

AIX
LYON
PARIS
STRASBOURG

WWW.CLEARSY.COM

Méthode Formelle Développement logiciel et Système sûr critique



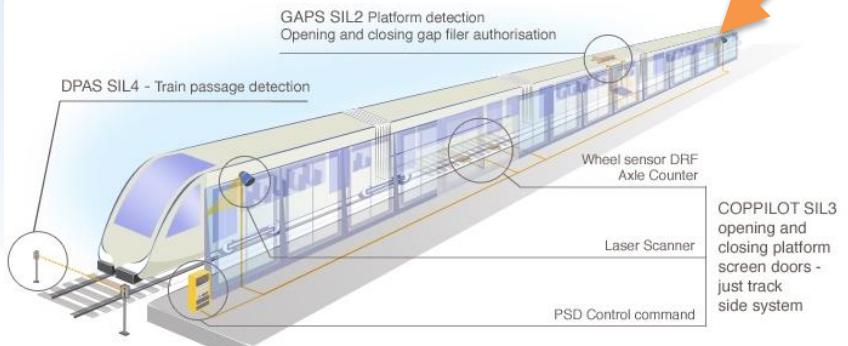
ETIENNE.PRUN@CLEARSY.COM

CLEARSY

Safety Solutions Designer



Pilote automatique de metro



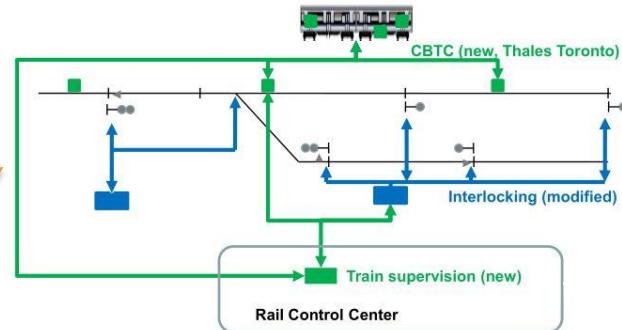
SIL2 - SIL4 système critique sûr

PME Crée en 2001
150+ ingénier
15 M€ CA 2020
Aix, Lyon, Paris,
Strasbourg



ATELIER 

Editeur



Ingénierie formelle système,
RAMS, services



SIL4 produits

<http://www.clearsy.com/>

Sûreté ?

24 juillet 2013 à Saint-Jacques-de-
Compostelle

79 morts

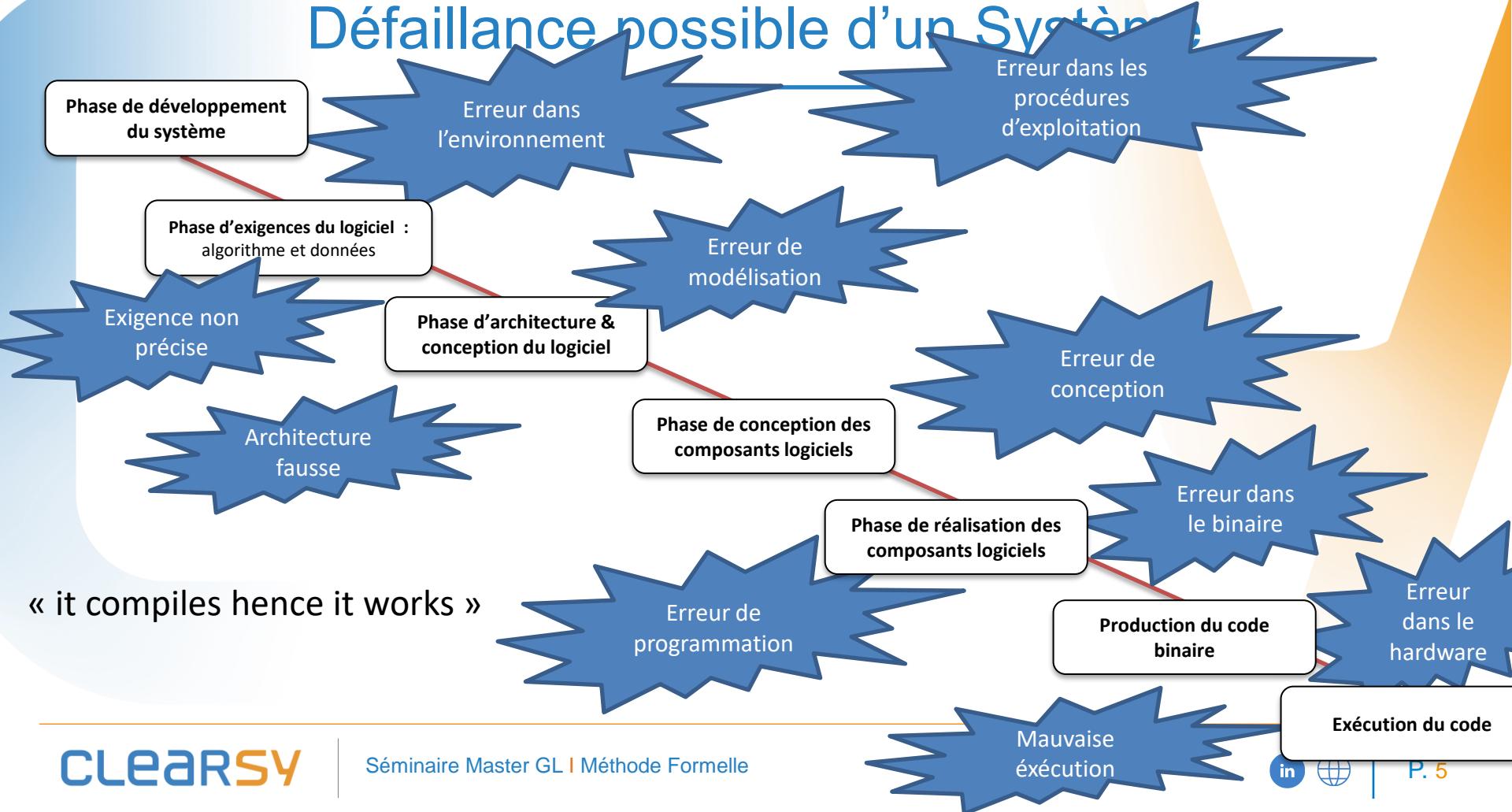
Système Critique Sûr

► Système ou la vie est en danger

- ▷ Trains
- ▷ Avions
- ▷ Voiture
- ▷ Nuclear
- ▷ Machines industrielles
- ▷ Etc.



Défaillance possible d'un Système



Pourquoi les méthodes formelles ?

- ▶ Continuer à maîtriser les systèmes que l'on conçoit et construit
 - ▷ 100 M embarquées sur une voiture (2021) : Estimé à 750 M en 2025
 - ▷ 80 à 100 calculateur



Pourquoi les méthodes formelles ?

- ▶ La commande de vol de l'Airbus A380 comprend 1 million de lignes de code (2007)
 - ▷ contre 100 000 sur les premiers A320 (1987)



Un exemple, en Java

► Recherche par dichotomie dans un tableau trié

```
public static int binarySearch(int[] a, int key) {  
    int low = 0;  
    int high = a.length - 1;  
  
    while (low <= high) {  
        int mid = (low + high) / 2; // bug !!!!!  
        int midVal = a[mid];  
  
        if (midVal < key)  
            low = mid + 1  
        else if (midVal > key)  
            high = mid - 1;  
        else  
            return mid; // key found  
    }  
    return -(low + 1); // key not found.  
}
```

Si $low + high >$
 $2^{31} - 1$ (32 bits)

2006 : Bug présent
dans le JDK
d'Oracle

Première publication de l'algorithme 1946
Prestige publication sans bug
1962

Normes pour les système sûr

- ▶ Normes liée au domaine d'application
 - ▶ Recommandations (REX, best practices)
 - ▷ Pas de recette pour produire les système
 - ▷ logiciel, électronique et processus de développement
 - ▶ Safety case
 - ▷ Démonstration que l'événement redouté ne se produira pas plus fréquemment que prévu
 - ▷ Qualité et bon dévellopement nécessaire
 - ▷ **Safety par conception**
- FDA: santé
 - 26262: automobile
 - EN5012{6,8,9}: ferroviaire
 - IEC61508: industri
 - DO178: aéronotique
 - CC: µelectronics

Certification de système critique

► Certification par un tiers

- ▷ indépendance
- ▷ Engagement de responsabilité

Engineers say Boeing pushed to limit safety testing in race to certify planes, including 737 MAX

May 5, 2019 at 6:00 am | Updated May 5, 2019 at 1:24 pm

► Tout les domaine a part l'automobile ...

