Software quality and formal methods: Hoare/Dijkstra approach

Dr. Ivan S. Zapreev

Software Quality

Programming Languages

Formal Verification

Frama - C platform

Verification in practice

Concluding remarks

Software quality and formal methods: Hoare/Dijkstra approach

Dr. Ivan S. Zapreev

Neat Software Designs

2020-01-24

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Global Outline:

- Software Quality
- Programming Languages
- Formal Verification
- Frama - C platform
- Verification in practice
- Concluding remarks

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software Quality

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Verification

Frama - C
platform

Verification in
practice

Concluding
remarks

Software Quality: Outline

- Motivating examples
- Software Development
- Software Verification

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Verification

Frama - C
platform

Verification in
practice

Concluding
remarks

Motivating examples: Therac-25

Years 1985–1987: Radiation overdose

- Control software flaw:
  - Race conditions
- Death of 6 (six) cancer patients

Figure 1: Radiation therapy

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Verification

Frama - C
platform

Verification in
practice

Concluding
remarks

Motivating examples: Ariane-5

Year 1996: Missile crash

- Control software flaw:
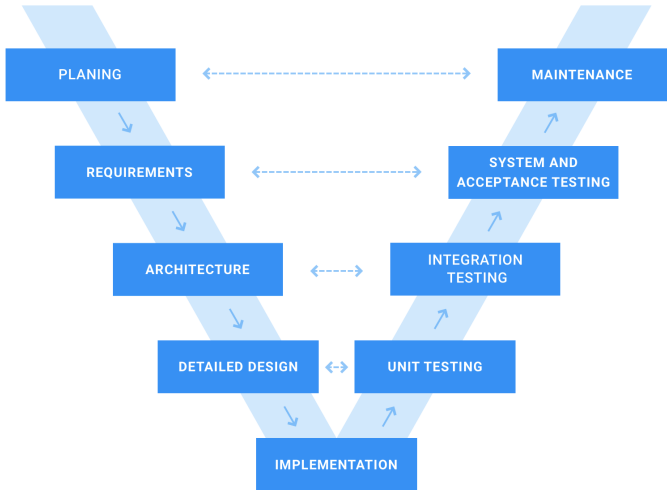 - 64-bit float to 16-bit integer conversion
- $137 million Rocket + $500 million cargo

Figure 2: Space flights

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Verification

Frama - C
platform

Verification in
practice

Concluding
remarks

Motivating examples: Toyota

Year 2005: Sudden unintended acceleration

- Control software flaw:
  - Recursion causing stack overflow
- 89 deaths and 57 injuries
- $1.2 billion compensations

Figure 3: Toyota Camry

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Verification

Frama - C
platform

Verification in
practice

Concluding
remarks

Motivating examples: Plenty More

The 12 Software Bugs That Caused Epic Failures: <link>

Software quality and formal methods: Hoare/Dijkstra approach

Dr. Ivan S. Zapreev

Software Quality

Programming Languages

Formal Verification

Frama - C platform

Verification in practice

Concluding remarks

Software Development: V-model

Figure 4: Software development process

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Verification

Frama - C
platform

Verification in
practice

Concluding
remarks

Software Development: V & V

Software **Validation** and **Verification**.

Formally defined by the International Organization for Standardization, see ISO-9000:2015:

- **Verification** – *"Confirmation, through the provision of objective evidence, that specified requirements have been fulfilled."*

- **Validation** – *"Confirmation, through the provision of objective evidence, that the requirements for a specific intended use or application have been fulfilled."*

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Verification

Frama - C
platform

Verification in
practice

Concluding
remarks

Software Development: Testing

- **Verification**:
    - Are we building the product right?
    - Does the system comply with its specification?
- **Validation**:
    - Are we building the right product?
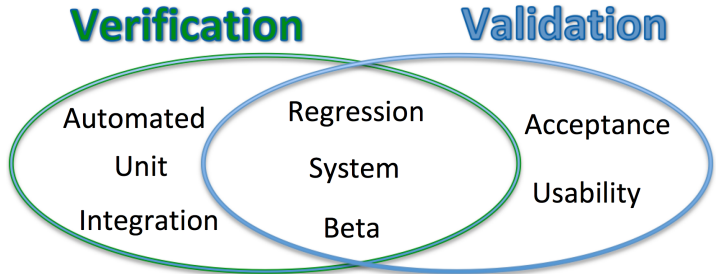   - Does the system meet the needs of the customer?

