5, r7
movw r18, r0
mul r4, r7
add r13, r0
adc r18, r1
adc r19, r21
mul r5, r6
add r13, r0
adc r18, r1
adc r19, r21
mul r5, r8
add r14, r18
adc r0, r19
adc r1, r21
add r15, r0
adc r16, r1
adc r17, r21
ldd r22 Y+4
ldd r23 Y+5
ldd r24 Y+6
ldd r25 Y+7
movw r28, r20
ld r18, X+
ld r19, X+
ld r20, X+
ld r21, X+
movw r26, r28
std Z+0, r10
std Z+1, r11
std Z+2, r12
std Z+3, r13
sub r2, r18
sbc r3, r19
sbc r4, r20
sbc r5, r21
sbc r0, r0
sub r6, r22
sbc r7, r23
sbc r8, r24
sbc r9, r25
sbc r1, r1
eor r2, r0
eor r3, r0
eor r4, r0
eor r5, r0
eor r6, r1
eor r7, r1
eor r8, r1
eor r9, r1
sub r2, r0
sbc r3, r0
sbc r4, r0
sbc r5, r0
sub r6, r1
sbc r7, r1
sbc r8, [illegible]
sbc [illegible]
eor [illegible]
bst [illegible]
mul [illegible]
add [illegible]
adc [illegible]
adc [illegible]
adc [illegible]
mul [illegible]
add [illegible]
adc r[illegible]
adc r2[illegible]
mul r19, r22
add r15, r0
adc r16, r1
adc r17, r29
adc r28, r26
mul r18, r24
add r16, r0
adc r17, r1
adc r28, r26
mul r19, r23
add r16, r0
adc r17, r1
adc r28, r26
mul r20, r22
add r16, r0
adc r17, r1
adc r28, r26
clr r29
mul r18, r25
add r17, r0
adc r28, r1
adc r29, [illegible]
mul r19[illegible]
add r[illegible]
[illegible]
[illegible]
[illegible]3
[illegible] r0
[illegible] r1
[illegible]9, r26
[illegible]r21, r22
[illegible]d r17, r0
adc r28, r1
adc r29, r26
mul r19, r25
movw r18, r26
add r28, r0
adc r29, r1
adc r18, r26
mul r20, r24
add r28, r0
adc r29, r1
adc r18, r26
mul r21, r23
add r28, r0
adc r29, r1
adc r18, [illegible]
mul r20[illegible]
add r[illegible]
adc [illegible]
ad[illegible]
[illegible]4
[illegible]r0
[illegible], r1
[illegible]19, r26
[illegible]r21, r25
[illegible]d r18, r0
adc r19, r1
mul r2, r6
movw r20, r0
movw r22, r26
mul r2, r7
add r21, r0
adc r22, r1
mul r3, r6
add r21, r0
adc r22, r1
adc r23, r26
movw r24, r26
mul r2, r8
add r22, r0
adc r23, r1
adc r24, r26
mul r3, r7
add r22, r0
adc r23, r1
a[illegible] r24, r26
[illegible] r4, r6
[illegible] r22, r0
[illegible] r23, r1
[illegible] r24, r26
[illegible]l r2, r9
add r23, r0
adc r24, r1
adc r25, r26
mul r3, r8
add r23, r0
adc r24, r1
adc r25, r26
mul r4, r7
add r23, r0
adc r24, r1
adc r25, r26
mul r5, r6
add r23, r0
adc r24, r1
adc r25, r26
mul r3, r9
movw r2, r26
add r24, r0
adc r25, r1
adc r2, r27
mul r4, r8
add r24, r0
adc r25, r1
adc r2, r27
mul r5, r7
add r24, r0
adc r25, r1
adc r2, r27
mul r4, r9
add r25, r0
adc r2, r1
adc r3, r27
mul r5, r8
add r25, r0
adc r2, r1
adc r3, r27
mul r5, r9
add r2, r0
adc r3, r1
add r10, r14
adc r11, r15
adc r12, r16
adc r13, r17
adc r14, r28
adc r15, r29
adc r16, r18
adc r17, r19
bld r27, 0
dec 27
adc r26, r27
mov r0, r26
asr r0
eor r20, r27
eor r21, r27
eor r22, r27
eor r23, r27
eor r24, r27
eor r25, r27
eor r2 , r27
eor r3 , r27
adc r10, r20
adc r11, r21
adc r12, r22
adc r13, r23
adc r14, r24
adc r15, r25
adc r16, r2
adc r17, r3
adc r28, r26
adc r29, r0
adc r18, r0
adc r19, r0
std Z+4, r10
std Z+5, r11
std Z+6, r12
std Z+7, r13
std Z+8, r14
std Z+9, r15
std Z+10, r16
std Z+11, r17
std Z+12, r28
std Z+13, r29
std Z+14, r18
std Z+15, r19
```

