The Logic Road to Software Homologation

II UB International PhD in Law Conference

Joost J. Joosten et al.

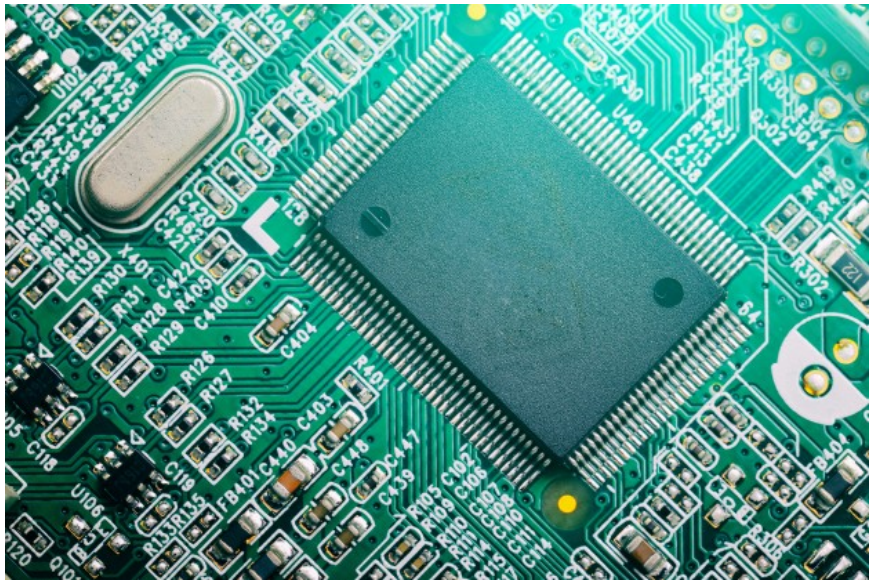
Wednesday, April 24, 2019

The Formal Verification Team:
The team is constituted in the University of Barcelona, and works conjointly with Formal Vindications S.L.
Júlia Pla Bauçà, Guillem Muñoz Mayoral, Gina Garcia Tarrach, Eric Sancho Adamson and Aleix Solé Sánchez have contributed to the making of these slides.

Project funded by the "Ministry of Science, Innovation and Universities", the "State Agency for Research" and the "European Regional Development Fund" (ERDF)

We become more and more dependent on automatised/digitalised processes. For example:

1. Electronic devices used in hospitals,
2. bureaucracy: institutions and companies,
3. the Paris metro system,
4. finance and the stock market,
5. access to press items about world events,
6. natural catastrophe prediction,
7. scientific research: gamma-ray detectors, electronic microscopes,



- hardware bugs are rare but do occur
- 2012 AMD CPU error : about a billion dollor loss;
- 2018: intel and AMD: Meltdown and spectre: various law suits going on

- On June 4, 1996, the maiden flight of the European Ariane 5 launcher crashed about 40 seconds after takeoff

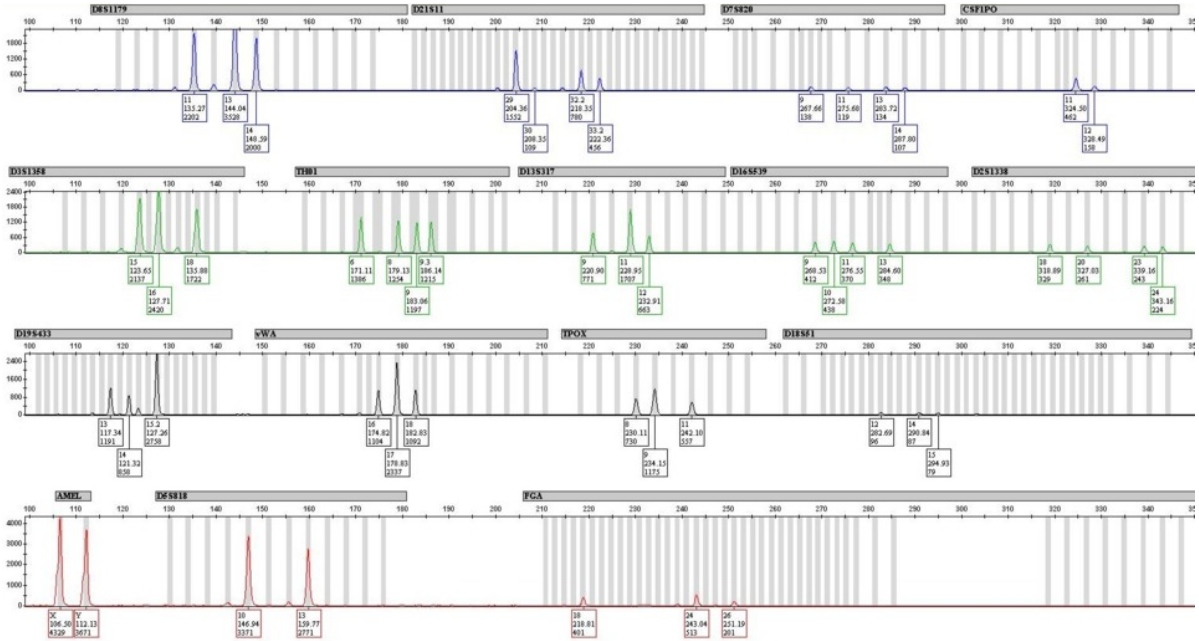
- On June 4, 1996, the maiden flight of the European Ariane 5 launcher crashed about 40 seconds after takeoff
- Media reports indicated that the amount lost was half a billion dollars – uninsured

- On June 4, 1996, the maiden flight of the European Ariane 5 launcher crashed about 40 seconds after takeoff
- Media reports indicated that the amount lost was half a billion dollars – uninsured
- A costly software error!