Figure 5: Types of testing

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Verification

Frama - C
platform

Verification in
practice

Concluding
remarks

Software Verification: Formal

Formal verification[1]:

> "Is the act of proving or disproving the correctness of intended algorithms underlying a system with respect to a certain formal specification or property, using formal methods of mathematics."

Formal methods[2]:

> "Formal methods are techniques used to model complex systems as mathematical entities."

> "By building a rigorous model of a complex system, it is possible to verify the system's properties in a more thorough fashion than empirical testing."

---

[1] "Formal Verification" on Wikipedia

[2] "Formal Methods", Michael Collins, CMU

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Programming Languages

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Verification

Frama - C
platform

Verification in
practice

Concluding
remarks

Programming Languages: Outline

- Language generations
- Declarative vs. Imperative
- What is ANSI-C?

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Language generations
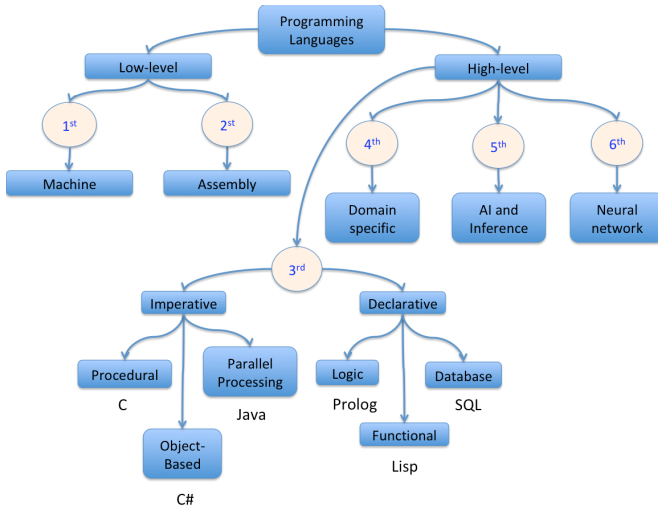
Figure 6: Generations of Programming languages

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Verification

Frama - C
platform

Verification in
practice

Concluding
remarks

Declarative vs. Imperative: Main

Consider a problem of multiplying all array elements by 2:

- **Declarative** – Specifies **what** to achieve:

```
//Declarative `JavaScript`
var arr_dbl = arr.map((x) => x * 2)
```

- **Imperative** – Defines the **how** steps:

```
//Imperative `JavaScript`
var arr_dbl = []
for (let i = 0; i < arr.length; i++) {
  arr_dbl.push(arr[i] * 2)
}
```

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Verification

Frama - C
platform

Verification in
practice

Concluding
remarks

Declarative vs. Imperative: Test
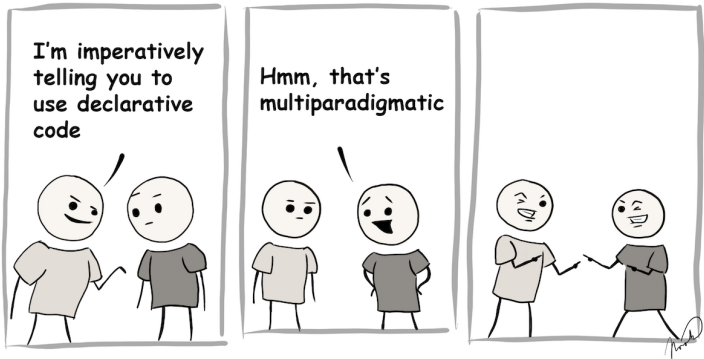
Figure 7: If you laugh, it means you've passed

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Verification

Frama - C
platform

Verification in
practice

Concluding
remarks

What is ANSI-C: Old C

C language:

C is an *imperative procedural* language.

Procedural language:

Is an imperative language in which the program is built from one or more subroutines commonly known as functions.