## How difficult is program verification, really?

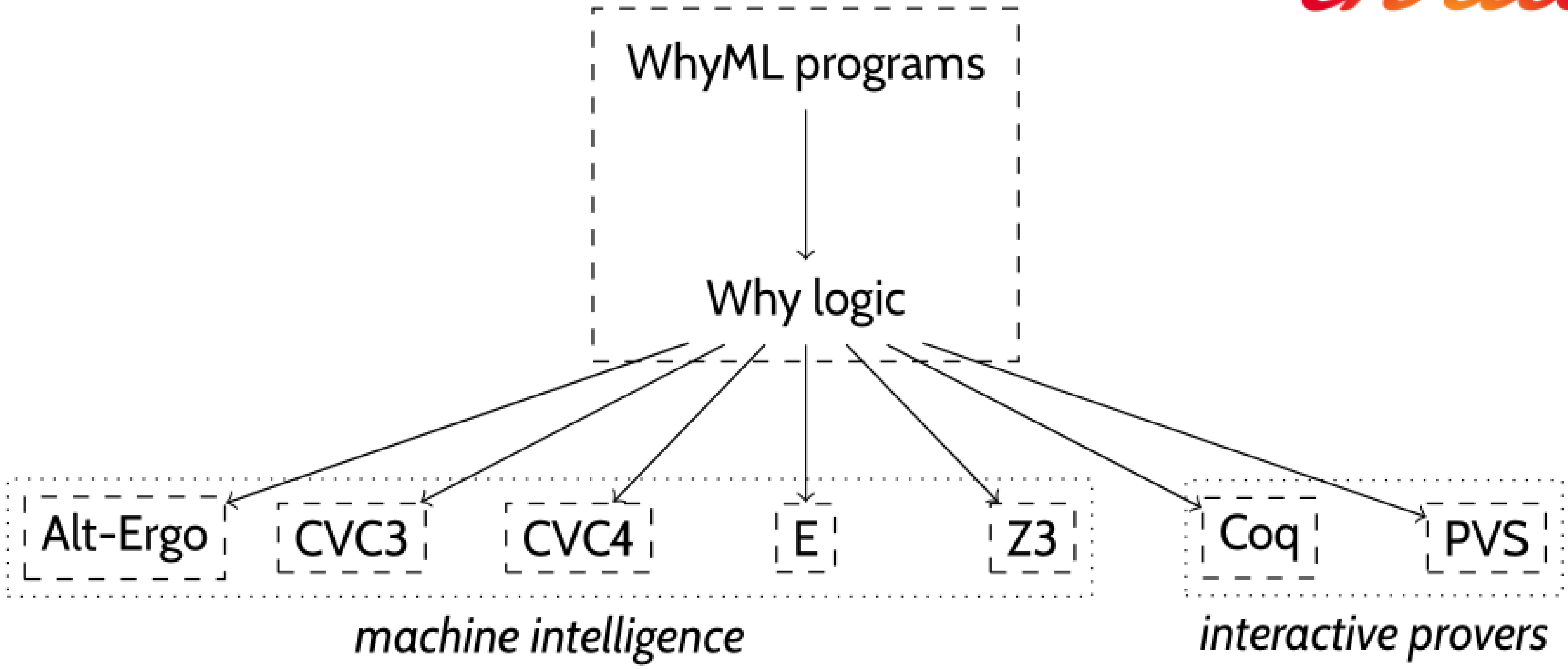
Experiences from (trying to) do research in cooperation with industry:

- Gap between industry and academia
- Tools used by academics (or are perceived to be) too esoteric:
  - *"Show me something an <u>educated software engineer</u> could use."*

***For wider adaption, this gap needs to be bridged.***

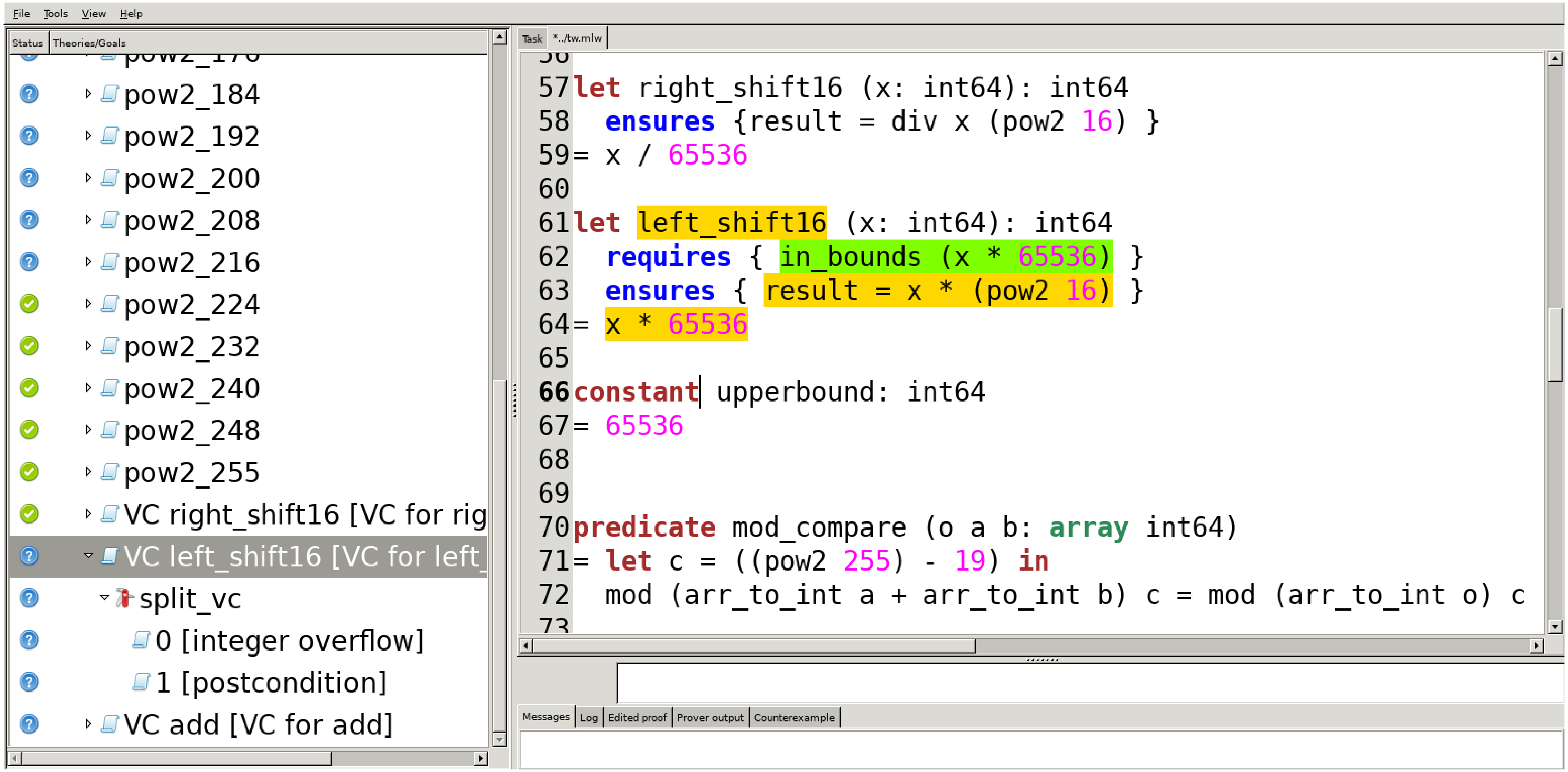
# Overview of the Why3 Verification Platform

*(J.C. Filliâtre, F. Bobot, C. Marché, G. Melquiond, A. Paskevich)*



# Overview of the Why3 Verification Platform

*(J.C. Filliâtre, F. Bobot, C. Marché, G. Melquiond, A. Paskevich)*



## Participants in the course *"Software Analysis"*

**22** actively participating students

- Possess a Bachelor degree: university/vocational university (HBO)
  - *Similar to many junior software engineers?*
- Expected: little/no experience in formal verification
  - Students from Radboud have seen some model checking, and pen-and-paper Hoare logic
  - JML used for 1 short exercise in parallel course

- **Goal: evaluate Why3**
  - **Learning by doing, teamwork, open problems.**

# Course structure

- **Lectures** (6 hours)