Prothèses mammaires PIP : "TÜV Rheinland a travaillé selon la réglementation" et "est victime de l'escroquerie de PIP"

"TÜV Rheinland a travaillé selon la réglementation" et "elle est victime de PIP qui au quotidien mentait et produisait des faux documents pour tromper les auditeurs de TÜV Rheinland"

Normes IEC61508/EN50128

► Logiciels et systèmes critiques:

▷ Préservation de la vie humaine

Les méthodes formelles « Highly recommended » par les normes industrielles pour le développement de systèmes de sécurité de niveau 4 (IEC 61508, EN50128)

Safety Integrity Level	Probability of Dangerous Failure per hour
SIL 4	$\geq 10^{-9}$ to 10^{-8}
SIL 3	$\geq 10^{-8}$ to 10^{-7}
SIL 2	$\geq 10^{-7}$ to 10^{-6}
SIL 1	$\geq 10^{-6}$ to 10^{-5}

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

Technique/Measure	Ref	SIL1	SIL2	SIL3	SIL4
1 Fault detection and diagnosis	C.3.1	---	R	HR	HR
2 Error detecting and correcting codes	C.3.2	R	R	R	HR
3a Failure assertion programming	C.3.3	R	R	R	HR
3b Safety bag techniques	C.3.4	---	R	R	R
3c Diverse programming	C.3.5	R	R	R	HR
3d Recovery block	C.3.6	R	R	R	R
3e Backward recovery	C.3.7	R	R	R	R
3f Forward recovery	C.3.8	R	R	R	R
3g Re-try fault recovery mechanisms	C.3.9	R	R	R	HR
3h Memorising executed cases	C.3.10	---	R	R	HR
4 Graceful degradation	C.3.11	R	R	HR	HR
5 Artificial intelligence - fault correction	C.3.12	---	NR	NR	NR
6 Dynamic reconfiguration	C.3.13	---	NR	NR	NR
7a Structured methods including for example, JSD, MASCOT, SADT and Yourdon	C.2.1	HR	HR	HR	HR

7b Semi-formal methods	Table B.7	R	R	HR	HR
7c Formal methods including for example, CCS, CSP, HOL, LOTOS, OBJ, temporal logic, VDM and Z	C.2.4	---	R	R	HR
8 Computer-aided specification tools	B.2.4	R	R	HR	HR

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.



- ▶ Sécurité des logiciels et systèmes :
 - ▷ Préservation de la confidentialité des données

EAL1 - functionally tested

EAL2 - structurally tested

EAL3 - methodically tested and checked

EAL4 - methodically designed, tested, and reviewed

EAL5 - semiformaly designed and tested

EAL6 - semiformaly verified design and tested

EAL7 - formally verified design and teste

EAL: Evaluation Assurance Level

Normes DO-178 C

► 5 niveaux de criticité (Development Assurance Level) :

- ▷ Niveau A - un problème catastrophique - Sécurité du vol ou atterrissage compromis - Crash de l'avion
- ▷ Niveau B - un problème majeur entraînant des dégâts sérieux voire la mort de quelques occupants
- ▷ Niveau C - un problème sérieux entraînant un dysfonctionnement des équipements vitaux de l'appareil
- ▷ Niveau D - un problème pouvant perturber la sécurité du vol
- ▷ Niveau E - un problème sans effet sur la sécurité du vol

2 paragraphes dans le DO178B à une annexe entière, le DO-333 «Formal Methods Supplement to DO-178C and DO-278A»

Pourquoi les méthodes formelles ?

► On sait calculer la fiabilité d'une résistance

Modèle général associé à la famille

$\lambda = \lambda_{\text{Physique}} \times \Pi_{\text{PM}} \times \Pi_{\text{Process}}$ avec :

$$\lambda_{\text{Physique}} = \lambda_{0_Résistance} \times \sum_i^{\text{Phases}} \left(\frac{t_{annuel}}{8760} \right)_i \times (\Pi_{\text{Thermo}-\text{électrique}} + \Pi_{\text{TCY}} + \Pi_{\text{Mécanique}} + \Pi_{\text{RH}})_i \times (\Pi_{\text{Induit}})_i$$

Facteur $C_{\text{sensibilité}}$

	Sensibilité relative (note sur 10)			$C_{\text{sensibilité}}$
	EOS	MOS	TOS	
Résistance fixe à couche à faible dissipation haute stabilité (RS), d'emploi courant (RC), « Minimelf »	5	2	4	3,85
Résistance fixe à couche à forte dissipation	2	3	1	2,25
Résistance fixe bobinée de précision à faible dissipation	2	1	3	1,75
Résistance fixe bobinée à forte dissipation	2	3	1	2,25
Potentiomètre non bobiné (CERMET)	1	5	2	2,50
Chip résistif	5	4	6	4,75
Réseau résistif CMS	4	5	3	4,25
Résistance de précision, haute stabilité à feuille métallique gravée	6	6	4	5,8

► Mais pas celle d'un système notamment d'un logiciel

- ▷ recours massif aux tests fonctionnels/d'intégration/unitaires
- ▷ qui peuvent être évités/réduits si recours aux méthodes formelles

Que sont les méthodes formelles ?

► Principe des méthodes formelles

- ▷ Utiliser les mathématiques pour concevoir et si possible réaliser des systèmes

Spécification formelle / Vérification formelle / Synthèse formelle

- ▷ Avantage : non ambiguïté des mathématiques !

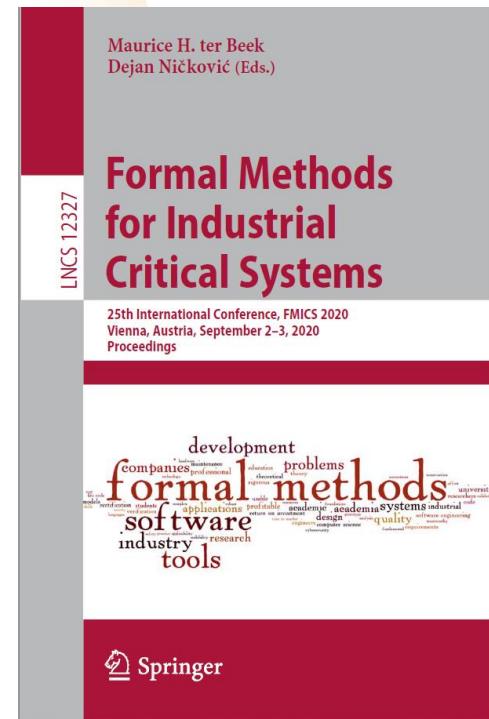
► Objectif global : améliorer la confiance !

- ▷ Dans le logiciel et le matériel

« Program testing can be used to show the presence of bugs, but never show their absence » – Edsger W. Dijkstra

Méthodes formelles

- ▶ Sont des techniques permettant de raisonner rigoureusement
 - ▷ Spécifier en utilisant les mathématiques
 - ▷ Vérifier certaines propriétés sur cette spécification
 - ▷ (parfois) Raffiner ou dériver de la spécification un logiciel concret
 - ▷ (ou parfois) Faire un lien entre la spécification et (une partie d') un logiciel concret
- ▶ 100+ formalismes & outils formels <https://fme-teaching.github.io/>
 - Niv 0: Spécification formelle + informel développement
 - Niv 1: Développement formel et vérification formelle
 - Niv 2: Démonstrateurs de théorèmes pour des preuves entièrement contrôlées par machine
- ▷ 50 années de recherche / 25 années de recherche pour les système sûr industriel



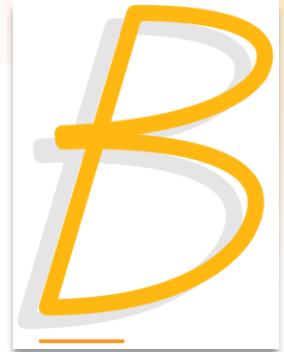
Méthode formelle

METHODE B <http://www.methode-b.com/>

- Inventé par un mathématicien français (J. R. Abrial)
- But : Assigning programs to meanings
- Approche descendante (Top-down)
- Les programmes sont prouvés conformes à leurs spécifications