Defining ANSI-C:

ANSI-C is a common name for two equivalent standards:

- C89 – American National Standards Institute (ANSI)
- C90 – International Organization for Standardization (ISO)

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Formal Verification

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Formal Verification: Outline

- Verification goal
- Hoare Approach
- Dijkstra Extension

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Verification

Frama - C
platform

Verification in
practice

Concluding
remarks

Verification goal: Global

A *program* shall satisfy a formal specification of its *behavior*.
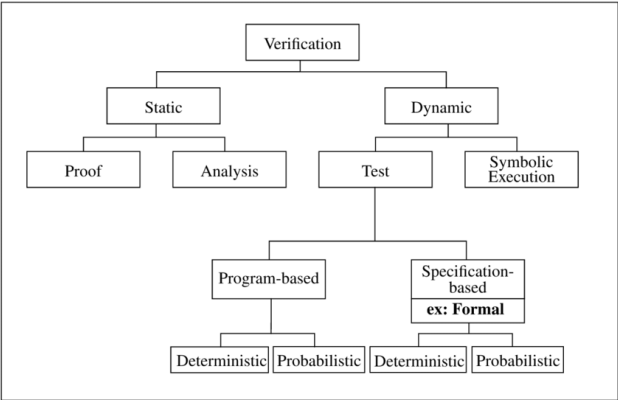
Figure 8: Verification methods

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Verification

Frama - C
platform

Verification in
practice

Concluding
remarks

Verification goal: Proving

Mathematically prove conformance to formal specifications.
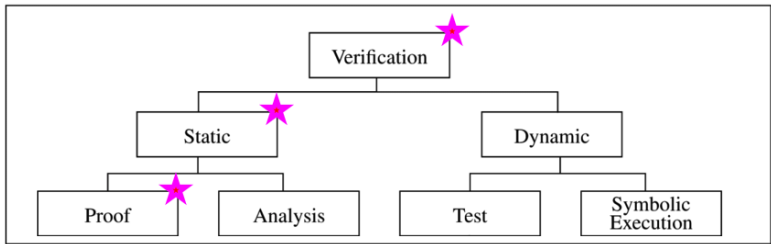
Figure 9: Formal correctness proving

Consider:

Hoare/Dijkstra approach for proving correctness of *imperative programs*.

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Verification

Frama - C
platform

Verification in
practice

Concluding
remarks

Hoare Approach[3]

Hoare triples: $\{P\}\,C\,\{Q\}$

C - code; P - pre-condition; Q - post-condition;

Axioms, e.g. *Skip* and *Assign*:

$$\frac{-}{\{P\}skip\{P\}} \quad \text{and} \quad \frac{-}{\{P[E/V]\}V:=E\{P\}}$$

Where E is any expression and V is any variable.

Inference rules, e.g. *Composition* and *Conditional*:

$$\frac{\{P\}S_1\{R\}, \{R\}S_2\{Q\}}{\{P\}S_1;\,S_2\{Q\}} \quad \text{and} \quad \frac{\{B \wedge P\}S\{Q\}, \{\neg B \wedge P\}T\{Q\}}{\{P\} \textbf{ if } B \textbf{ then } S \textbf{ else } T \textbf{ elseif } \{Q\}}$$

Partial correctness: If P holds before executing C then Q holds afterwards, ONLY if C terminates.

---

[3]"An Axiomatic Basis for Computer Programming", Tony Hoare, 1969.

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Verification

Frama - C
platform

Verification in
practice

Concluding
remarks

Dijkstra Extension[4]

The *weakest precondition calculus* for

- A predicate transform semantics to mechanize the proofs.
- Explains how C transforms P into Q.

Backward reasoning:

- Based on Q and C calculate the *weakest pre-condition* $\widehat{P}$
- If $P \implies \widehat{P}$, then the proof is complete

Forward reasoning:

- Based on P and C calculate the *strongest post-condition* $\widehat{Q}$
- If $\widehat{Q} \implies Q$, then the proof is complete

---

[4]"Guarded commands, non-determinacy and formal derivation of
programs", Edsger Dijkstra, 1975

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Frama - C platform

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Frama - C platform: Outline

- Platform description
- Plugins overview
- What is ACSL?

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Verification

Frama - C
platform

Verification in
practice

Concluding
remarks

Platform description

Frama-C is a:

- Plug-in-based
- Open-source
- Cross-platform

framework for ANSI-C source-code analysis:

- Browse unfamiliar code
- Static code analysis
- Dynamic code analysis
- Code transformations
- Certification of critical software

You can easily build upon the existing plug-ins to implement your own analysis.