  1) Motivation for verification, introduction of Why3
  2) Why3 data type system
  3) Techniques to work around "stuck" proof efforts
  4) WhyML as a modelling language

- **Small exercises** (20 hours)
  - Supporting the lectures, formative feedback

- **Verification task** (24-36 hours)
  - Report + evaluation of Why3

# Case study 1: safe string concatenation (taken from CloudLibc)

```
size_t strlcat(char *restrict s1, const char *restrict s2, size_t n) {
  size_t skipped = 0;
  while (n > 0 && *s1 != '\0') {
    ++s1;
    --n;
    ++skipped;
  }
  const char *begin = s2;
  while (n > 1) {
    *s1 = *s2;
    if (*s2 == '\0')
      return s2 - begin + skipped;
    ++s1;
    ++s2;
    --n;
  }
  if (n > 0)
    *s1 = '\0';
  while (*s2 != '\0')
    ++s2;
  return s2 - begin + skipped;
}
```

*Challenges*:

- Arguments must not alias
- "Safety valve" prevents out-of-bounds access, but breaks the naive contract.
- Null-terminated strings

# Case study 2: a routine inspired by “TweetNaCl”

```
void add(int64_t o[16], int64_t a[16], int64_t b[16])
{
    // add limbs
    for(int i=0; i<16; i++) {
        o[i] = a[i] + b[i];
    }

    // carry propagation
    for(int i=0; i<15; i++) {
        int64_t c = o[i] >> 16;
        o[i+1] = o[i+1] + c;
        o[i] = o[i] - (c << 16);
    }

    // reduce mod 2^255 - 19
    int64_t c = o[15] >> 16;
    int64_t t = 38*c;
    o[0] = o[0] + t;
    o[15] = o[15] - (c<<16);
}
```

***Challenges*:**

- Weird representation of integers
- Unspecified what this is supposed to do
- The third comment **lies**

## Case study 1 results:

- 7 teams were successful
  - Formal specification of `strlcat`, verified.
- 2 teams ran into difficulties
  - Likely cause: poor initial modelling choices

## Case study 2 results:

- 1 team: verified mathematical correctness of `add`
- 1 team: proved that the final loop is seldom necessary
  - both teams quickly proved absence of signed integer overflow, and discovered known flaws in this code

## Subjective observations

- Most effort required: modelling C code in WhyML
  - *Also due to unfamiliarity with C...*

- Students can deal with concepts unique to Why3
  - E.g. *"proof context size", "ghost code", "logic functions"*

- Student remarks in reports (paraphrased):
  - *Positive*: "Why3 is intuitive and gives strong guarantees."
  - *Negative:* "This was very time-consuming."
  - *Verdict*: "Useful for safety-critical software, overkill elsewhere."

