Applying a Formal Method In Industry
A 25-Year Trajectory
All about Safety Critical Systems

Systems where life is at risk

Would you dare to execute your program if someone would die in case of a crash / core dump / fatal error / etc. ?
Systems = Software + hardware + environment

Specification error
Development error
Programming error
Bad compilation

Wrong execution
Processor reliability: $10^{-6}$ to $10^{-7}$ / h

2 processors in parallel (or more)
Protecting mechanisms in case of perturbation
Safety Critical Systems & standards

Standards
- Domain dependent
- Recommendations:
  - No definitive recipe to produce safe systems
  - Software and hardware, development process

IEC 61508: Software design and dev. (table A.2)

<table>
<thead>
<tr>
<th>Technique/Measure</th>
<th>Ref</th>
<th>SIL1</th>
<th>SIL2</th>
<th>SIL3</th>
<th>SIL4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Fault detection and diagnosis</td>
<td>C.3.1</td>
<td>---</td>
<td>R</td>
<td>R</td>
<td>HR</td>
</tr>
<tr>
<td>2 Error detecting and correcting codes</td>
<td>C.3.2</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>3a Failure assertion programming</td>
<td>C.3.3</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>HR</td>
</tr>
<tr>
<td>3b Safety bag techniques</td>
<td>C.3.4</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>3c Diverse programming</td>
<td>C.3.5</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>3d Recovery block</td>
<td>C.3.6</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>3e Backward recovery</td>
<td>C.3.7</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>3f Forward recovery</td>
<td>C.3.8</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>3g Re-try fault recovery mechanisms</td>
<td>C.3.9</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>HR</td>
</tr>
<tr>
<td>3h Memorising executed cases</td>
<td>C.3.10</td>
<td>R</td>
<td>---</td>
<td>R</td>
<td>HR</td>
</tr>
<tr>
<td>4 Graceful degradation</td>
<td>C.3.11</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>5 Artificial intelligence - fault correction</td>
<td>C.3.12</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>6 Dynamic reconfiguration</td>
<td>C.3.13</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>7a Structured methods including for example, ISO, MASCOT, SADT and Yourdon</td>
<td>C.2.1</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
</tr>
<tr>
<td>7b Semi-formal methods</td>
<td>Table B.7</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>HR</td>
</tr>
<tr>
<td>7c Formal methods including for example, CCS, CSP, HOL, LOTOS, OBJ, temporal logic, VDM and Z</td>
<td>C.2.4</td>
<td>---</td>
<td>R</td>
<td>R</td>
<td>HR</td>
</tr>
<tr>
<td>8 Computer-aided specification tools</td>
<td>B.2.4</td>
<td>R</td>
<td>R</td>
<td>HR</td>
<td></td>
</tr>
</tbody>
</table>

a) Appropriate techniques/measures shall be selected according to the safety integrity level. Alternate or equivalent techniques/measures are indicated by a letter following the number. Only one of the alternate or equivalent techniques/measures has to be satisfied.

b) The measures in this table concerning fault tolerance (control of failures) should be considered with the requirements for architecture and control of failures for the hardware of the programmable electronics in part 2 of this standard.
All about Safety Critical Systems

Standards

- Safety case: a demonstration that the feared event will not happen more frequently than expected

Safety Integrity Level 3: 1 failure every 100 years

Safety Integrity Level 4: 1 failure every 10,000 years
Formalizing software
Presenting the B method and Atelier B

Verifying formally parameters
Presenting formal data validation

Executing software on a safety hardware platform
Presenting CLEARSY Safety Platform

CLEARSY Safety Platform
Starter kit SK₀
The B Method in a Nutshell

≡ Is a **formal method** to develop software mathematically **proved** to comply with its specification

≡ [**Formal**] : it relies on a mathematical model of the software, containing both
• what the software is expected to do and
• its algorithm

≡ [**Method**] : Model is decomposed into smaller models in order to manage complexity (“divide and conquer”)

≡ The model is mathematically [**proved**]: the algorithm doesn’t contradict its specification

≡ The software code is generated from the model
• Code is readable, very close to the model and is easily checked