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Verification

Frama - C
platform

Verification in
practice

Concluding
remarks

Plugins overview: Main
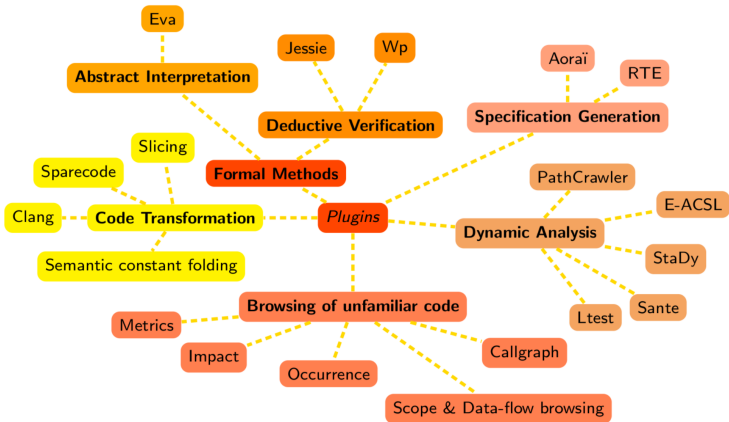
Figure 10: Frama-C plugins

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Verification

Frama - C
platform

Verification in
practice

Concluding
remarks

Plugins overview: WP

WP – *weakest precondition* for ACSL specs of ANSI-C programs.
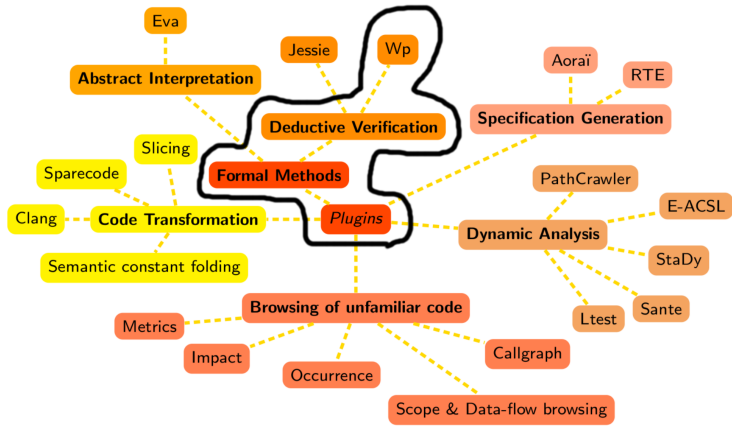
Figure 11: Frama-C WP plugin

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Verification

Frama - C
platform

Verification in
practice

Concluding
remarks

What is ACSL: General

In short:

- ACSL – ANSI/ISO C Specification Language
- Allows to formally specify properties of C programs

It is all about function contracts:

```
/*@ ensures \result >= x && \result >= y;
    ensures \result == x || \result == y;
 */
int max (int x, int y) {
  return(x > y) ? x : y;
}
```

A function contract is a combination of:

- post-conditions - ensures
- pre-conditions - requires

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Verification

Frama - C
platform

Verification in
practice

Concluding
remarks

What is ACSL: Pointers

ACSL allows to reason about, e.g.:

- Pointers
- Arrays
- Termination

Consider pointers:

```
/*@ requires \valid(x) && \valid(y);
    ensures *x <= *y;
 */
void max_ptr (int *x, int *y) {
  if(*x > *y) {
    int tmp = *x;
    *x = *y;
    *y = tmp;
  }
}
```

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Verification

Frama - C
platform

Verification in
practice

Concluding
remarks

What is ACSL: Completeness

Is the following `max_ptr` implementation correct?

```
/*@ requires \valid(x) && \valid(y);
    ensures *x <= *y;
 */
void max_ptr (int *x, int *y) {
  *x = *y = 0;
}
```

We need to make our specification **complete**:

```
/*@ requires \valid(x) && \valid(y);
    ensures *x <= *y;
    ensures (*x == \old(*x) && *y == \old(*y)) ||
            (*x == \old(*y) && *y == \old(*x));
 */
void max_ptr(int *x, int *y);
```

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Verification

Frama - C
platform

Verification in
practice

Concluding
remarks

What is ACSL: The whole spec.