# Evaluation of Why3 by students

**Common theme: more attention to user-friendliness!**

- Error messages should be more helpful
- Better counterexample generation
- Automatic warnings about logical inconsistencies
- Single-step integrated debugger
- “Trivial” loop invariants should be generated automatically
- Online support community (e.g. *StackExchange*)
- Usable on Windows instead of only Linux and MacOS

## Research questions:

- *Are these students really representative for software engineers?*
- *How much time did the students need, quantitatively?*
- *Do the reports contain honest evaluations?*

## Anonymous research survey

**15** respondents out of **22** active students: ***68% response ratio***

**Limitations:**

- No open questions
- Not enough to do statistics on

**Primary goal:**

- Test assumptions about student population
- Validate the evaluation in the report

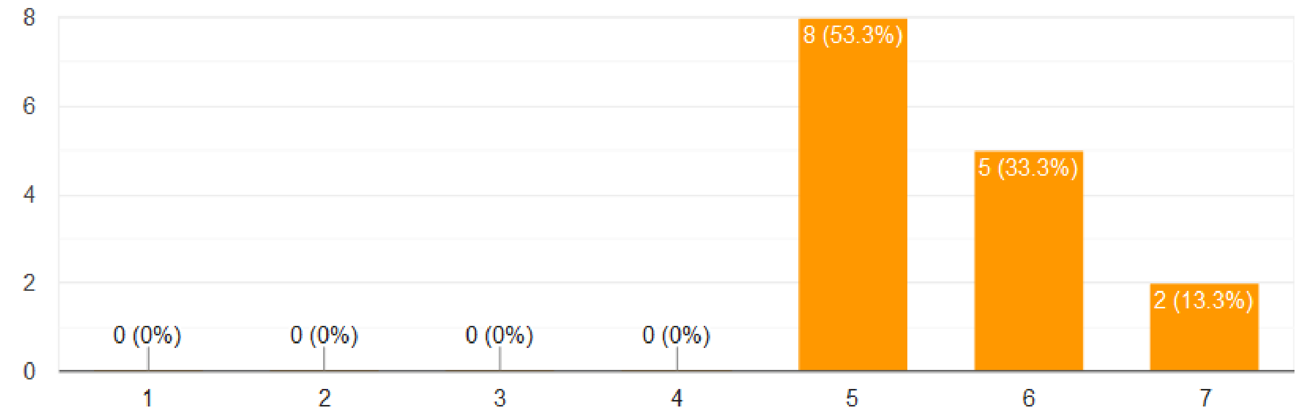
# Anonymous research survey: results

## Student background

- Consider themselves fairly skilled programmers
- Clearly not "pure logicians"
- Only two students report taking more than 30EC of math/logic courses

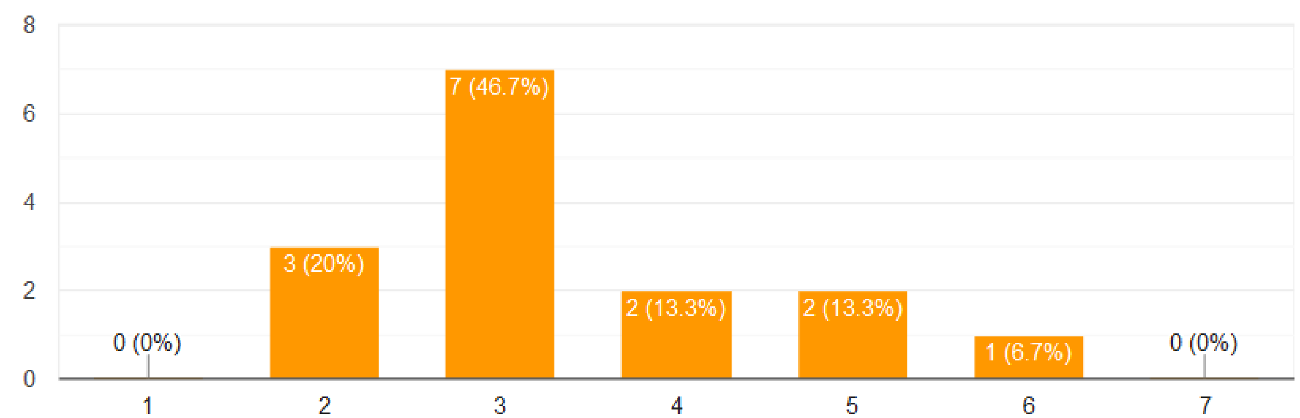
How skilled would you say you are in programming?