≡ The final software application is made of parts developed with B and parts not developed formally
« Only inactive sequences can be added to the active sequences execution queue. »

```
activation_sequence = /* Activation d'une séquence non active */
PRE ~(sequences = sequences_actives) THEN
  ANY sequ WHERE
    sequ ∈ sequences - sequences_actives
  THEN
    sequences_actives := sequences_actives U {sequ}
  END;
END;

activation_sequence = /* Activation d'une séquence non active */
VAR sequ IN
  sequ <-- indexSequenceInactive;
  activateSequence(sequ)
END;
```

```
void MO_activation_sequence(void)
{
  CTX_SEQUENCES sequ:
    sequence_manager_indexSequenceInactive({sequ});
    sequence_manager_activateSequence({sequ});
}
```

```
0x01F970 FFFF D84C 2440 89C6 BD7D 0C88 4110 89CE
0x01F990 89C6 0CD3 148E 0000 0000 8D42 0889 F807
0x01F990 7617 87C7 0400 0000 740F 8441 0C88 7D10
0x01F9A0 83C6 0495 483C 8D42 04FC 95C1 C6E9 02F3
```
« Only inactive sequences can be added to the active sequences execution queue. »

```
void NC__activation_sequence(void)
{
    CTX__SEQUENCES seq;
    sequence_manager__indexSequenceInactive(seq);
    sequence_manager__activeSequence(seq);
}
activation_sequence = /* Activation d'une séquence non active */
PRE ~(sequences = sequences_actives) THEN
    ANY sequ WHERE
        sequ ∈ sequences - sequences_actives
    THEN
        sequences_actives := sequences_actives U (sequ)
    END;
END;
```

**Natural language requirement**

**B Specification**

**B Implementation**

**C generated code**

**Binary code**

**Proof (coherence)**

**Proof (refinement)**

**Proof (coherence)**

---

**Cyclic software single-thread**
The B method in one example

**MACHINE = specification**

- **Variables**
  - Types for variables (set, function, scalar, boolean, integer, table, etc.)
  - Constraints for variables
  - Invariant is always true
    - Established by initialization
    - If true before operation execution, still true after

- **Invariant**

- **Initialization**
  - First value for variables
  - May be non-deterministic
  - All variables initialized at the same time

- **Operation**
  - Modify variables
  - Specified as a transfer function (no algorithm)
The B method in one example

```
1- MACHINE
2- M0
3- VARIABLES
4- xx
5- INVARIANT
6 2/2 xx: INTEGER
7- INITIALISATION
8 1/1 xx := 0
9- OPERATIONS
10 1/1 inc =
11
12
13 END
```
The B method in one example
**The B method in one example**

**MACHINE = specification**

```
1- MACHINE
2-   M0
3- VARIABLES
4-   xx
5- INVARINT
6- 2/2   xx: INTEGER
7- INITIALISATION
8- 1/1   xx := 0
9- OPERATIONS
10 1/1   inc =
11
12 END
```

**IMPLEMENTATION = algorithm**

```
1- IMPLEMENTATION M0_i
2- REFINES M0
3- CONCRETE_VARIABLES
4- 1/2   xx
5- INVARINT
6- 1/2   xx: INT
7- INITIALISATION
8- 1/1   xx := 0
9- OPERATIONS
10 0/1   inc = |
The B method in one example

```
IMPLEMENTATION M0_i
REFINES M0

CONCRETE_VARIABLES
1/2   xx

INVARIANT
1/2   xx : INT

INITIALISATION
1/1   xx := 0

OPERATIONS
0/1   inc = xx := xx + 1  Endless increment

END
```

If we increment enough, we get out of type INT
The B method in one example