The complete ACSL specification *v1.4* has 93 pages:
https://frama-c.com/download/acsl_1.4.pdf

Figure 12: Feel free to explore

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Verification in practice

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Verification in practice: Outline

- Verification Examples
- Verification Outcomes
- Experience summary

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Verification

Frama - C
platform

Verification in
practice

Concluding
remarks

Verification Examples: abs(.)

Consider a primitive integer absolute value computation:

```c
/*@
    ensures \result >= 0;
 */
int abs(int val) {
  if(val < 0) return -val;
  return val;
}
```

The verification shall return **OK**, right?

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Verification

Frama - C
platform

Verification in
practice

Concluding
remarks

Verification Examples: Issue #1

NOP - the verification results are inconclusive:

Figure 13: WP detects a possible overflow

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Verification

Frama - C
platform

Verification in
practice

Concluding
remarks

Verification Examples: Issue #2

Extend the specification with a pre-condition:

```
/*@ requires INT_MIN < val;
    ensures \result >= 0;
 */
int abs(int val) {
  if(val < 0) return -val;
  return val;
}
```

The verification is **OK**, but the spec is NOT complete:

```
/*@ requires INT_MIN < val;
    ensures \result >= 0;
 */
int abs(int val) {
  return 1;
}
```

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Verification

Frama - C
platform

Verification in
practice

Concluding
remarks

Verification Examples: Final?

An explicit \result value specification makes it complete:

```
/*@ requires INT_MIN < val;
    ensures (val == 0 ==> \result == 0) &&
            (val > 0 ==> \result == val) &&
            (val < 0 ==> \result == -val);
 */
int abs(int val) {
  if(val < 0) return -val;
  return val;
}
```

What if the implementation was wrong?

Would we be able to identify the root-cause?

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Verification

Frama - C
platform

Verification in
practice

Concluding
remarks

Verification Examples: Faulty

Consider a lengthy and potentially buggy implementation:

```
/*@ requires INT_MIN < val;
    ensures (val == 0 ==> \result == 0) &&
            (val > 0 ==> \result == val) &&
            (val < 0 ==> \result == -val);
 */
int abs(int val) {
  if(val == 0) {
    return 0;
  } else {
    if(val < 0) {
      return val;
    } else {
      return -val;
    }
  }
}
```

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Verification

Frama - C
platform

Verification in
practice

Concluding
remarks

Verification Examples: Issue #3

The verification is inconclusive, the prover has failed!

Figure 14: What is the actual reason?

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Verification

Frama - C
platform

Verification in
practice

Concluding
remarks

Verification Examples: Split

What if we split the post-condition from:

```
/*@ requires INT_MIN < val;
    ensures (val == 0 ==> \result == 0) &&
            (val > 0 ==> \result == val) &&
            (val < 0 ==> \result == -val);
 */
```

into separate statements:

```
/*@ requires INT_MIN < val;
    ensures (val == 0 ==> \result == 0);
    ensures (val > 0 ==> \result == val);
    ensures (val < 0 ==> \result == -val);
 */
```

and then run verification again.

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Verification

Frama - C
platform

Verification in
practice

Concluding
remarks

Verification Examples: Insights

This gives us insights into what could be wrong:

Figure 15: Finding the root-causes

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Verification

Frama - C
platform

Verification in
practice

Concluding
remarks

Verification Examples: Bugs

Now we can now look into the code and identify bugs:

```
/*@ requires INT_MIN < val;
    ensures (val == 0 ==> \result == 0);
    ensures (val > 0 ==> \result == val);
    ensures (val < 0 ==> \result == -val);
 */
int abs(int val) {
  if(val == 0) {
    return 0;      //OK
  } else {
    if(val < 0) {
      return val;  //BUG #1, should return -val
    } else {
      return -val; //BUG #2, should return val
    }
  }
}
```

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Verification

Frama - C
platform

Verification in
practice

Concluding
remarks

Verification Examples: Issue #4

Fixing BUG #1 turns the corresponding post-conditions green!

Figure 16: Sequential issue resolution

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Verification

Frama - C
platform

Verification in
practice

Concluding
remarks

Verification Examples: Issue #5

Fixing BUG #2 yields an **OK** verification result!