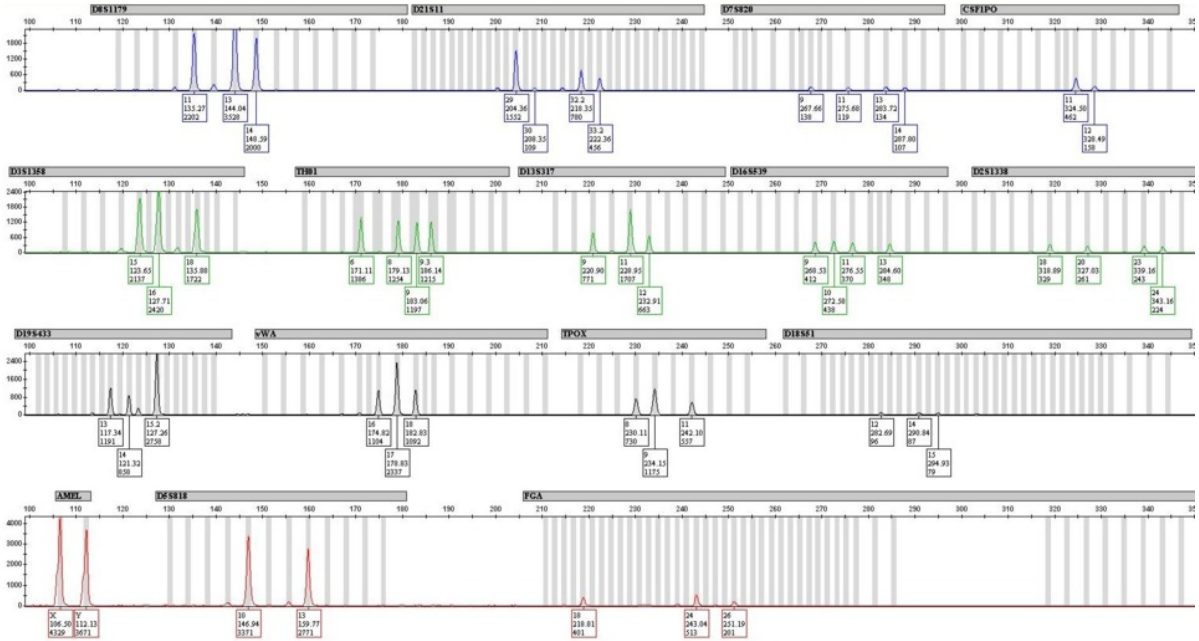
1994: A Chinook Mk2 helicopter which went into service despite a series of failures involving its safety-critical engine control system, whose failure was mortal for 29 people, suspected to be caused by the known software failure.

How was this software developed?

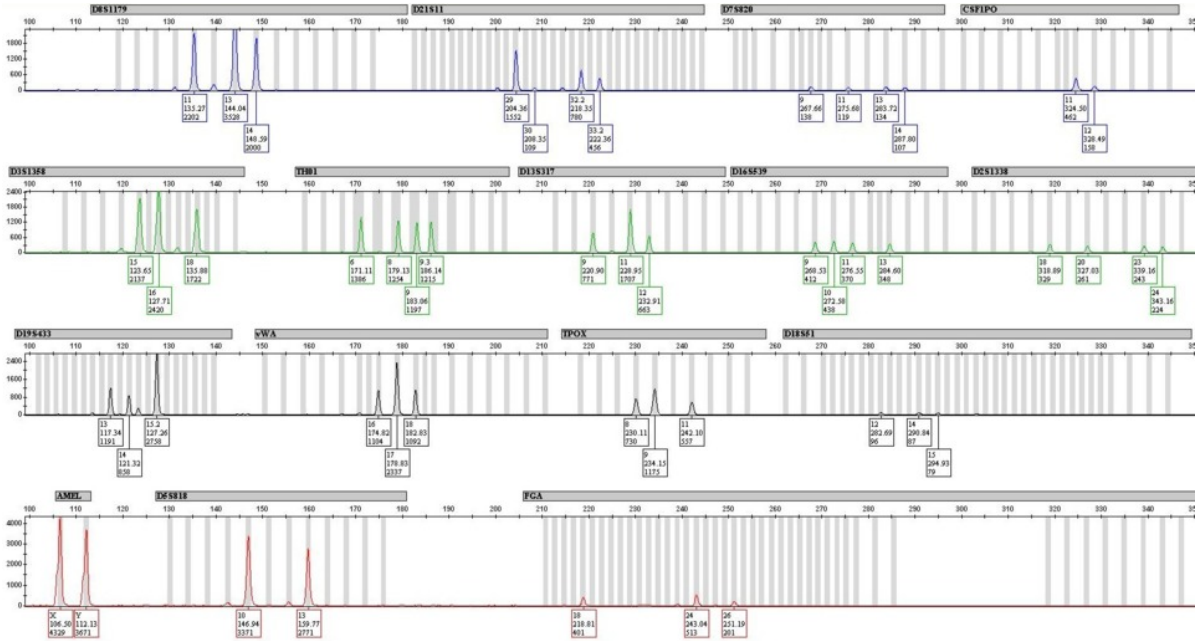
- Legal examples concerning unreliable software:
  - In three cases, judges from USA have petitioned to **make the software open to the public** from proprietary DNA comparing software, due to some unreliable results:



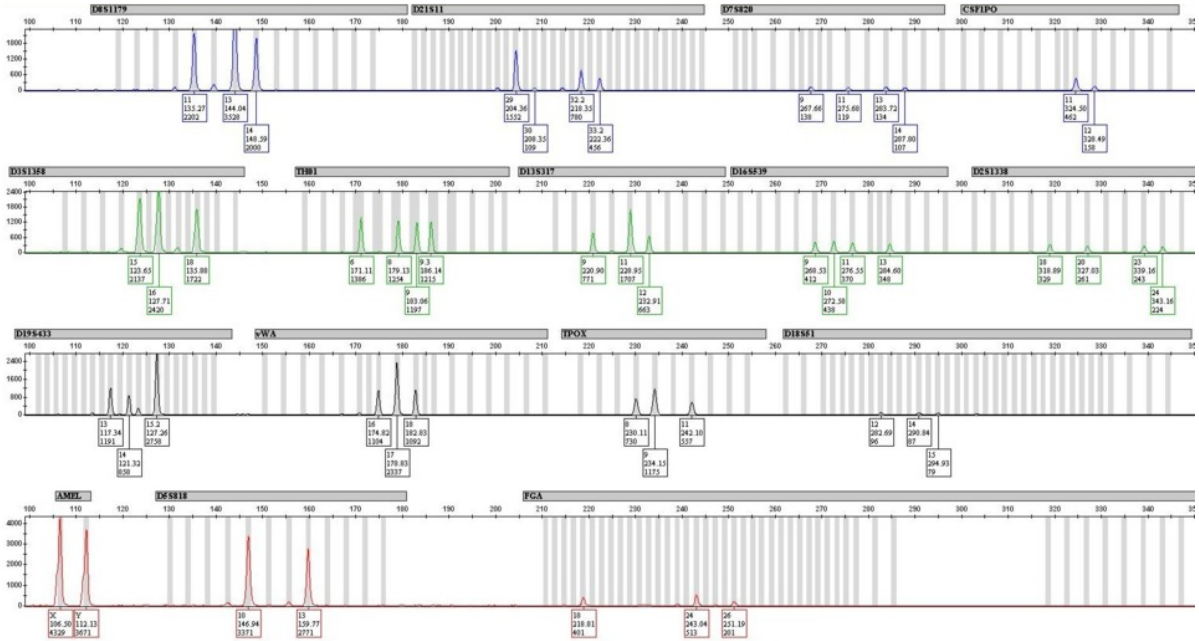
- Legal examples concerning unreliable software:
  - In three cases, judges from USA have petitioned to **make the software open to the public** from proprietary DNA comparing software, due to some unreliable results:
    - STRmix



- Legal examples concerning unreliable software:
  - In three cases, judges from USA have petitioned to **make the software open to the public** from proprietary DNA comparing software, due to some unreliable results:
    - STRmix
    - FST



- Legal examples concerning unreliable software:
  - In three cases, judges from USA have petitioned to **make the software open to the public** from proprietary DNA comparing software, due to some unreliable results:
    - STRmix
    - FST
    - TrueAllele (still proprietary)



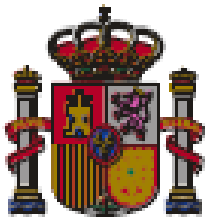
- *New York Times*: Software Designer Reports Error in Anthony Trial. Lizette Alvarez. July 18, 2011.

- *New York Times*: Software Designer Reports Error in Anthony Trial. Lizette Alvarez. July 18, 2011.
  - In this case, in Miami, the fact that a mother had searched the word "chloroform" online **84 times** was used conclusively as evidence in an infanticide case.

- *New York Times*: Software Designer Reports Error in Anthony Trial. Lizette Alvarez. July 18, 2011.
  - In this case, in Miami, the fact that a mother had searched the word "chloroform" online **84 times** was used conclusively as evidence in an infanticide case.
  - The software had counted 84 times **instead of 1**, due to an error. **The programmer apologized**.

- In Valladolid, the court sentenced that a fine was not to be issued in virtue of the software involved **is not homologated**.

- In Valladolid, the court sentenced that a fine was not to be issued in virtue of the software involved **is not homologated**.
- N. Sentence: 30/2019, CONTENCIOSO/ADMTVO court. N. 4 of Valladolid (Spain)



infracción imputada y sancionada en cuanto que no se han incumplido los tiempos de descanso semanales.

En segundo lugar considera que los hechos denunciados no están suficientemente probados a efectos de poderlos considerar constitutivos de la infracción sancionada. En este apartado señala que el tacógrafo del que se han obtenido datos tiene una programación o configuración de fábrica que adolece de errores y que hace que sus resultados no sean fiables ni ciertos. No se trata de una avería o de un mal funcionamiento sino de errores de fabricación, configuración y/o programación llamando la atención sobre la falta de homologación del tacógrafo y, especialmente, del software utilizado dentro del mismo. A lo anterior añade que no consta, y por lo tanto falta, la homologación del software utilizado por las autoridades para obtener y procesar los datos registrados en el tacógrafo.

Se acepta lo alegado por la parte demandante en lo que se refiere a la ausencia de prueba de cargo suficiente respecto al software utilizado por la autoridad correspondiente para obtener los datos registrados en el tacógrafo por lo que, sin necesidad de analizar el resto de la fundamentación jurídica

- To become a practicing medical doctor in Spain you have 6 years of study, an examination (MIR), 4 or 5 years of residence, and more depending on the specialty.

- To become a practicing medical doctor in Spain you have 6 years of study, an examination (MIR), 4 or 5 years of residence, and more depending on the specialty.
- To become a practicing lawyer in Spain you have 4 years of study, 2 years of specialisation (Master's degree), an exam to affiliate with the Bar, 18 month of induction period, and another exam to confirm one has received the appropriate knowledge, including the full memorization of the Spanish constitution.

- Required preparation to become a programmer: There exist 4 year bachelor's degrees, 2 year professional training courses, online courses, or even one can be self-taught.

  Anyone can become a programmer!