```
IMPLEMENTATION M0_i
REFINES M0

CONCRETE_VARIABLES
3/4
   xx

INVARIANT
3/3
   xx: INT

INITIALISATION
1/1
   xx := 0

OPERATIONS
2/3
   inc = IF xx = MAXINT THEN xx := 0 ELSE xx := xx + 1 END

END
```

- If we get to the upper bound (MAXINT) then we reset else we increment.

- If we get to the upper bound then the value should be 0.
The B method in one example

MACHINE M0
VARIABLES
  xx
INARIANT
  xx: INTEGER
INITIALISATION
  xx := 0
OPERATIONS
  inc = CHOICE xx := 0 OR xx := xx + 1 END
END

IMPLEMENTATION M0_i
REFINES M0
CONCRETE_VARIABLES
  xx
INARIANT
  xx: INT
INITIALISATION
  xx := 0
OPERATIONS
  inc = IF xx = MAXINT THEN xx:=0 ELSE xx := xx +1 END
END

The 2 models are compatible
#ifndef _M0_h
#define _M0_h

#include <stdio.h>
#include <stdbool.h>

#ifdef __cplusplus
extern "C"
#endif /* __cplusplus */

#endif /* _M0_h */

#include "M0.h"

/* Clause CONCRETE_CONSTANTS */
/* Basic constants */

/* Clause SETS */

/* Clause CONCRETE_VARIABLES */

static int32_t M0_xx;

/* Clause INITIALISATION */

void M0___INITIALISATION(void)
{
    M0_xx = 0;
}

/* Clause OPERATIONS */

void M0_inc(void)
{
    extern void M0___INITIALISATION(void);
    if(M0_xx == 2147483647)
    {
        M0_xx = 0;
    }
    else
    {
        M0_xx = M0_xx + 1;
    }
}
Guaranty: no programming error

IMPLEMENTATION M0_i
REFINES M0

CONCRETE_VARIABLES
xx, yy, zz,
tab

INITIALISATION
xx := 0;
yy := 0;
zz := 0;
tab := (0..10)*{0}

OPERATIONS
op =
BEGIN
xx := yy / zz;
yy := MAXINT;
xx := yy + 1;
tab(yy) := 0
END

Programming error
• no overflow,
• no division by zero,
• no access to a table outside its domain

3 integers
1 table of 11 elements: tab(0), ..., tab(10)
Division by 0
Overflow
Table used outside its domain
Guaranty: no programming error

IMPLEMENTATION M0_i
REFINES M0

CONCRETE_VARIABLES
xx, yy, zz,
tab

INITIALISATION
xx := 0;
yy := 0;
zz := 0;
tab := (0..10)\{0\}

OPERATIONS
op =
BEGIN
xx := yy / zz;
yy := MAXINT;
xx := yy + 1;
tab(yy) := 0
END

END

When we save the model, a number of proof obligations are not automatically proved
Guaranty: no programming error

Overflow is detected because not provable
Division by 0 is detected because not provable
Access to table outside its domain is detected because not provable.
Models Architecture

- One operation cannot call other operations from the same implementation
- One operation can call operations defined in other machines
- These machines have to be imported in an implementation
- Variables defined in imported machines have to be different: a variable cannot be modified in 2 components
IMPORTS have to be applied iteratively to obtain the target decomposition

The decomposition graph should be a tree

Some components do not have an implementation in B
Integrating B to a Software Project

≡ Compatible with any SW development cycle
To be used when something has to be proved
Why using B /1

- Improved level of confidence
  Brought by mathematical formalism and proof

- Ambiguities are removed
  The mathematical model captures the meaning of the software

- Early error detection
  Errors are discovered
  • by proof during the modelling and
  • not by test once the software is built
Why using B/2

≡ Most testing useless
Testing is replaced by proof
Mathematical proof is exhaustive, testing is not

≡ Avoid redundant software development
For highest safety integrity level, only one B model required, compared with two software developed more traditionally by two independent teams

≡ Accepted for certification
Several industrial standards recommend or strongly recommend the use of formal methods (EN50128, IEC61508)
Why not using B?

- Numerical algorithms
  - FFT, Kalman, etc.

- Non functional properties (cannot be proved within B)
  - “The software terminates in less than xxx milliseconds”

- Optimality
  - “The software is the smallest / fastest / most efficient one”

- Nothing to prove
  - No properties: no other specification than “my software should do A then B then C”
Atelier B

≡ A tool, freely available and fully functional, for
  • developing [safety critical] software (B language, B method)