ATELIER B <http://www.atelierb.eu/>

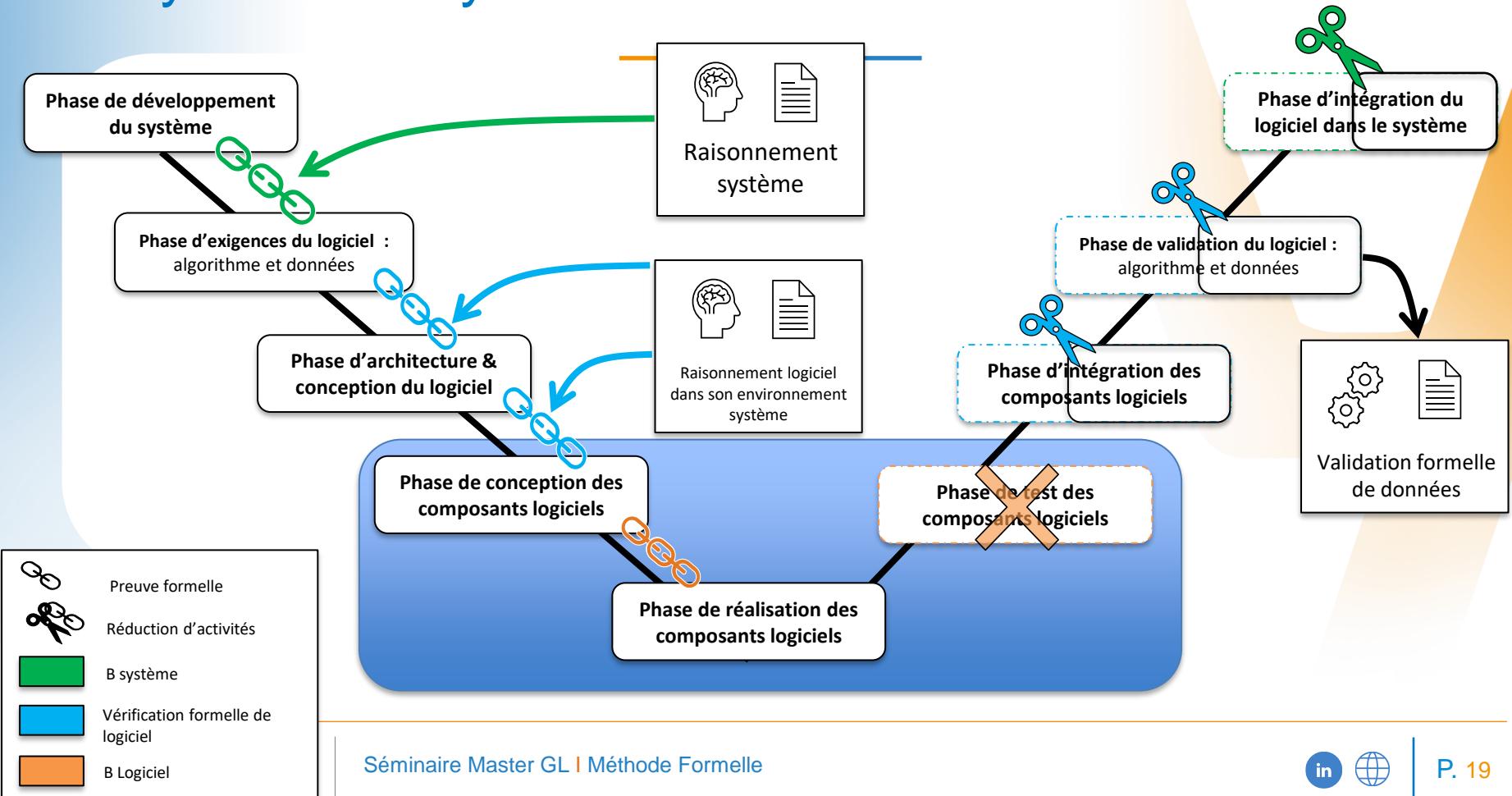
- Atelier de développement de la méthode B
- Premier métro autonome Paris L14 Meteor (1998)
- 30% des métros automatiques dans le monde
- Maintenu & développé par CLEARSY



La méthode B est une méthode formelle

- ▶ Language formel
 - ▷ Théorie des ensembles ($A \subseteq B$)
 - ▷ Logique des prédicat 1er ordres ($P \Rightarrow Q$, existentiel, universel)
 - ▷ Partie Statique : propriété
 - ▷ Partie dynamique : comportement
- ▶ Cohérence : obligations de preuve
- ▶ Raffinement (spécification vers code) : obligations de preuve
- ▶ Vérification déductive
- ▶ Traduction vers langages compilés (C, Ada, etc.)

Cycle en V système issu de la norme EN 50128



Qu'est-ce qu'un programme ?

Il y a plusieurs aspects en jeux :

- ▶ Ce que l'on calcule (**quoi**)
- ▶ La manière de le calculer (**comment**)
- ▶ La raison pour laquelle c'est correct (**pourquoi**)

Qu'est-ce qu'un programme ?

- ▶ Le programme, ce n'est que le « comment », et rien d'autre
 - ▷ Le « quoi » et le « pourquoi » n'en font pas partie

Ce sont des cahiers des charges, des commentaires, des documents, des croquis, de la connaissance du domaine, souvenir

Méthodes formelles : B logiciel

- ▶ Modéliser de façon abstraite du comportement d'un programme
- ▶ Raffinements successifs jusqu'à un modèle concret transcodable
- ▶ Preuve de consistance et de raffinement
- ▶ Génération de code cible.

REFERENCES

- [1] *Météor: A Successful Application of B in a Large Project*
FM 1999, Toulouse
Patrick Behm, Paul Benoit, Alain Faivre, Jean Marc Meynadier
- [2] *Using B as a High Level Programming Language in an Industrial Project: Roissy VAL*
ZB 2005, Guildford
Arnaud Amelot, Frédéric Badeau

« Only inactive sequences can be added to the active sequences execution queue. »

Natural language requirement

```
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 ∪ {sequ}  
  END  
END;
```

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

```
void M0_activation_sequence(void)  
{  
  CTX_SEQUENCES sequ;  
  
  sequence_manager_indexSequenceInactive(&sequ);  
  sequence_manager_activeSequence(sequ);  
}
```

0x01F970	FFFF 8B4C 2440 89C5 8D7D 0C8B 4110 89CE
0x01F980	83C6 0C8D 1485 0000 0000 8D42 0883 F807
0x01F990	7617 F7C7 0400 0000 740F 8B41 0C8D 7D10
0x01F9A0	83C6 0489 450C 8D42 04FC 89C1 C1E9 02F3

B Specification

Behaviour + properties

B Implementation

Behaviour + properties

C generated code

Binary code

« Only inactive sequences can be added to the active sequences execution queue. »

Natural language requirement

```
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 ∪ {sequ}  
  END  
END;
```

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

```
void M0_activation_sequence(void)  
{  
  CTX_SEQUENCES sequ;  
  
  sequence_manager_indexSequenceInactive(&sequ);  
  sequence_manager_activeSequence(sequ);  
}
```

0x01F970	FFFF 8B4C 2440 89C5 8D7D 0C8B 4110 89CE
0x01F980	83C6 0C8D 1485 0000 0000 8D42 0883 F807
0x01F990	7617 F7C7 0400 0000 740F 8B41 0C8D 7D10
0x01F9A0	83C6 0489 450C 8D42 04FC 89C1 C1E9 02F3

Cyclic software single-thread

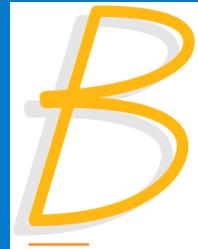
B Specification

B Implementation

C generated code

Binary code

Proof (coherence)



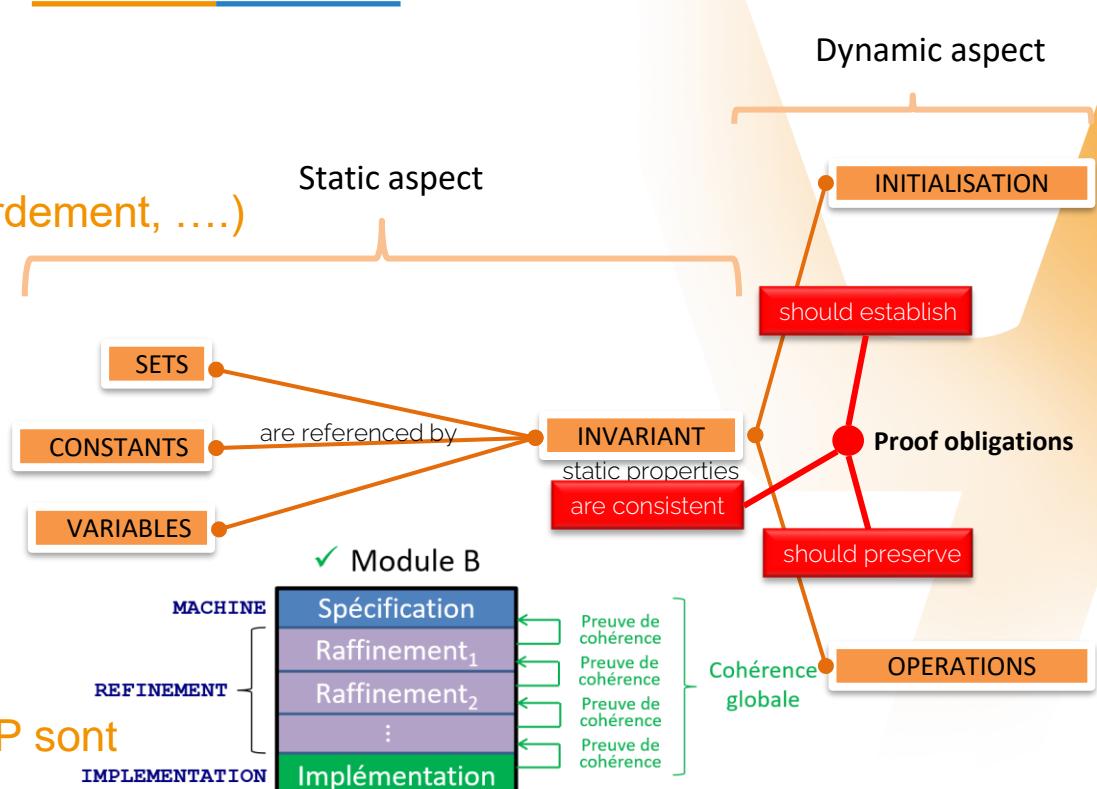
Proof (refinement)

Proof (coherence)

Obligation de preuve (OP)

Automatiquement générer

- ▷ fonctionnel
- ▷ Bonne définition (/0, débordement,)
- ▷ Bonne implémentation
- ▷ Terminaison de boucle
- ▷ Overflow (option)



Un projet B correct

- ▷ Si et seulement si 100% OP sont prouvé

Liste des obligations de preuve

The screenshot shows the Atelier B tool interface with two main windows:

- Left Window (PO selectionnée):** Displays the proof obligation `btrue => xx + 1 : INTEGER`.
- Right Window (M0.mch):** Displays the model code:

```
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 = XX := XX+1
11
12
13 END
```

Annotations with arrows point from the text to specific parts of the code:

- An arrow points from "Liste des obligations de preuve" to the left window.
- An arrow points from "obligation de preuve" to the right window.
- An arrow points from "Partie du modèle liée à l'obligation de preuve" to the line `XX: INTEGER`.
- An arrow points from "Nombre d'obligation de prevue liée à une ligne + couleur (vert: OK, rouge: NOK)" to the line numbers (1-13) and the green background color of the code area.