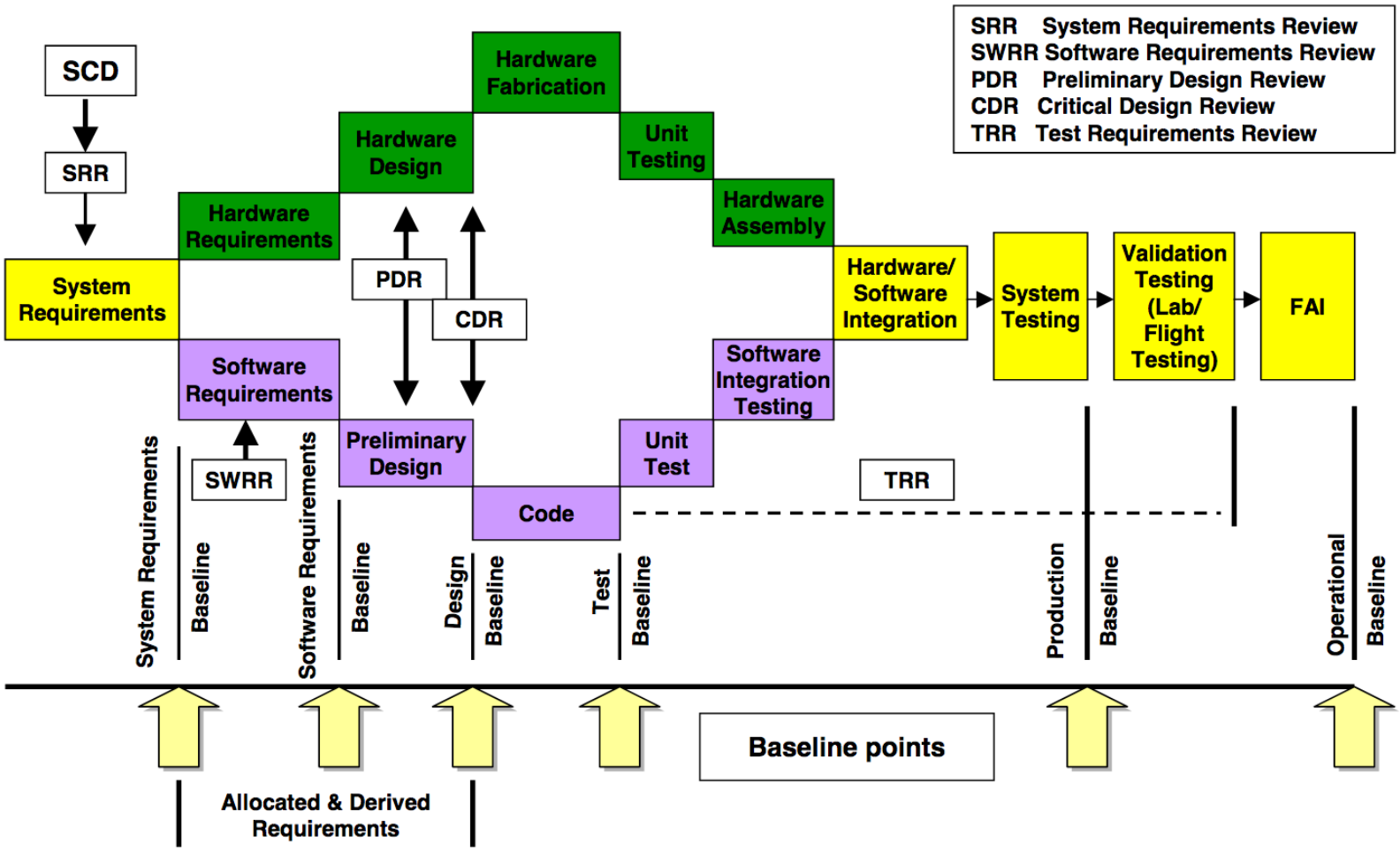
Do we need standards?

The highest standard for verification used in developing aviation software in the European Union is specified within various protocols DO-178B (ED-12B), DO-178C and DO-333. It roughly consists of:

1. Development plan,
2. (non-formal) verification plan,
3. plan for quality testing,
4. plan for configuration.

Development Process Model

(1 of 4)



Key ingredients:

- be careful;
- have lots of sanity checks;
- in many stages of the process;
- do lots of testing.

Can we do any better?

Is there no way to ***know*** that a program is correct, rather than having much circumstantial evidence?

industry	fault density Error for Kloc
Automotive	3
Aviation	1
Shuttle	0.1
Traditional	200
Agile	22

http://leanagilepartners.com/publications.html

Can we do any better?

Is there no way to ***know*** that a program is correct, rather than having much circumstantial evidence?

- Ἐν ἀρχῇ ἦν ὁ λόγος, καὶ ὁ λόγος ἦν πρὸς τὸν θεόν, καὶ θεὸς ἦν ὁ λόγος.

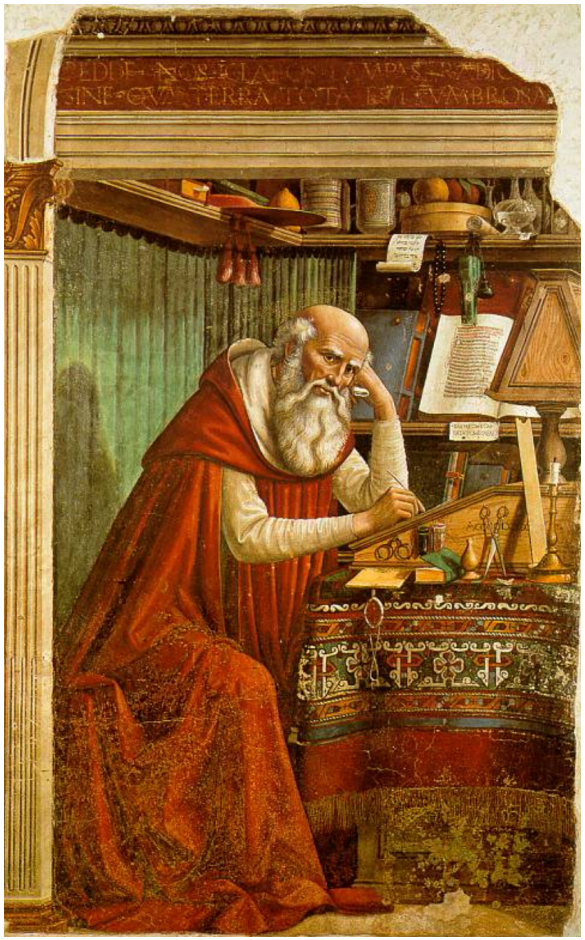
- Ἐν ἀρχῇ ἦν ὁ λόγος, καὶ ὁ λόγος ἦν πρὸς τὸν θεόν, καὶ θεὸς ἦν ὁ λόγος.
- In the beginning was the Word, and the Word was with God, and the Word was God.

