

Software quality and formal methods: Hoare/Dijkstra approach

Dr. Ivan S. Zapreev

Neat Software Designs

2020-01-20

Outline:

- Software Quality
 - Motivating Examples
 - Software Development
 - Software Verification
- Programming Languages
 - Language generations
 - Declarative vs. Imperative
 - ANSI-C
- Formal Methods
 - Formal Verification
 - Hoare Approach
 - Edsger Dijkstra
- Frama - C
 - Platform description
 - Plugins overview
 - Introduction to ACSL
- Verification Examples
- Concluding remarks

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Methods

Frama - C

Verification
Examples

Concluding
remarks

Software Quality

Motivating Examples: Major

- 1985–1987 – *Therac-25*:
 - Radiation therapy overdose
 - Control software flaw:
 - Race conditions
 - Death of 6 (six) cancer patients
- 1996 – *Ariane-5 missile*:
 - Missile crash
 - Control software flaw:
 - 64-bit float to 16-bit int
 - \$7 billion development program
 - \$500 million cargo
- 2005 – *Toyota Camry*:
 - Sudden unintended acceleration:
 - Control software flaw:
 - Recursion causing stack overflow
 - 89 deaths and 57 injuries
 - \$1.2 billion compensations

Motivating Examples: More

The 12 Software Bugs That Caused Epic Failures: [<link>](#)

BUGS



BUGS EVERYWHERE

Software Development: V-model

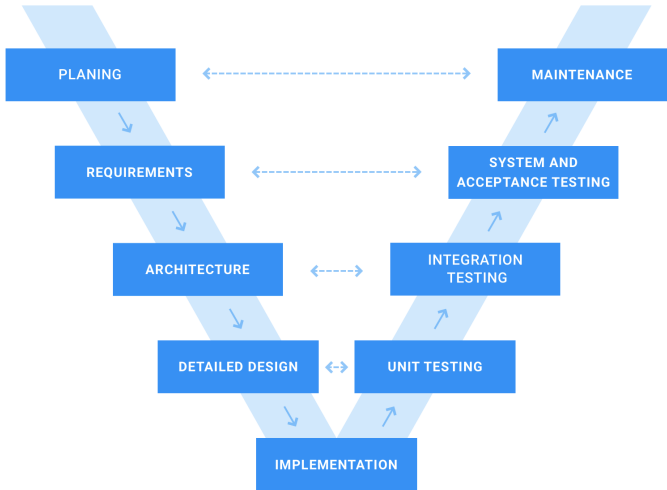


Figure 1: Software development process

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Methods

Frama - C

Verification
Examples

Concluding
remarks

Software Development: V & V

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Methods

Frama - C

Verification
Examples

Concluding
remarks

Is formally defined in, e.g.: 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 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?

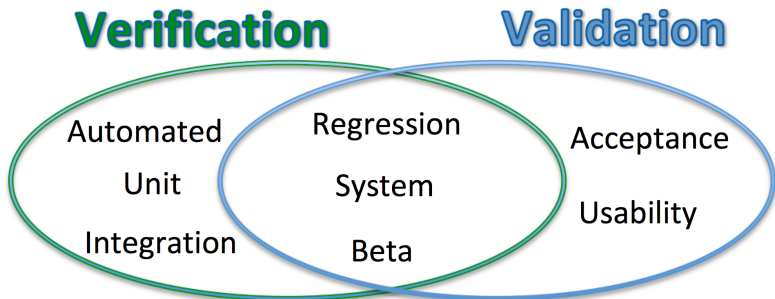


Figure 2: Devision of testing types

Formal Verification

Facts:

- No globally recognized definition of Formal Methods¹.
- Local attempts to have one², e.g.:

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.

Conclusion:

Formal methods are techniques suitable for Verification.

¹“Formal Methods for Industrial Critical Systems”, S. Gnesi, T. Margaria

²“Formal Methods”, Michael Collins, CMU

Software Verification

Goal:

A program shall satisfy a formal specification of its behavior.