  • modelling systems (Event-B language)

≡ Includes editor, proof tools and code generator

≡ Dedicated support for maintenance contract holders

≡ Used for certified applications up to SIL4

≡ Version 4.5 to be published in June 2018

http://www.atelierb.eu/en
Educational resources

http://mooc.imd.ufrn.br

MOOC on B (under construction)
• 20 lectures covering B for software development
• Collaboration with UFRN/IMD (Natal)
Industrial applications

3M€ to strengthen Alstom Atelier B and fully support the B method
Funded by RATP, SNCF, INRETS
Tool qualification by RATP

accepted by standards

2 requirements:
• 10s to prove 1 proof obligation
• Ability to support the development of the first driverless metro

Paris Line 14 driverless metro ready by end 1998
• 110 kloc B model, 83% automatic proof
• 86 kloc Ada code
• Still in version 1.0 after 19 years exploitation
ATELIER B

METROS AND TRAINS EQUIPPED WITH B SIL4 SOFTWARE
Software Developer

- 30%+ of the driverless metro in the world (Alstom, Siemens)

- Automatic Train Protection System

What is proved:
if the train is not allowed to go forward, then the emergency brake should be triggered

- Automation to reduce intervals between trains (from 120s to 90s / 75s)
Passive security not sufficient (power off)
Active security is required (trains have to brake when emergency)

We cannot prove absence of collision. 100% safe systems are miracles
Safety Critical Railway Applications

Top level implementation

- Imports 55 components
- Specify top level one-cycle function:
  - Compute location, manage kinetic energy, control PSD, trigger emergency braking, etc.
  - Localization: graph-based algorithms
  - Energy control: integer arithmetic (braking curve)
  - Emergency braking: Boolean predicates

Metrics

- 233 machines, 50 kloc
- 46 refinements, 6 kloc
- 213 implementations, 45 kloc
- 3 000 definitions
- 23 000 proof obligations (83 % automatic proof)
- 3 000 added user rules (85 % automatic proof)

Unit and integration testing useless
Automatic Refinement

≡ Progressive transformation of a model containing all design decisions into a model able to be translated into an imperative language

≡ Refinement engine based on refinement rules
  • Abstract model is hand-written
  • Concrete model is generated

[2004] Canarsie line (ATP): 38k lines of handwritten B, 115k lines of generated B

[2007] Roissy airport shuttle (ATP+ZC): 40k lines of handwritten B, 225k lines of generated B

[2007] Symbolic Calculus Engine: 200k lines of generated B

[2015] SysML Compiler: 300k lines of generated B

Development cost / 2

Automatic Refinement, L. Burdy, J.M. Meynadier, FM 1999
Return of Experience on Automated Refinement in B, T. Lecomte, SETS 2014
There is overEnergy iff I can find a track section starting at X2M, complying with the dynamic chaining of blocks, on which I can

- either find a restriction belonging to a block such as the energy on that restriction, computed by summing deltas of energy of all restrictions located between X2MRes and this restriction, is greater than the energy associated to this restriction,

- or find 2 restrictions belonging to the EOA block, one being before the track section under consideration, the other after the track section, such as the energy associated to the EOA by using these restrictions is positive. »
The Corresponding Predicate

Formalities do not save us from complexity
Systems

Platform screen doors (PSD)

Paris line 1- PSD model animation (Event-B)
Platform Screen Doors Control

- System to install to prepare driverless operation
  - Avoid people to die (pushed) on tracks
  - No direct communication with the train: train arrival and door opening to be detected with diverse sensors
  - SIL4: one failure every 10,000 years
  - 99.999% reliability: one train max missed per year
  - To be developed from scratch in 6 months
Virtual becomes real

≡ Connecting things with the real station

Et voilà !
Formal Data Validation

≡ Proving parameters (constants)
• What is the use of a formally proven software if some of its (non trivial) parameters are wrong?