- Ἐν ἀρχῇ ἦν ὁ λόγος, καὶ ὁ λόγος ἦν πρὸς τὸν θεόν, καὶ θεὸς ἦν ὁ λόγος.
- In the beginning was the Word, and the Word was with God, and the Word was God.
- From Gospel of John (1:1)

- "Logos" had many meanings:
- number
- esteem
- spoken word
- word as container of thought
- content of thought
- Spirit
- reason

Saint Jerome chose to use "Word" for "Logos" in his Vulgata Bible translation from the fourth century.

Stoic predominant reading: "logos spermatikos": the generative principle of the Universe which creates and takes back all things.



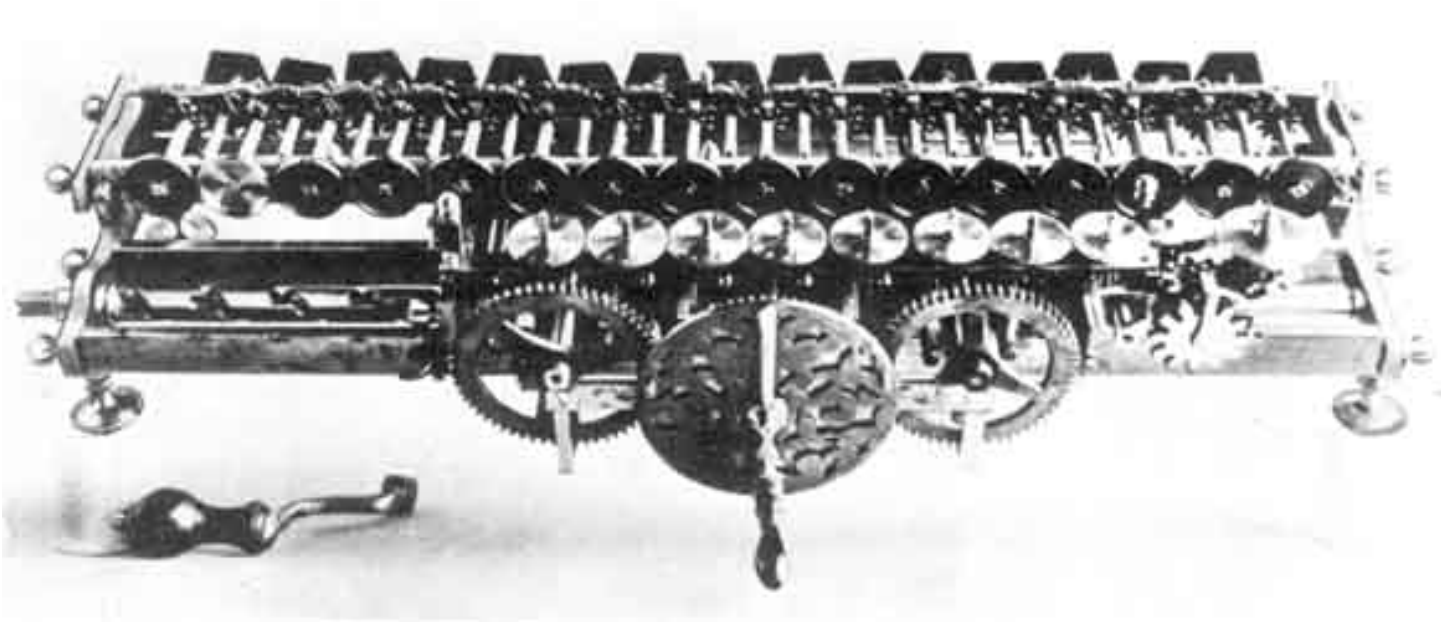
- "Reason" is nowadays the most common connotation of "logic"

- "Reason" is nowadays the most common connotation of "logic"
- Logic: the art of valid reasoning

Gottfried Leibniz (1646-1716)

"The only way to rectify our reasonings is to make them as tangible as those of the Mathematicians, so that we can find our error at a glance, and when there are disputes among persons, we can simply say: Let us calculate [calculemus], without further ado, to see who is right."

Leibniz' Step Reckoner (1671) could add, multiply, calculate square roots, etc.



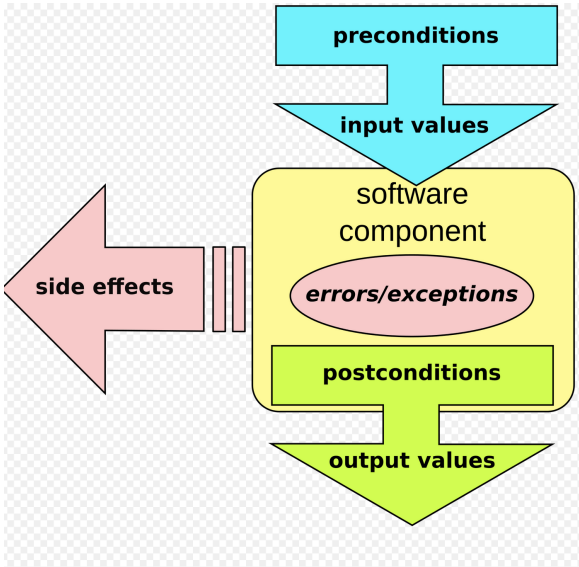
- Leibniz believed in a *universal language* (Characteristica Universalis) in which all disputes could be written

- Leibniz believed in a *universal language* (Characteristica Universalis) in which all disputes could be written
- No ambiguity

- Leibniz believed in a *universal language* (Characteristica Universalis) in which all disputes could be written
- No ambiguity
- Sufficient level of specification

- Leibniz believed in a *universal language* (Characteristica Universalis) in which all disputes could be written
- No ambiguity
- Sufficient level of specification
- The language of logic (propositional, first or higher order, typed or untyped) is closest to a universal language as we know of today

- Leibniz believed in a *universal language* (Characteristica Universalis) in which all disputes could be written
- No ambiguity
- Sufficient level of specification
- The language of logic (propositional, first or higher order, typed or untyped) is closest to a universal language as we know of today
- But still, there is a huge gap between the preciseness in mathematics and in real life



- Edsger Dijkstra: Weakest Precondition
- Hoare Logic
- Design by Contract

L.E.J. Brouwer (1881-1966).