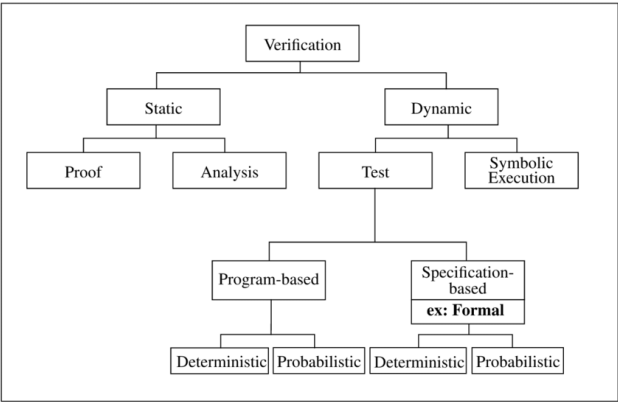


Figure 3: Verification methods

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Methods

Frama - C

Verification
Examples

Concluding
remarks

Programming Languages

Language generations

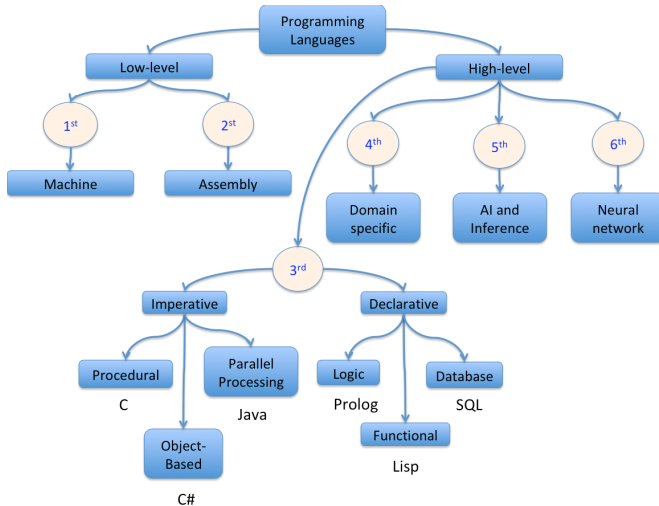


Figure 4: Generations of Programming languages

Declarative vs. Imperative: Main

- *Declarative* – Expresses what to accomplish without specifying concrete steps.

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

- *Imperative* – Describes computation in terms of statements that change a program state.

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

Declarative vs. Imperative: Test

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Methods

Frama - C

Verification
Examples

Concluding
remarks

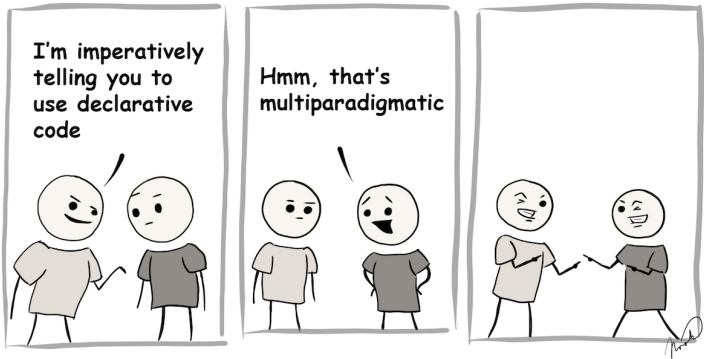


Figure 5: If you laugh, it means you've passed

ANSI-C: Just an old C

Procedural language:

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

C language:

C is an *imperative procedural* language.

ANSI-C:

ANSI-C is a common name for two equivalent language specs:

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

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Methods

Frama - C

Verification
Examples

Concluding
remarks

Formal Methods

Formal verification

Question: Does formal *validation* exist?