Partie du modèle liée
à l'obligation de preuve

Nombre d'obligation de prevue liée à une ligne
+ couleur (vert: OK, rouge: NOK)

MACHINE = spécification

M0.mch*

```
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     END
```

IMPLEMENTATION = algorithme

M0.mch M0_i.imp*

```
1      IMPLEMENTATION M0_i
2 -      REFINES M0
3
4 -      CONCRETE_VARIABLES XX
5 1/2      XX
6 -      INVARIANT
7 1/2      XX: INT
8 -      INITIALISATION
9 1/1      XX := 0
10
11 -      OPERATIONS
12 0/1      inc = |
13
14     END
```

Implémente cette spécification

Variable implémentable

Type implémentable

POs on line 12 of M0_i



M0.mch M0_i.imp

inc.1

```
1 IMPLEMENTATION M0_i
2 REFINES M0
3
4 CONCRETE_VARIABLES
5 1/2 xx
6 INVARIANT
7 1/2 xx: INT
8 INITIALISATION
9 1/1 xx := 0
10
11 OPERATIONS
12 0/1 inc = xx := xx + 1 ━━━━ Incrémantation sans fin
13
14 END
```

OP sélectionnée:M0_i.inc.1



btrue &

btrue

=>

xx\$1 + 1 : INT

Si on incrémenter trop on sort du type implémentable

POs on line 12 of M0_i



inc.1
inc.2
inc.3

M0.mch M0_i.imp

```
1 IMPLEMENTATION M0_i
2 - REFINES M0
3
4 - CONCRETE_VARIABLES
5 3/4     xx
6 - INVARIANT
7 3/3     xx: INT
8 - INITIALISATION
9 1/1     xx := 0
10
11 - OPERATIONS
12 2/3     inc = IF xx = MAXINT THEN xx := 0 ELSE xx := xx + 1 END
13
14 END
```

On réinitialise si on arrive à la borne supérieure (MAXINT)
Sinon on incrémente

OP sélectionnée:M0_i.inc.1



```
btrue &
xx$1 = MAXINT &
btrue
=> [
xx$1 + 1 = 0
```

Si xx vaut la limite supérieure, la xx devrait être à 0

M0.mch M0_i.imp

```
1- MACHINE
2-   M0
3- VARIABLES
4-   XX
5- INVARIANT
6 3/3   XX: INTEGER
7- INITIALISATION
8 1/1   XX := 0
9- OPERATIONS
10 2/2   inc = CHOICE XX := 0 OR XX := XX + 1 END
11
12 END
```

Soit on incrémenté soit on réinitialise

Les deux modèles
sont compatibles

M0.mch M0_i.imp

```
1- IMPLEMENTATION M0_i
2- REFINES M0
3-
4- CONCRETE_VARIABLES
5 5/5   XX
6-
7 3/3   XX: INT
8-
9 1/1   XX := 0
10
11- OPERATIONS
12 4/4   inc = IF XX = MAXINT THEN XX:=0 ELSE XX := XX +1 END
13 END
```

Génération du code B

Le code logiciel est généré à partir du modèle Le code est lisible, très proche du modèle et se vérifie facilement

```
M0.mch M0_i.imp
1 IMPLEMENTATION M0_i
2 REFINES M0
3
4 CONCRETE_VARIABLES
5 5/5      xx
6 INVARIANT
7 3/3      xx: INT
8 INITIALISATION
9 1/1      xx := 0
10
11 OPERATIONS
12 4/4      inc = IF xx = MAXINT THEN xx:=0 ELSE xx := xx +1 END
13 END
```

```
11 static int32_t M0_xx;
12 /* Clause INITIALISATION */
13 void M0_INITIALISATION(void)
14 {
15     M0_xx = 0;
16 }
17
18 /* Clause OPERATIONS */
19
20 void M0_inc(void)
21 {
22     if(M0_xx == 2147483647)
23     {
24         M0_xx = 0;
25     }
26     else
27     {
28         M0_xx = M0_xx+1;
29     }
30 }
31 }
```

Pourquoi utiliser la méthode B?

► Détection rapide des erreurs

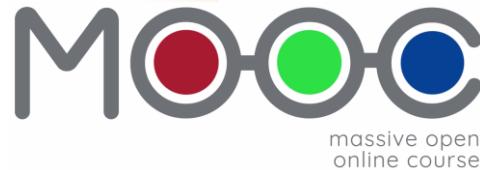
- ▷ Les erreurs sont découverte durant la modélisation et non une fois le logiciel construit.

► Une partie des tests est inutiles

- ▷ Les tests sont remplacés par de la preuve.
- ▷ La preuve est exhaustive, contrairement au test

► Garentie: pas d'erreur de programmation

- ▷ Pas de débordement,
- ▷ Pas de division par zéro,
- ▷ Pas d'accès à une table en dehors de son domaine



massive open
online course

<https://mooc.imd.ufrn.br/>



Lecture 0: Marketing video

This video explains why you should follow the MOOC on B and what its expected benefits on your career are.

Level: Basic

Video duration: 02:55



Lecture 1: Course Introduction

This video presents the structure of the course, provides an overview of the different kinds of formal methods and specification styles, and tells us some myths on formal methods

Level: Basic

Video duration: 08:43

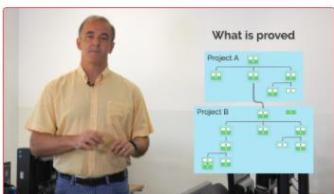


Lecture 2: Overview of the B method

This video briefly introduces the tool Atelier-B, the B and Event-B languages, and some industrial references. The main concepts of B are exposed.

Level: Basic

Video duration: 15:03



Lecture 3: The concepts of B

This video presents the founding notions of B: projects, libraries, modules, components, abstract machine, refinement, implementation, and proof.

Level: Basic

Video duration: 09:29



Lecture 4 : introduction to Abstract Machines

This video introduces the notion of abstract machines, based on an example that is verified, animated and for which C source code is generated.