15 responses




How skilled would you say you are in mathematics?

15 responses

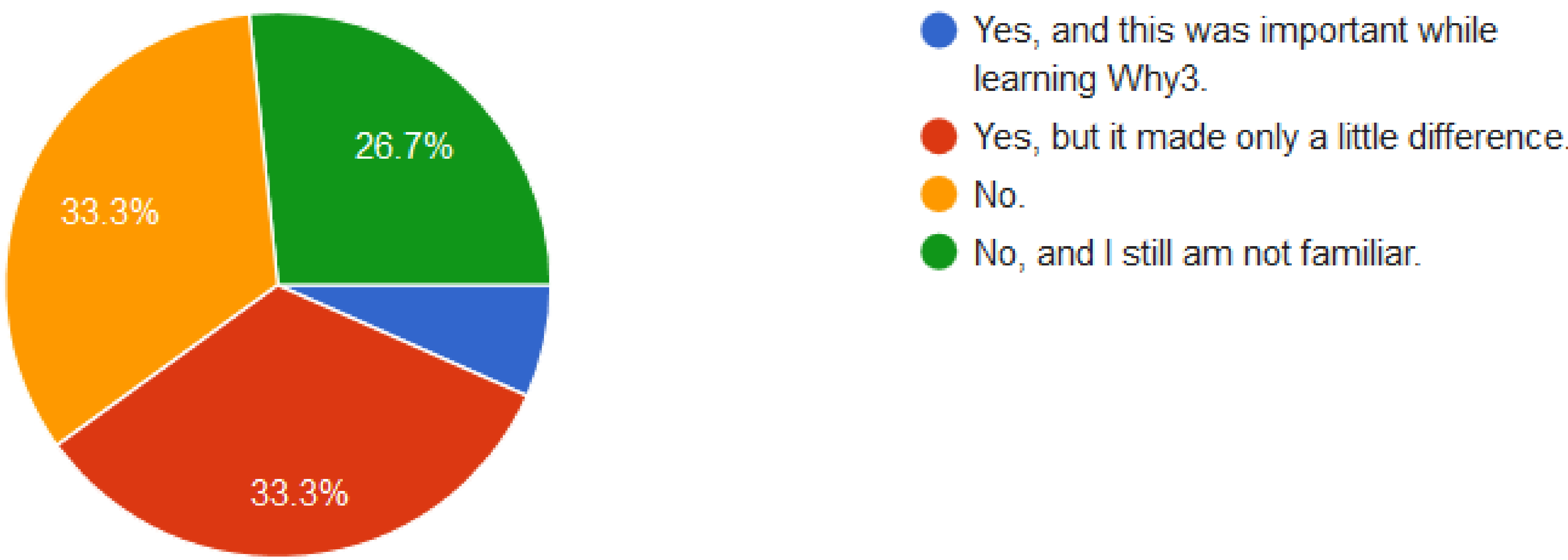


# Anonymous research survey: results