Figure 17: Now we are all fine

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Verification

Frama - C
platform

Verification in
practice

Concluding
remarks

Verification Outcomes

If the verification result is **OK**:

- The program satisfies the specification, BUT
- Is the specification correct/complete?

If the verification result is **NOK**[5]:

- An *incorrect implementation*:
  - Find counter-example via test generation;
- A *wrong specification*:
  - Complete spec. and proof analysis;
  - Change/extend the specification;
- A *prover's failure*:
  - Alternative provers;
  - Interactive proof assistants;

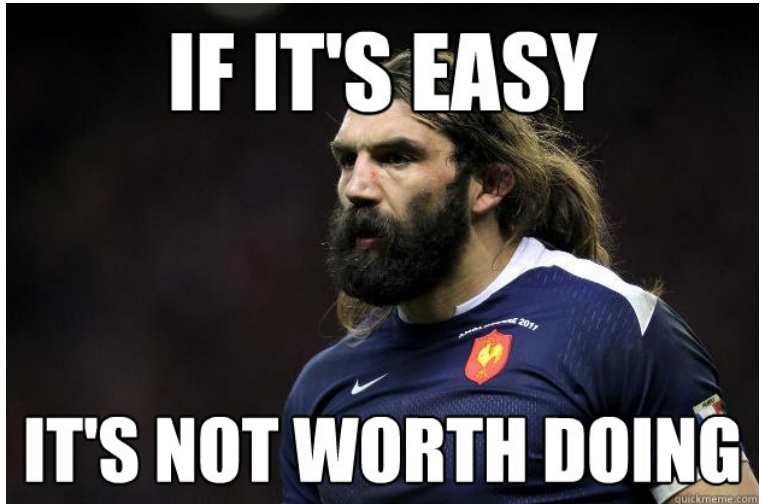[5]This includes a failed verification attempt, e.g. a time out

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Verification

Frama - C
platform

Verification in
practice

Concluding
remarks

Experience summary

Figure 18: It is not so easy but it helps!

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Concluding remarks

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Verification

Frama - C
platform

Verification in
practice

Concluding
remarks

Concluding remarks

We have looked into:

- Software quality and software engineering
- Programming language classification
- Formalization of software verification
- Hoare/Dijkstra approach to formal proving
- Frama-C a platform for ANSI-C code analysis
- Experienced practical program verification

We can conclude that:

- Formal software verification is useful
- It is not yet fully automated
- There is a lot more to learn about it!
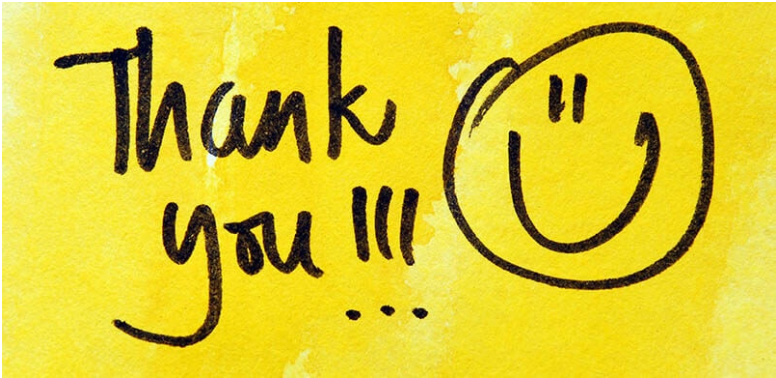
Thank you and are there any questions?

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

I appreciate your time!

Figure 19: It was great to give you a talk!

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Verification

Frama - C
platform

Verification in
practice

Concluding
remarks

More useful links:

- **ACSL Mini-Tutorial:**
  https://frama-c.com/download/acsl-tutorial.pdf
- **ACSL-tutorial:**
  https://frama-c.com/download/acsl-tutorial.pdf
- **ACSL-by-Example:**
  https://www.cs.umd.edu/class/spring2016/cmsc838G/
  frama-c/ACSL-by-Example-12.1.0.pdf
- **Frama-C website:** https://frama-c.com/
- **Frama-C v20.0 manual:** https://frama-
  c.com/download/user-manual-20.0-Calcium.pdf
- **Frama-C WP tutorial:** https://allan-
  blanchard.fr/publis/frama-c-wp-tutorial-en.pdf