Level: Basic

Video duration: 16:31



Méthodes formelles : B logiciel

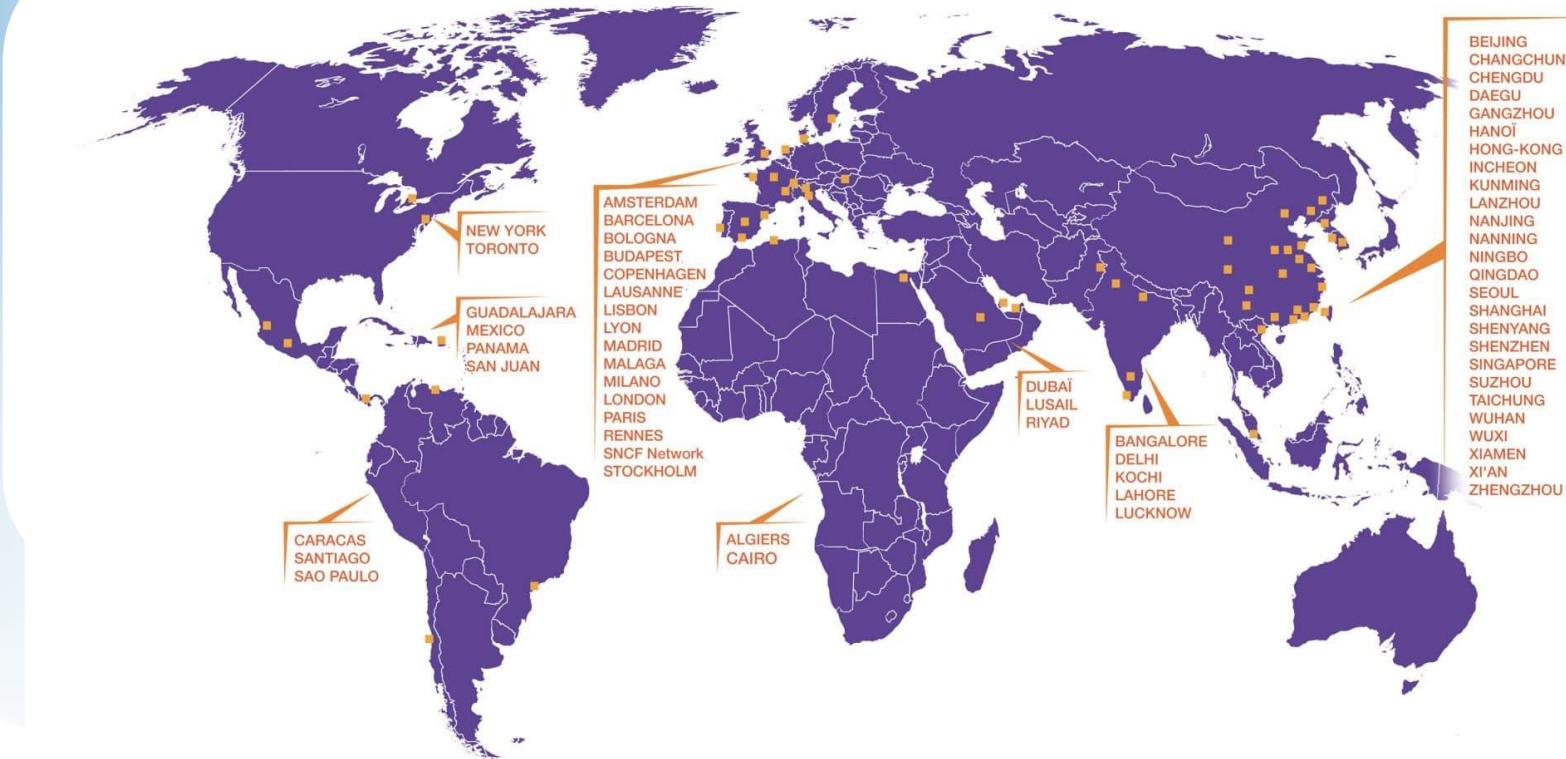
Développement formel de logiciel pour la ligne 14 de la RATP. (SACEM en 1998)

► Aucuns BUG découvert après la preuve :



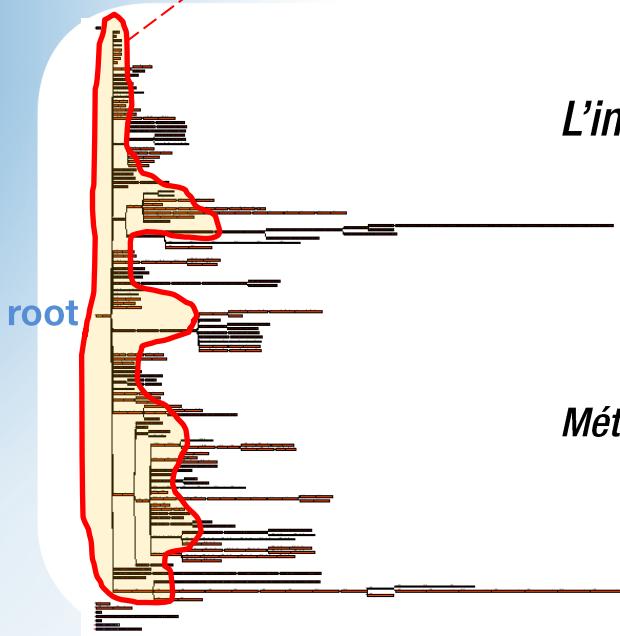
- Ni lors des Test, d'intégration, fonctionnel, sur site
- Ni depuis que la ligne est en opération : 23 ans

30% des métros automatique



Abstract model

Métrique sur un Métro



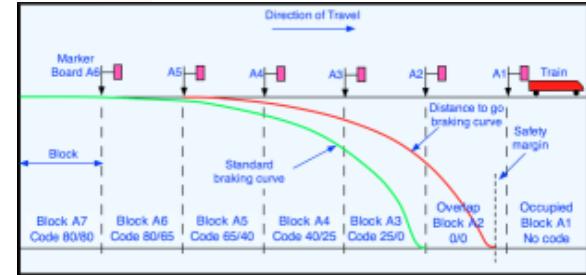
Logiciel du automatique train protection (2015)

Métrique

- 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)

L'implementation de la racine

- Imports 55 composants
- La racine specify un cycle fonctionnel :
 - Compute location, manage kinetic energy, control PSD, trigger emergency braking, etc.

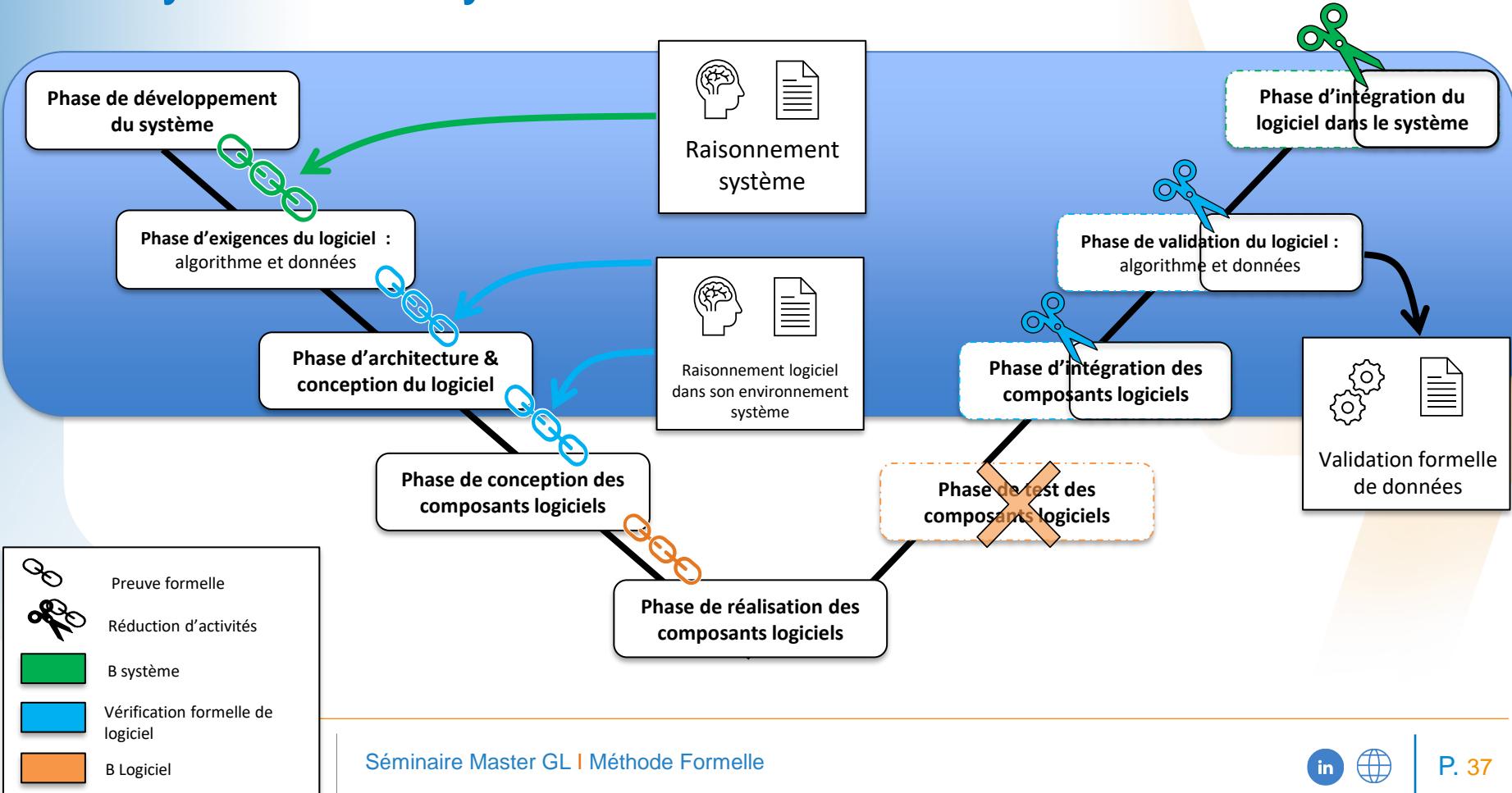


Braking curves

Cette partie du logiciel couvre le fait de freiner si non autorisé a avancer.
Eviter la collisions est un besoin système