• Initially metro line static data used by the automatic pilot (software) to drive safely

≡ Data Validation
• Automatic check of large data sets against properties
• Properties: international standards, national regulations, manufacturer habits, customer requirements
• Initially metro line static data used by the automatic pilot (software) to drive safely
• Model-checking applied to

![Table]

Are they
• Consistent?
• Correct?
• Safe?

Up to 100,000+ raw data chunks

Formal Data Validation in the Railways, T. Lecomte, SSS 2016
Modelling Language

≡ Modelling language based on set theory and first order predicates logic (B mathematical language)

Let the set \( \text{TrackCircuit} = \{t_1, t_2, t_3, t_4, t_5\} \)

Let the function \( \text{Next} : \text{TrackCircuit} \rightarrow \text{TrackCircuit} \)

Example: \( \text{Next}(t_1) = t_2, \text{Next}(t_2) = t_3, \text{Next}(t_3) = t_4, \text{Next}(t_4) = t_5 \)

\( \text{Next} = \{t_1 \mapsto t_2, t_2 \mapsto t_3, t_3 \mapsto t_4, t_4 \mapsto t_5\} \)

One may define \( \text{FirstTrackCircuit} \in \text{TrackCircuit} - \text{ran(Next)} \)

Let the function \( \text{KpAbs} : \text{TrackCircuit} \rightarrow \mathbb{N} \)

\( \forall x \cdot (x \in \text{TrackCircuit} \land x \in \text{dom(Next)} \Rightarrow \text{KpAbs(Next(x))} > \text{KpAbs(x)}) \)
A rule requires a minima:

- A selection predicate
- A verification predicate
- An error message if the verification fails

Let $S_g$ be the signals of the track $T_r$
Each signal $S_i$ of $S_g$ is defined in the table $T_r$
"The signal % used by track % is not defined ", $S_i$, $T_r$

\[
\begin{array}{c|c|c|c}
T_r^0 & T_r^1 & T_r^2 & T_r^3 \\
\hline
S_0 & S_2 & S_3 & S_5 \\
S_1 & & & S_4 \\
\end{array}
\]

\[
S_g = S_0, S_1, S_2, S_3, S_4, S_5
\]

\[
\begin{array}{l}
\text{ANY} \\
S, i \\
\text{WHERE} \\
i \in \text{dom}(Tr) \& \\
S \in \text{Tr}(i) \\
\text{EXPECTED} \\
S \in S_g \\
\text{END}
\end{array}
\]

$Tr$ is a function defined for the values 0 to 3 (its domain).
$Tr(0) = \{S_0, S_1\}$
$Tr(1) = \{S_2\}$
$Tr(2) = \{S_3, S_4\}$
$Tr(3) = \{S_5\}$
Formal Data Validation

- Consistency, correctness, safety
  - Expressed with the mathematical language of B
  - Work well with graph-based properties
  - Provide counter examples when errors found

- Model-checking
  - Performed with ProB
  - Rodin (Siemens) and Alstom have funded development and validation to obtain mature tool
  - Replace months of (boring) engineer work by hours of computer verification
  - Engineer models properties (1 000/line)
Industry-Ready [PUSH-BUTTON !]
- Deal with permanent changes in data and properties
- Redundant tools to obtain diversity
- Rules reused from one project to another
- More than 30 sites verified including:
  - Singapore, Panama, Riyadh, Mecca

Improve the level of confidence of the V&V
For each GradientTopology (GradientTopology.BOT-Zone) totally included in a segment, a Gradient (Gradient.BOT-Zone) is created with the same attributes.
For GradientTopology intersecting different segments, several Gradients (Gradient.BOT-Zone) are created so that each of them is located in only one segment.

When the gradient is constant (GradientTopology.isConstant = Yes):
- the variable gradient information (Gradient.VariableGradient) is not set.
- the constant gradient information is set with the same information of GradientTopology for both parts.
- the elevationDifference.elevationEnd of the part1 and elevationDifference.elevationStart of the part2 (reference to the above figure) are equal to elevationStart + gradient*Length1.
- the information isConstant is set to Yes for both parts.

When the gradient is not constant (GradientTopology.isConstant = No):
- the constant gradient information (ConstantGradient) is not set.
- the elevationDifference.elevationEnd of the part1 and elevationDifference.elevationStart of the part2 (reference to the above figure) are equal to elevationStart + 2*radius*sin(Length1/(2*radius))*sin(gradientStart + Length1/(2*radius)).