## 60% of students did not know what ‘Hoare logic’ meant!

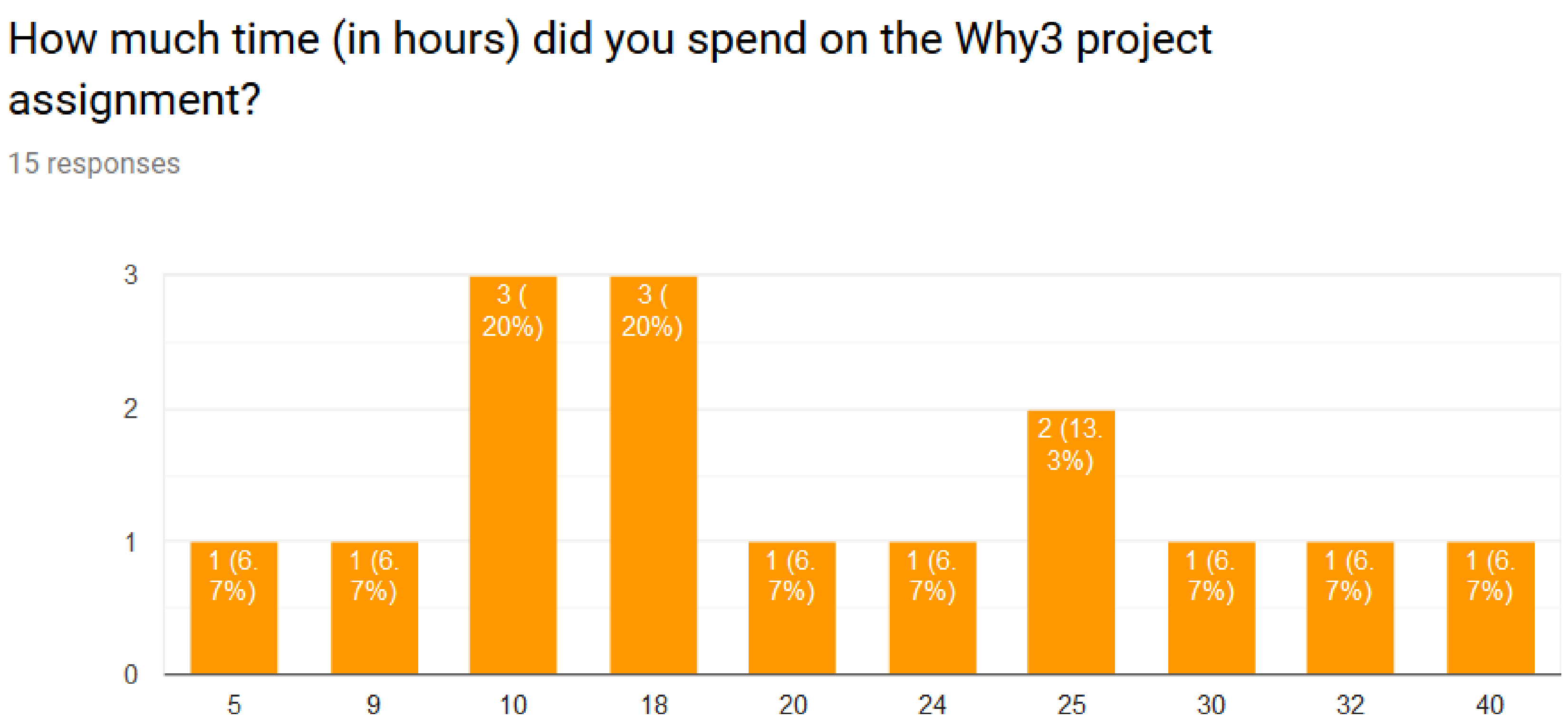
Were you familiar with Hoare logic before taking this course?

15 responses



# Anonymous research survey: results

## Time spent on project:




## Worst case scenario:

- Assume: all 6 students that were not successful are in this dataset
- Then: successful *teams* needed on average ≤26 hours

# Anonymous research survey: results