La spécification n'est pas complètement écrite dans la racine

Cycle en V système issu de la norme EN 50128



Méthodes formelles : B système

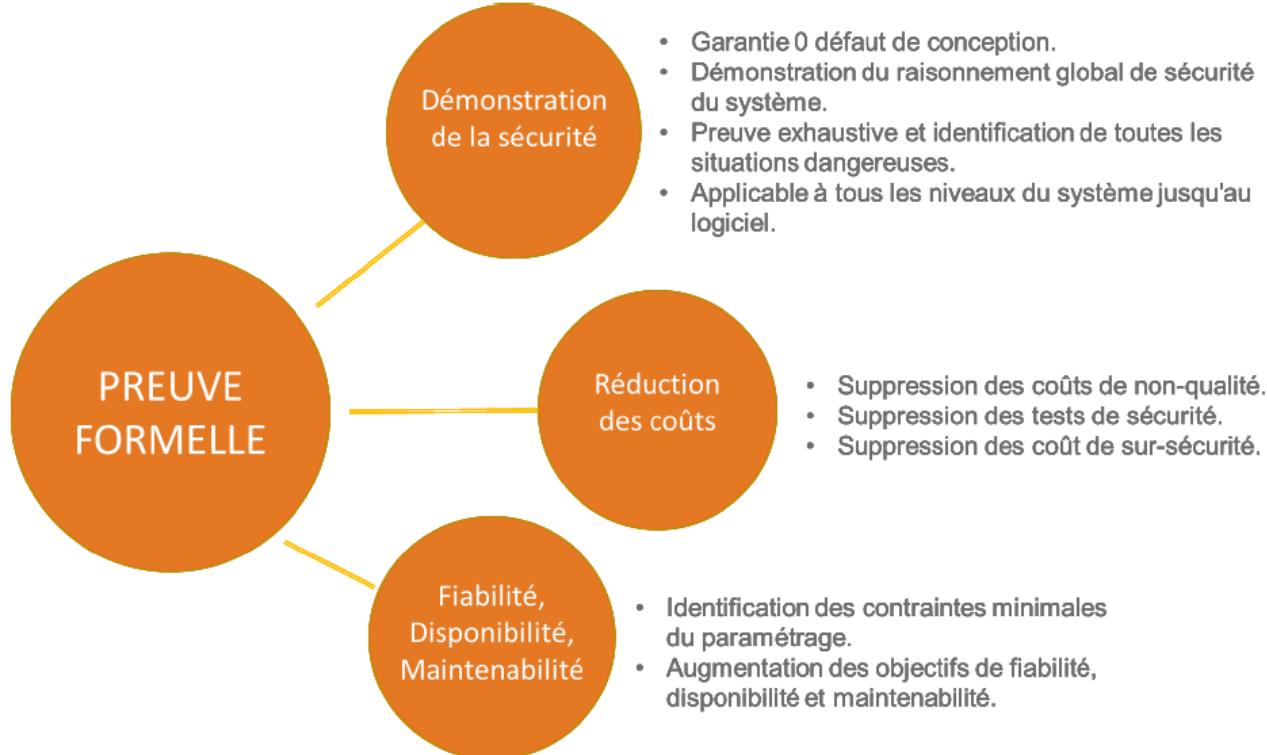
► Analyse formelle de système

- ▷ Modélisation du raisonnement utilisé pour la conception sûre
- ▷ Utile pour des systèmes à mi-vie

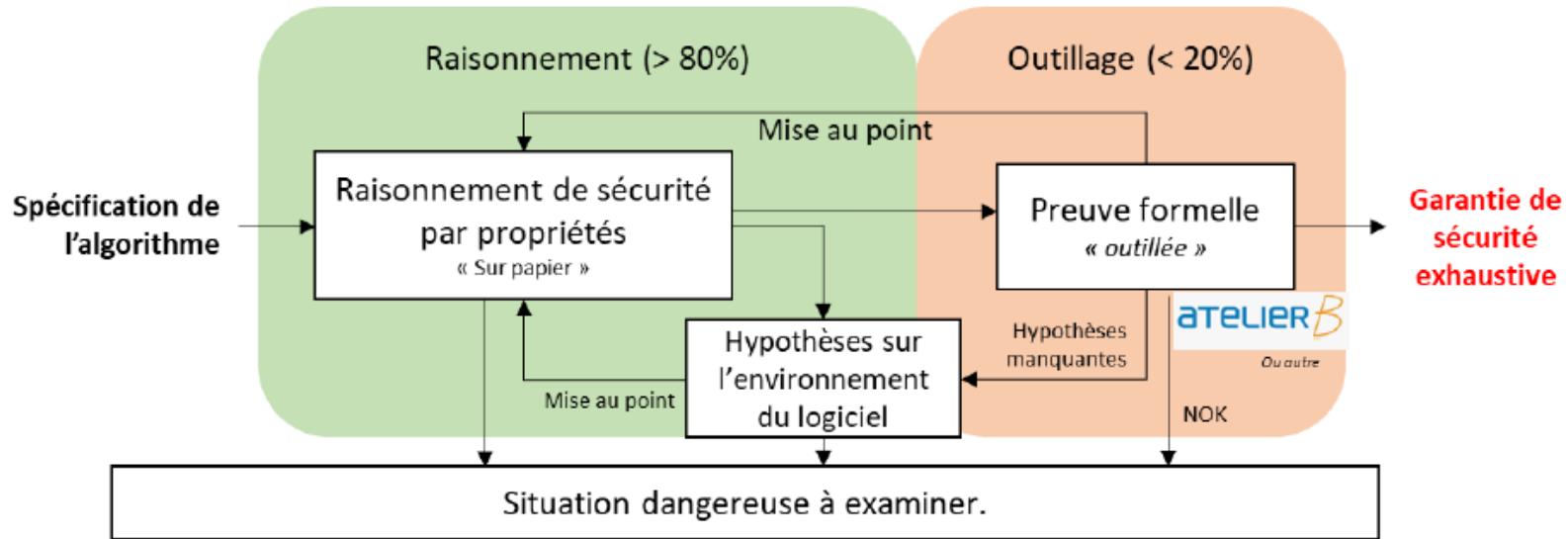
REFERENCES

- [1] *Using formal proof and B method at system level for industrial projects.*
RSSR 2016, Paris
Denis Sabatier
- [2] Safety Analysis of a CBTC System: A Rigorous Approach with Event-B
RSSR 2017, Pistoia
Mathieu Comptier, David Deharbe, Julien Molinero Perez, Denis Sabatier
- [3] Property-Based Modelling and Validation of a CBTC Zone Controller in Event-B
In book: Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification
Mathieu Comptier, Michael Leuschel, Luis-Fernando Mejia, Julien Molinero Perez, Mareike Mutz

Méthodes formelles : B système



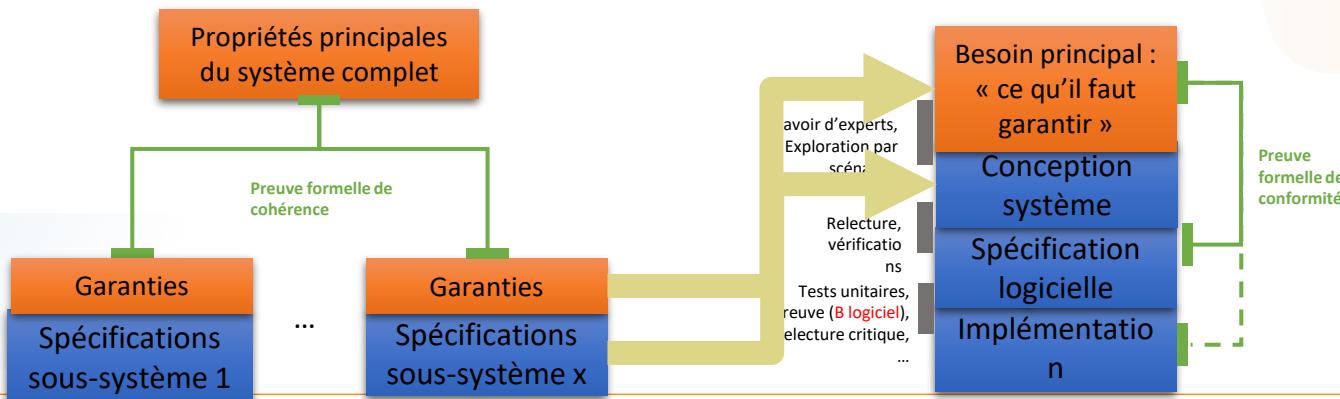
Méthodes formelles : B système



Méthodes formelles : B système

Position des études B-système

- ▶ Ces études formelles agissaient au niveau inter-système.
 - ▷ « Si le sol garantit que ...
 - ▷ Si le bord garantit que ...
 - ▷ Si la sig garantit que ...
 - ▷ Alors il n'y aura **pas de collision ni de déraillement.** »



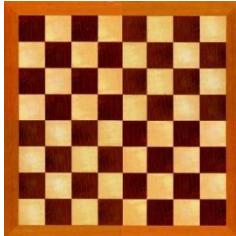
Position de l'analyse formelle

- ▶ La méthode présentée se place à un niveau inférieur.
- ▶ Mais au dessus du **B-logiciel**.
- ▶ Le sol doit garantir qu'il ne peut pas perdre un train.
 - ▷ « **Le fait-il vraiment ?** »

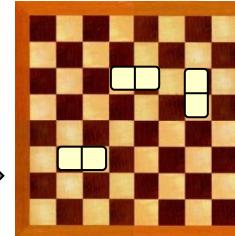
Méthodes formelles : B système

► L'exemple célèbre de Dijkstra :

▷ Conception :