- Another paradigm: reverse programming via theorem proving

L.E.J. Brouwer (1881-1966).

- Another paradigm: reverse programming via theorem proving
- using constructive logic

L.E.J. Brouwer (1881-1966).

- Another paradigm: reverse programming via theorem proving
- using constructive logic
- constructive logic does not validate the *excluded middle* $p \vee \neg p$

- Another paradigm: reverse programming via theorem proving

- Another paradigm: reverse programming via theorem proving
- Main ingredient: Curry-Howard Isomorphism

- Another paradigm: reverse programming via theorem proving
- Main ingredient: Curry-Howard Isomorphism
- $$\frac{\text{programs}}{\text{program specification}} = \frac{\text{proofs}}{\text{theorems}}$$

- Another paradigm: reverse programming via theorem proving
- Main ingredient: Curry-Howard Isomorphism
- $$\frac{\text{programs}}{\text{program specification}} = \frac{\text{proofs}}{\text{theorems}}$$
- Thus we *state a theorem* which can be seen as the specification of a program

- Another paradigm: reverse programming via theorem proving
- Main ingredient: Curry-Howard Isomorphism
- $$\frac{\text{programs}}{\text{program specification}} = \frac{\text{proofs}}{\text{theorems}}$$
- Thus we *state a theorem* which can be seen as the specification of a program
- Next, we *prove the theorem* using constructive logic

- Another paradigm: reverse programming via theorem proving
- Main ingredient: Curry-Howard Isomorphism
- $$\frac{\text{programs}}{\text{program specification}} = \frac{\text{proofs}}{\text{theorems}}$$
- Thus we *state a theorem* which can be seen as the specification of a program
- Next, we *prove the theorem* using constructive logic
- Since we used constructive logic, the proof encapsulates a *program*

- Another paradigm: reverse programming via theorem proving
- Main ingredient: Curry-Howard Isomorphism
- $$\frac{\text{programs}}{\text{program specification}} = \frac{\text{proofs}}{\text{theorems}}$$
- Thus we *state a theorem* which can be seen as the specification of a program
- Next, we *prove the theorem* using constructive logic
- Since we used constructive logic, the proof encapsulates a *program*
- The program can be automatically *extracted*

- Another paradigm: reverse programming via theorem proving
- Main ingredient: Curry-Howard Isomorphism
- $$\frac{\text{programs}}{\text{program specification}} = \frac{\text{proofs}}{\text{theorems}}$$
- Thus we *state a theorem* which can be seen as the specification of a program
- Next, we *prove the theorem* using constructive logic
- Since we used constructive logic, the proof encapsulates a *program*
- The program can be automatically *extracted*
- All the proof steps are checked by a simple and easy computer program: a *proof assistant*

- Another paradigm: reverse programming via theorem proving
- Main ingredient: Curry-Howard Isomorphism
- $$\frac{\text{programs}}{\text{program specification}} = \frac{\text{proofs}}{\text{theorems}}$$
- Thus we *state a theorem* which can be seen as the specification of a program
- Next, we *prove the theorem* using constructive logic
- Since we used constructive logic, the proof encapsulates a *program*
- The program can be automatically *extracted*
- All the proof steps are checked by a simple and easy computer program: a *proof assistant*
- If we trust the proof assistant we can say that the extracted software has **ZERO ERRORS**

- Suppose we wish to write a sorting algorithm using our new paradigm

- Suppose we wish to write a sorting algorithm using our new paradigm
- a program that sorts lists of natural numbers

- Suppose we wish to write a sorting algorithm using our new paradigm
- a program that sorts lists of natural numbers
- so, for example on input $[3, 2, 3, 2, 0]$ it will output $[0, 2, 2, 3, 3]$

- Suppose we wish to write a sorting algorithm using our new paradigm
- a program that sorts lists of natural numbers
- so, for example on input $[3, 2, 3, 2, 0]$ it will output $[0, 2, 2, 3, 3]$
- we first say what it mathematically means to be a sorting function

- Suppose we wish to write a sorting algorithm using our new paradigm
- a program that sorts lists of natural numbers
- so, for example on input $[3, 2, 3, 2, 0]$ it will output $[0, 2, 2, 3, 3]$
- we first say what it mathematically means to be a sorting function
- in a mathematically precise but rich enough language called *Gallina*

- Suppose we wish to write a sorting algorithm using our new paradigm
- a program that sorts lists of natural numbers
- so, for example on input $[3, 2, 3, 2, 0]$ it will output $[0, 2, 2, 3, 3]$
- we first say what it mathematically means to be a sorting function
- in a mathematically precise but rich enough language called *Gallina*
- We thus define a predicate is_sorting_function

- Suppose we wish to write a sorting algorithm using our new paradigm
- a program that sorts lists of natural numbers
- so, for example on input $[3, 2, 3, 2, 0]$ it will output $[0, 2, 2, 3, 3]$
- we first say what it mathematically means to be a sorting function
- in a mathematically precise but rich enough language called *Gallina*
- We thus define a predicate is_sorting_function
-

Specification of sorting function

```
Definition is_sorting_function (f : list nat -> list nat) : Prop :=
  forall (w : list nat),
    permutation w (f w) /\ sorted (f w).
```

Never mind . . .