## What was it like for students?

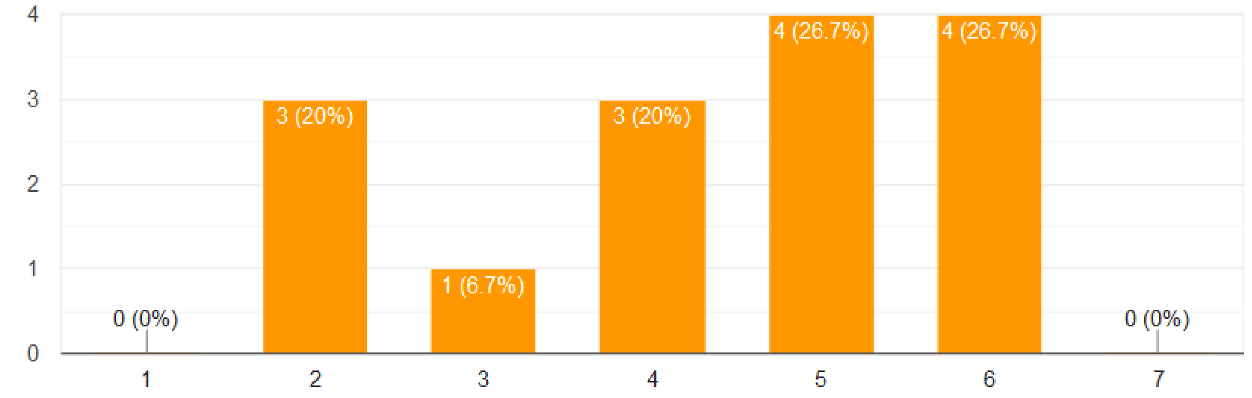
- "Somewhat harder" and more mathematical than ordinary programming

## Hardest activities:

1) Finding loop invariants
2) Modelling a C program
3) Writing a formal specification

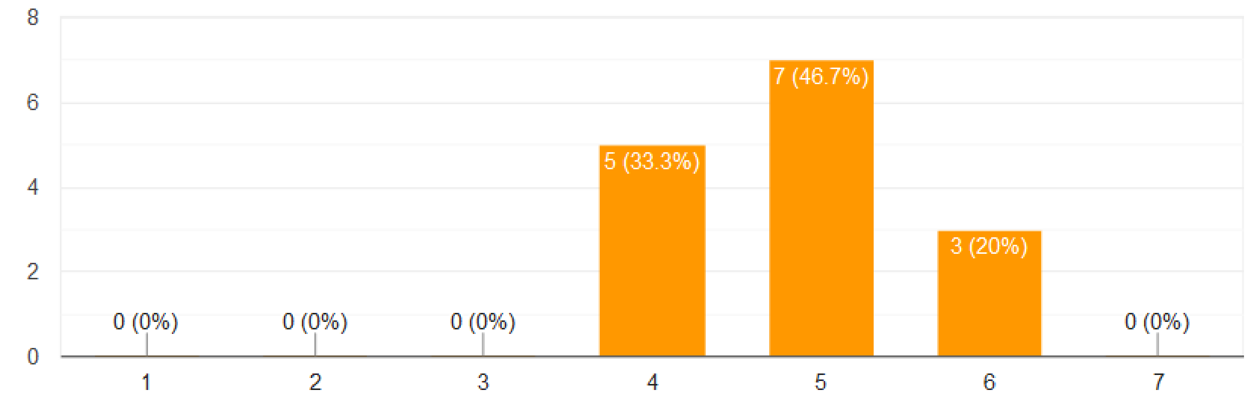
Did verification with Why3 feel more like a programming activity, or like a mathematical activity?

15 responses




How would you compare learning Why3 with learning a new programming language?

15 responses



# Anonymous research survey: results

**What do students think about applying formal methods?**

*Formal methods are appropriate for:*

Small security-critical libraries, programming language design

*No clear consensus:*

Self-driving cars, compilers, operating systems

*Formal methods are not appropriate for:*

Smartphone apps

► **Consistent with the non-anonymous reports**

## Conclusion

**Novices can apply Why3 usefully in a short amount of time**

- Verifying <u>small but real</u> program code
- Four weeks of training, ~26 hours of work
- Background: comparable to junior software engineers

**Formal tools can benefit from a fresh perspective**

- Problem may be *usability* instead of inherent difficulty

**Thank you for your attention!**