On place des dominos en couvrant les cases



▷ La Propriété suivante est elle vraie ? : les dominos ne couvriront jamais tout le damier sauf exactement la case bas / gauche et la case haut / droit



▷ Raisonnement : si B et W sont le nombre de cases black / white non couvertes, $B = W$ tout le temps à chaque fois que l'on place un nouveau domino

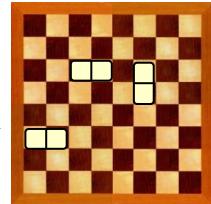
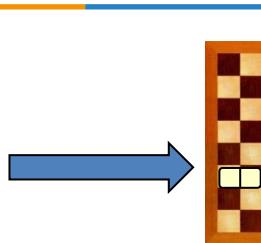
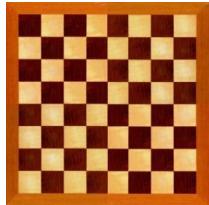
► Car chaque domino couvre toujours un blanc et un noir

Donc atteindre un état où $W = 2$ et $B = 0$ est impossible, donc la propriété est VRAIE

► Le raisonnement est simple, lisible et vérifiable

▷ Plus simple que si on explore tous les cas

Méthodes formelles : B système



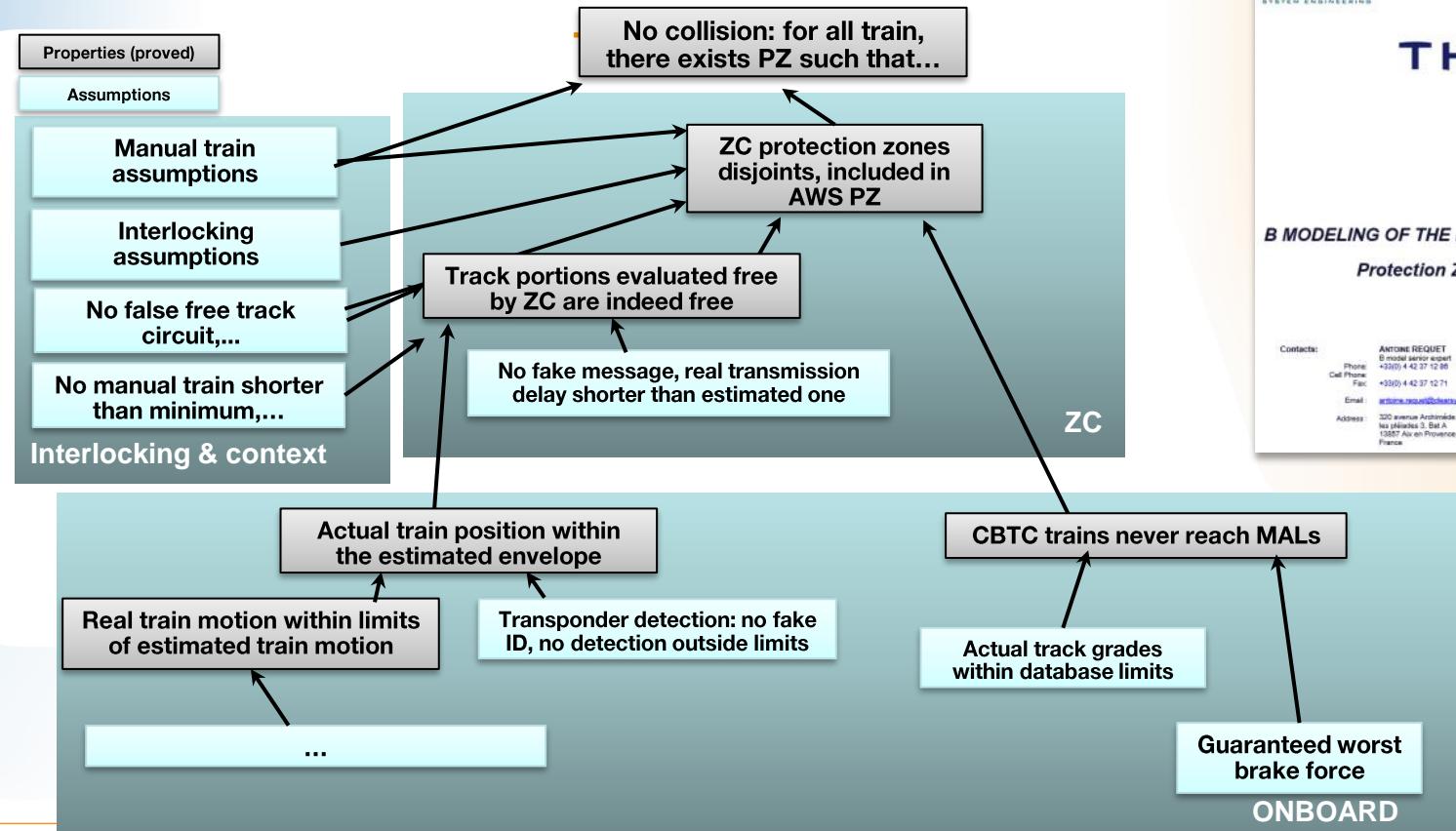
- ▶ **Le damier = le contexte projet**
Eléments préexistants dans lesquels le nouveau système s'intègre
- ▶ **Jamais couvrir tout sauf 2 coins = imaginons que cette propriété soit voulue**
Ce pourrait être une propriété de sécurité par exemple...
- ▶ **Employer des dominos couvrant 2 cases adjacentes = la solution de l'industriel**
S'il l'a choisie, c'est bien qu'il a une certitude que cette solution garantit la propriété voulue
- ▶ **La clef de raisonnement « à chaque domino ajouté, le nb de cases blanches et noires non couvertes reste égal » = élément de la démonstration de sécurité**
Qui doit trouver cette démonstration ? L'industriel à l'origine de la conception ...
Mais la démonstration doit être vérifiable indépendamment
- ▶ *Parallèle forcément un peu caricatural sur un tel exemple « jouet »*

Méthodes formelles : B système

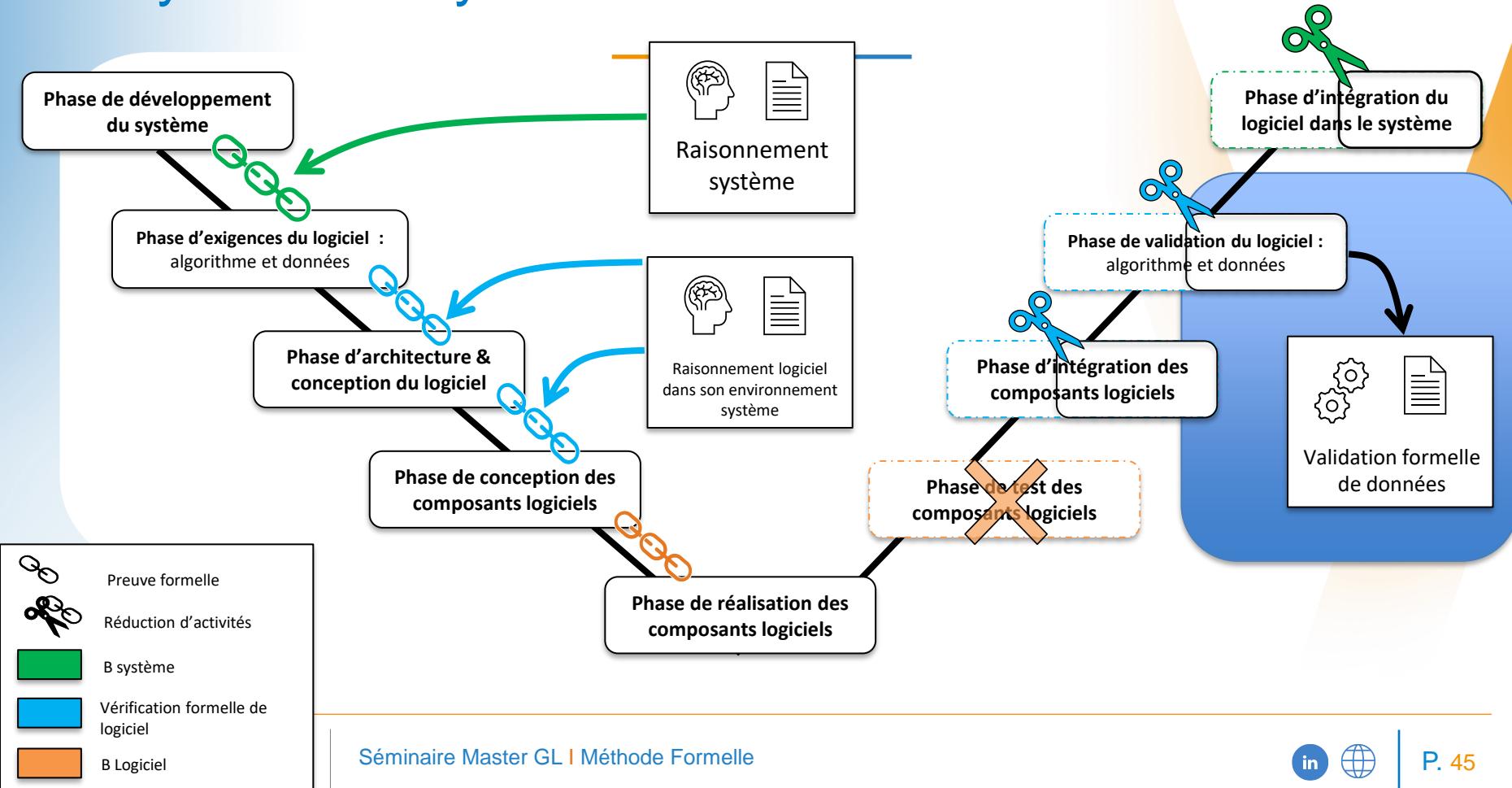
B MODELING OF THE FLUSHING LINE CBTC

Protection Zones Proof

Contacts:	ANTONIE REQUEST	Denis SABATIER
Phone:	+33(0) 4 42 37 12 80	Teléfono directo: +33(0) 6 15 55 41 74
Cell Phone:	+33(0) 4 42 37 12 71	+33(0) 6 15 55 41 91
Fax:		+33(0) 4 42 37 12 71
Email:	antonie.request@clearsy.com	denis.sabatier@clearsy.com
Address:	320 avenue Aristide Briand les plénières 3, Bat A 13857 Aix en Provence cedex 3 France	