Definition of sorted

```
Inductive sorted : list nat -> Prop :=
  | sorted_nil : sorted nil
  | sorted_singleton : forall n : nat, sorted [n]
  | sorted_full :
    forall (n m : nat) (w : list nat),
      n <= m ->
      sorted (m :: w) ->
      sorted (n :: m :: w).
```

Definition of permutation

```
Definition permutation (w w' : list nat) :=
  forall n : nat, (number_occurrences n w) = (number_occurrences n w').
```

Definition of number of occurrences

```
Fixpoint number_occurrences (n : nat) (w : list nat) : nat :=
  match w with
    | nil => 0
    | m :: w' =>
      if m == n then
        S (number_occurrences n w')
      else
        number_occurrences n w'
  end.
```

We can now define a particular function insertion_sort

Insertion sort

```
Fixpoint insertion_sort (w : list nat) : list nat :=
  match w with
  | nil => nil
  | n :: tl => insert n (insertion_sort tl)
  end.
```

Insertion

```
Fixpoint insert (n : nat) (w : list nat) : list nat :=
  match w with
  | nil => n :: nil
  | m :: w' =>
    if n <= m then
      n :: m :: w'
    else
      m :: (insert n w')
  end.
```

- We can now state our theorem that the function insertion_sort satisfies the predicate is_sorting_function

- We can now state our theorem that the function insertion_sort satisfies the predicate is_sorting_function
  - ▶

```
Theorem insertion_sort_correct :
  is_sorting_function insertion_sort.
```

- We can now state our theorem that the function insertion_sort satisfies the predicate is_sorting_function

▶

```
Theorem insertion_sort_correct :
  is_sorting_function insertion_sort.
```

- The proof is checked by the computer

- We can now state our theorem that the function insertion_sort satisfies the predicate is_sorting_function

▶

```
Theorem insertion_sort_correct :
  is_sorting_function insertion_sort.
```

- The proof is checked by the computer
- As such, the proof can be conceived as a *legal certificate* that the program meets its specification

- We can now state our theorem that the function insertion_sort satisfies the predicate is_sorting_function

▶

```
Theorem insertion_sort_correct :
  is_sorting_function insertion_sort.
```

- The proof is checked by the computer
- As such, the proof can be conceived as a *legal certificate* that the program meets its specification
- The programming team has full legal coverage

- We can now state our theorem that the function insertion_sort satisfies the predicate is_sorting_function

▶

```
Theorem insertion_sort_correct :
  is_sorting_function insertion_sort.
```

- The proof is checked by the computer
- As such, the proof can be conceived as a *legal certificate* that the program meets its specification
- The programming team has full legal coverage
- The program is as good as its specification is

- Logic-based software verification is getting applied more often nowadays

- Logic-based software verification is getting applied more often nowadays
- In the early days it was only used for high risk projects

- Logic-based software verification is getting applied more often nowadays
- In the early days it was only used for high risk projects
- Metro system of Paris runs without an actual driver!

- Logic-based software verification is getting applied more often nowadays
- In the early days it was only used for high risk projects
- Metro system of Paris runs without an actual driver!
- The list of companies involved is ever growing (Escher Technologies, Intel, AMD, Precision Design Technology)

Who else works with this? A very incomplete list:

- Formally:

Who else works with this? A very incomplete list:

- Formally:
  - **Adacore**: `https://www.adacore.com/`

Who else works with this? A very incomplete list:

- Formally:
  - **Adacore**: `https://www.adacore.com/`
  - **Imandra**: `https://www.imandra.ai/`

Who else works with this? A very incomplete list:

- Formally:
  - **Adacore**: `https://www.adacore.com/`
  - **Imandra**: `https://www.imandra.ai/`
  - **Formal Vindications**: `http://formalvindications.com/`

Who else works with this? A very incomplete list:

- Formally:
  - **Adacore**: `https://www.adacore.com/`
  - **Imandra**: `https://www.imandra.ai/`
  - **Formal Vindications**: `http://formalvindications.com/`
- Non-formally (dynamic testing, validation, etc.):

Who else works with this? A very incomplete list:

- Formally:
  - **Adacore**: `https://www.adacore.com/`
  - **Imandra**: `https://www.imandra.ai/`
  - **Formal Vindications**: `http://formalvindications.com/`
- Non-formally (dynamic testing, validation, etc.):
  - **System Verification**: `https://systemverification.com/`

Who else works with this? A very incomplete list:

- Formally:
  - **Adacore**: `https://www.adacore.com/`
  - **Imandra**: `https://www.imandra.ai/`
  - **Formal Vindications**: `http://formalvindications.com/`

- Non-formally (dynamic testing, validation, etc.):
  - **System Verification**: `https://systemverification.com/`
  - **Boston Atlantic Technology, INC**: `https://www.bostonatlantic.net/`

Who else works with this? A very incomplete list:

- Formally:
  - **Adacore**: `https://www.adacore.com/`
  - **Imandra**: `https://www.imandra.ai/`
  - **Formal Vindications**: `http://formalvindications.com/`

- Non-formally (dynamic testing, validation, etc.):
  - **System Verification**: `https://systemverification.com/`
  - **Boston Atlantic Technology, INC**: `https://www.bostonatlantic.net/`
  - **OAK systems PVT. LTD**: `https://oaksys.net`

Who else works with this? A very incomplete list:

- Formally:
  - **Adacore**: `https://www.adacore.com/`
  - **Imandra**: `https://www.imandra.ai/`
  - **Formal Vindications**: `http://formalvindications.com/`

- Non-formally (dynamic testing, validation, etc.):
  - **System Verification**: `https://systemverification.com/`
  - **Boston Atlantic Technology, INC**: `https://www.bostonatlantic.net/`
  - **OAK systems PVT. LTD**: `https://oaksys.net`
  - **Rapita Systems**: `https://www.rapitasystems.com/`

Who else works with this? A very incomplete list:

- Formally:
  - **Adacore**: `https://www.adacore.com/`
  - **Imandra**: `https://www.imandra.ai/`
  - **Formal Vindications**: `http://formalvindications.com/`
- Non-formally (dynamic testing, validation, etc.):
  - **System Verification**: `https://systemverification.com/`
  - **Boston Atlantic Technology, INC**: `https://www.bostonatlantic.net/`
  - **OAK systems PVT. LTD**: `https://oaksys.net`
  - **Rapita Systems**: `https://www.rapitasystems.com/`
- Interested: many, like Facebook, Google, Bitcoin, Automated Driving, Aviation, Volkswagen, etc.