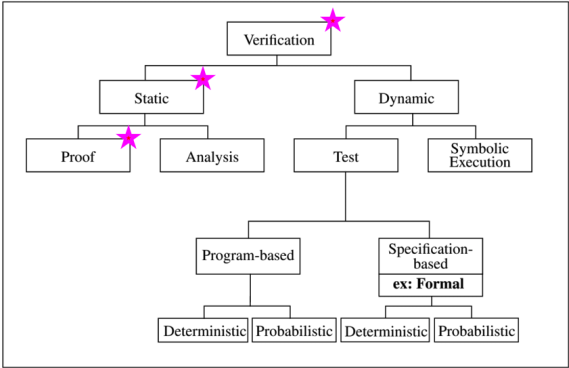


Figure 6: Verification methods

Prove conformance to specifications for *imperative programs*.

Hoare Approach³

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

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

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

$$\overline{\{P\} \text{skip} \{P\}} \text{ and } \overline{\{P[E/V]\} V := E \{P\}}$$

Where E is any expression and V is any variable.

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

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

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

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

Edsger Dijkstra⁴

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* \hat{P}
- If $P \implies \hat{P}$, then the proof is complete

Forward reasoning:

- Based on P and C calculate the *strongest post-condition* \hat{Q}
- If $\hat{Q} \implies Q$, 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

Software
Quality

Programming
Languages

Formal
Methods

Frama - C

Verification
Examples

Concluding
remarks

Frama - C

Platform description

A plug-in-based open-source cross-platform framework for C source-code analysis:

- Browsing 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.

Plugins overview: Main

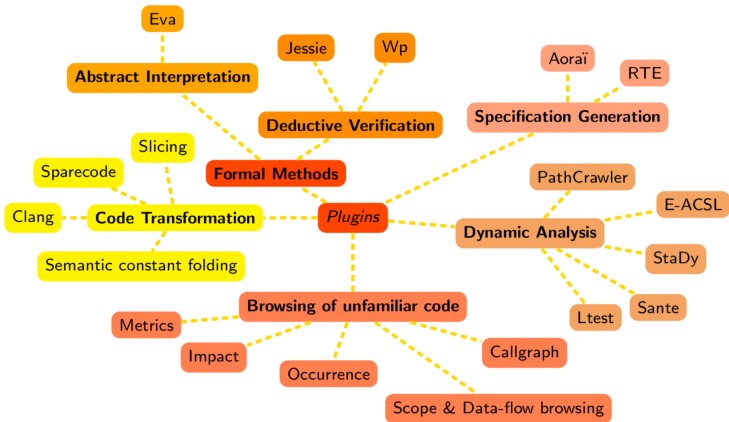


Figure 7: Frama-C plugins

Plugins overview: WP

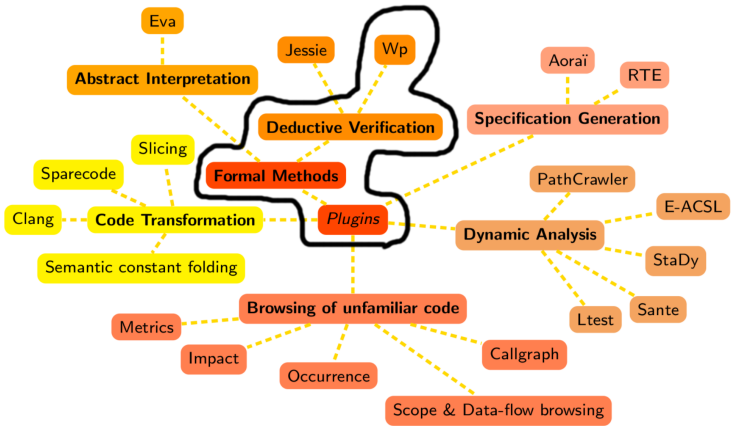


Figure 8: Frama-C WP plugin

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Methods

Frama - C

Verification
Examples

Concluding
remarks

Introduction to ACSL

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Methods

Frama - C

Verification
Examples

Concluding
remarks

Verification Examples

Software
quality and
formal
methods:
Hoare/Dijkstra
approach

Dr. Ivan S.
Zapreev

Software
Quality

Programming
Languages

Formal
Methods

Frama - C

Verification
Examples

Concluding
remarks

Concluding remarks