- the information radius and transitionCurveType of the variableGradient information are the same for both parts (as initial GradientTopology information).
- the information gradientEnd for part1 and gradientStart of part2 for variableGradient information are set to (gradientEnd-gradientStart)/(Length1 + Length2)*Length1 + gradientStart.
- the information isConstant is set to No for both Part.
Validation of compilation principles of a SIL2 IDE (TGV Atlantique)

- Compilation of hierarchical grafcets into binary file
- Property: sub-grafcets activated in the binary file should correspond to sub-grafcets activated in the models

GRF = {main, G1, G2, G3, G4, ... }
next ∈ GRF ↔ GRF
next = { ..., G7 ↦ G11,...}

ADR = {0x01, 0x13, 0x15, ... }
suiv ∈ ADR ↔ ADR
suiv = { ..., 0x10 ↦ 0x15, ...}

It exists a bijection $bij$ which associates to any GRF node an ADR node such as successors match.

$bij \in \text{GRF} \mapsto \text{ADR} \& \forall xx.(xx \in \text{GRF} \Rightarrow bij[next[xx]] = suiv[bij[xx]])$
Other applications

Matching between 2 graphs, containing 167 nodes, in less than 10 s
CLEARSY Safety Platform

- Need for a technical solution to overcome difficulties to develop SIL3/SIL4 systems
  - Require rare human resources to complete successfully
  - Very short delays (~6 months) to design new systems
  - Off-the-shelf block solutions difficult to adapt

Technology based on double-processor and formal method
Technology experimented several times

Platform screen doors controller installed in Stockholm (Citybanan)

Sao Paulo L15 (2016)

Stockholm Citybanan (2017)

Gateway SATURN (2016)

COPPILOT.M Stockholm application « série A »

implementing the SIL3 safety function

“Automatic Sliding Doors (ASD) Opening Authorization”

Certificate N°: 6393741
Date of issue: 03rd March, 2017
LCHIP in a Nutshell

SOFTWARE & HARDWARE PLATFORM
FOR SAFETY CRITICAL APPLICATIONS

• Factory to generate a function that executes safely

DESIGN

≡ ease the development of safety critical applications

EXECUTION

• Cover the whole development cycle
• Safety principles are built-in
• Based on a formal language and related proof tools
• Mathematical proof replaces unit and integration testing
• Avoid expert transactions over multiple languages

≡ ease the certification of safety critical applications

Disclaimer: The developer is responsible for the safety demonstration
Technical principles

To beam up a function on 2 microcontrollers

- Function → B model → Implementable B model
- Automatic proof
- C translator Compiler → Binary 2
- Compiler → Binary 1

Program
- Sequencer + Safety library

Verification
- μC1
- μC2

Coherency, no programming error

Legend:
- tool
- file
- LCHIP

Development → Execution
Starter Kit SK₀

USB-C connector
Power, upload, monitor

Switches
Simulate digital inputs

Atelier B 4.4 LCHIP Early access
Model, prove, compile, upload
Roadmap

Proof more efficient / automatic
Integration of diverse provers

IDE in dual mode:
- "push-button"
- engineering

Starter Kit $SK_2$
Q3 2019
Roadmap

2 boards connected sharing inputs status
Roadmap

More boards ...
FM successes

Line 1: 350 M passengers going through PSDs in 4 years, no problem. 7 B+ passengers went through the PSDs we control.

Line 14: ATP released in 1998, still in version 1.0, no bug detected so far. Most renovated lines in Paris are going to use B.

LCHIP is being installed on-board all Thales Toronto trains (remote los).

Line 4: system-level study “without the correct by construction” stamp allowed to detect a major leak in the interoperability specification. 7 independent timers related to identified situations may lead to an unwanted collision.
Many opportunities to apply formal methods:
• Emerging domains where safety is not (yet) a subject of interest
• Increasing complexity of our (interconnected) world
• Autonomous vehicles

Many traps as well:
• A software is not safe by itself. It has to be considered in its environment
• Address the problem fully
• FM are not mandatory, most existing systems do not embed them
• Proof has to converge towards full automation
• Need for Leonard(s) da Vinci with FM skills