- Fines are issued to (truck) drivers on the basis of computer readings of tachographs

- Fines are issued to (truck) drivers on the basis of computer readings of tachographs
- some drivers are even sent to prison

- Fines are issued to (truck) drivers on the basis of computer readings of tachographs
- some drivers are even sent to prison
- tons of errors occur

- Fines are issued to (truck) drivers on the basis of computer readings of tachographs
- some drivers are even sent to prison
- tons of errors occur
- A consortium University of Barcelona, Formal Vindications S.L. and Guretruck S.L. develops legal formally verified software for tachographs

- Fines are issued to (truck) drivers on the basis of computer readings of tachographs
- some drivers are even sent to prison
- tons of errors occur
- A consortium University of Barcelona, Formal Vindications S.L. and Guretruck S.L. develops legal formally verified software for tachographs
- The project is funded through the Generalitat, The Spanish Ministry of Science, Innovation and Universities and European Regional Development Funds (ERDF = FEDER) for about one million euro.

'If we desire respect for the law, we must first make the law respectable'.

Louis D. Brandeis, Cleveland Hill

'If we desire algorithmic implementation of a law, we must first make that law computable'.

JjJ

A computable law should have

- clear delimitation of applicability;
- in particular: unambiguous input specification;
- property of being itself unambiguous;
- algorithmic precision: no arbitrary choices left to the engineers;
- certain logical requirements:
  - Consistency;
  - Completeness;
- reflections on physical limits;
- reflections on computational limits;
- mathematically clean properties.

Regulation (EU) 2016/799 lays down the requirements for the construction, testing, installation, operation and repair of tachographs —digital devices which record the activities of road transport drivers.

We analyse Requirements (51) and (52) of the regulation with the goal of identifying problematic behaviour.

Tachographs are required to output one activity per minute.

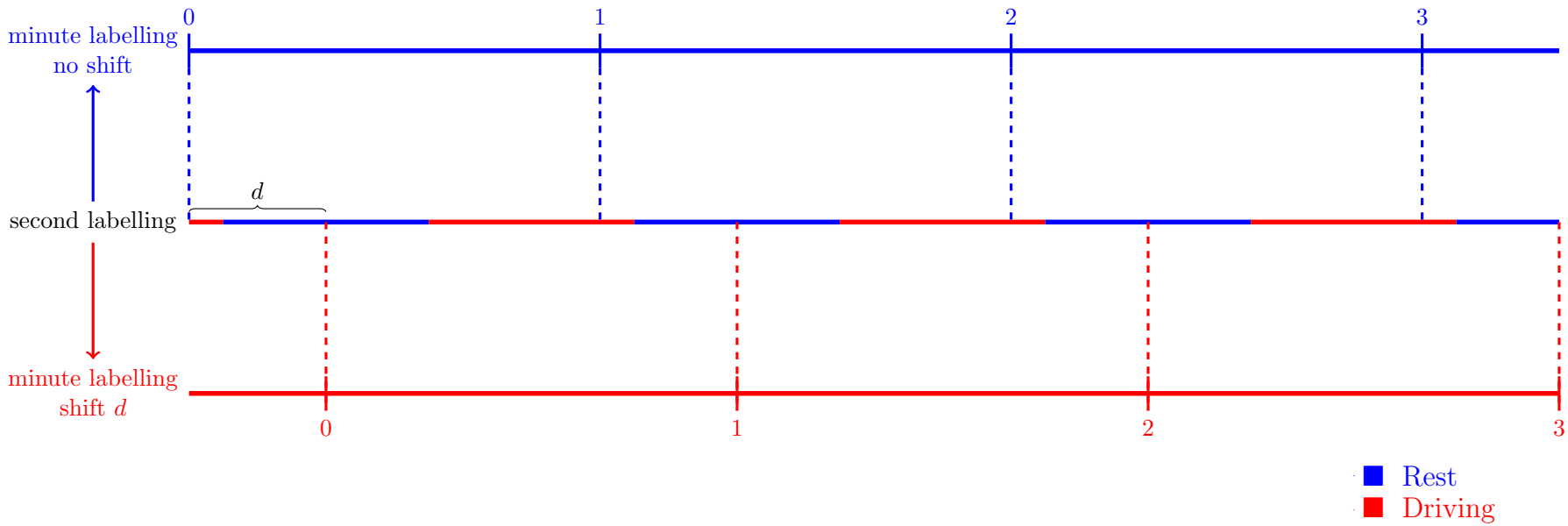
Requirements (51) and (52) are meant to prescribe how to label minutes according to the recorded labelling of seconds.

(51) Given a calendar minute, if DRIVING is registered as the activity of both the immediately preceding and the immediately succeeding minute, the whole minute shall be regarded as DRIVING.

(52) Given a calendar minute that is not regarded as DRIVING according to requirement 051, the whole minute shall be regarded to be of the same type of activity as the longest continuous activity within the minute (or the latest of the equally long activities).

A mathematical analysis reveals:

1. The order should be **first apply (52)** and **next apply (51)**;
2. The law is **not shift-invariant**



So, using one clock the tachograph says you have been driving all the time

using another clock with a couple of seconds of difference the tachograph says you have been resting all the time

- Is it bad that the law is **not shift-invariant**?
- English: "Given a calendar minute";
- The Spanish version says *un minuto cualquiera*, “any minute”;
- the Italian says *un intervallo di un minuto*, “an interval of a minute”;
- And actually, the law prescribes usage of UTC (including leap-seconds)
- But in practice everyone uses UT1 (currently 27 seconds of difference with UTC)
- Guretruck SL and Formal Vindications SL have found differences in real driver files of up to 8% caused by using UT1 instead of UTC

Conclusions

1. Legal/critical software should contain ZERO errors
2. The only way to achieve this is using verified software (strong homolgation)
3. But the laws that are amenable to this should be computable
4. Having good mathematical, legal, and computational properties
5. Law and mathematical logic: a long-lasting marriage?

THANK YOU FOR YOUR ATTENTION!!!