Cycle en V système issu de la norme EN 50128



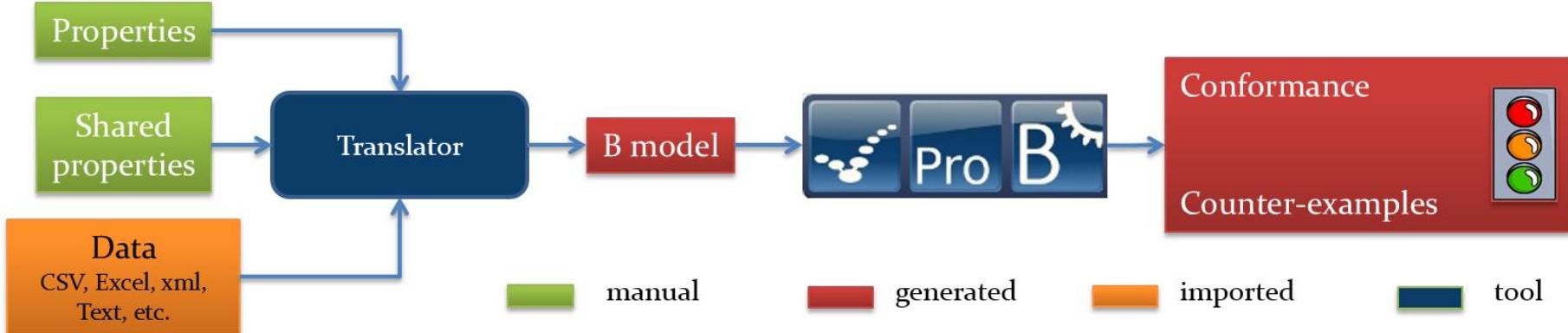
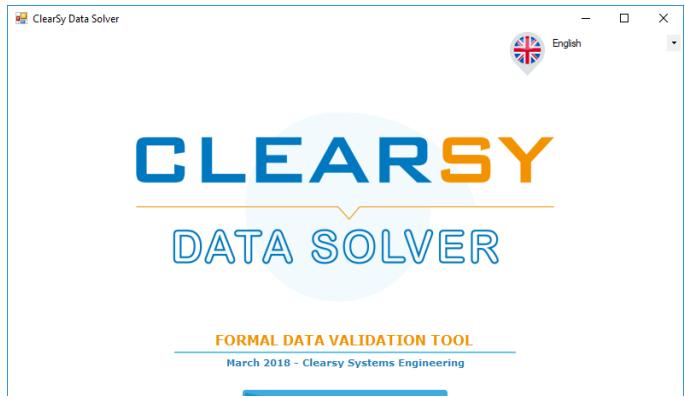
Méthode Formelle et vérification de Donnée

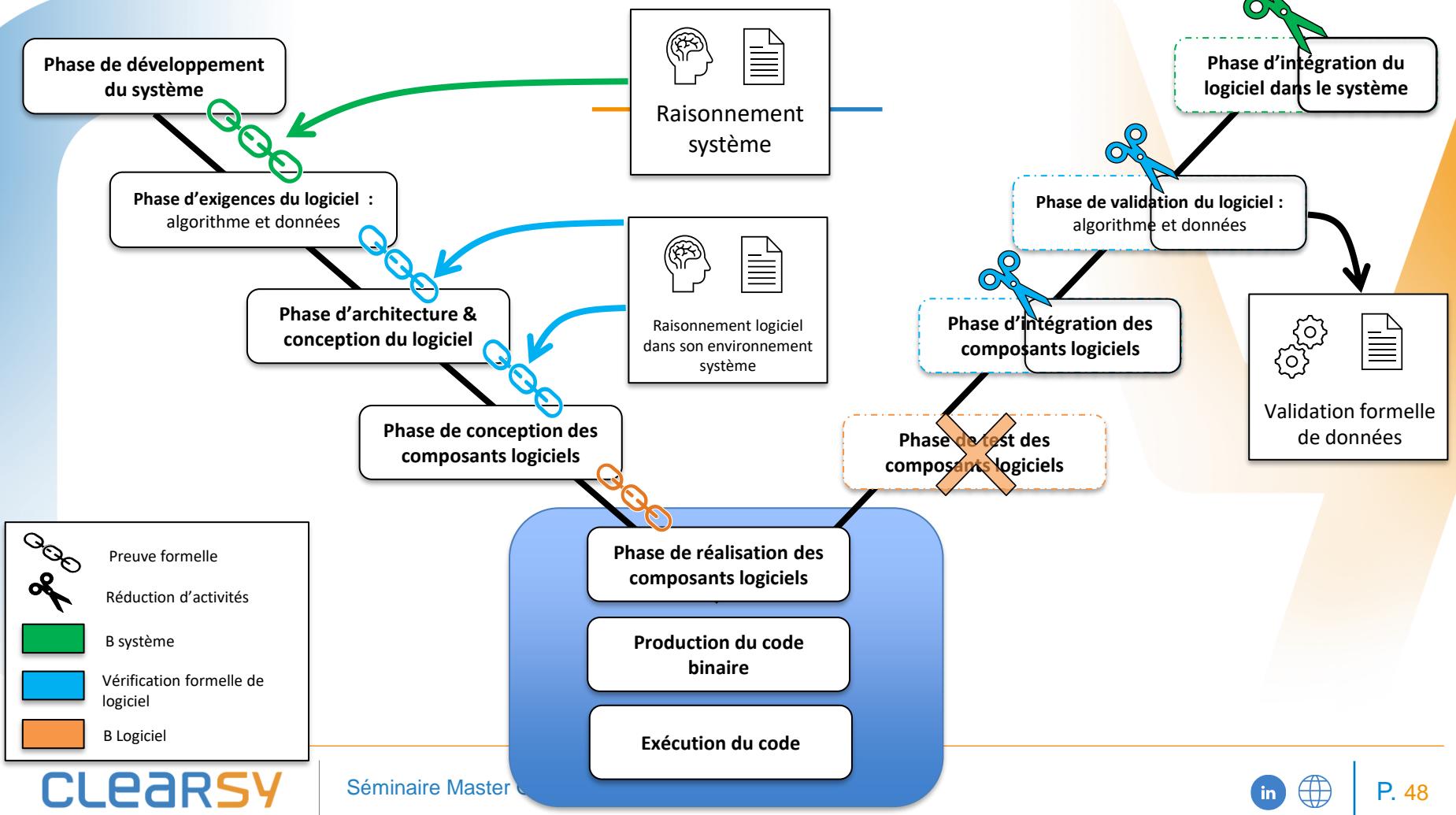
- ▷ Utilisée pour en vérifier mathématiquement les propriétés sur les données
 - ▶ Formalisation des données (consistantes, corrections)
 - ▶ Validation d'un ensemble de données partiellement construit
 - ▶ Mise en évidence des contre-exemples de manière simple
- ▷ Utilisation courante dans le monde industriel : Alstom, RATP, Siemens, Thales, GE,

REFERENCES

- [1] *Formally Checking Large Data Sets in the Railways*
ICFEM 2012, Kyoto
Thierry Lecomte, Lilian Burdy, Michael Leuschel
- [2] *Formal Data Validation in the Railways*
SSS'16, Brighton
Erwan Mottin, Thierry Lecomte

« A method without tool support is useless »





Clearsy Safety Platform



► Cœur de calcul Sûr programmé avec du B

- ▷ IDE et une plate-forme matérielle qui intègre nativement les principes de sécurité
- ▷ Pour développer des applications cycliques sans OS sous-jacent

REFERENCES

- [1] *The CLEARSY Safety Platform: 5 Years of Research, Development and Deployment*
SBMF 2019, São Paulo

Thierry Lecomte, David Deharbe, Paulin Fournier, Marcel Oliveira

Clearsy Safety Platform



► Sûreté en hardware

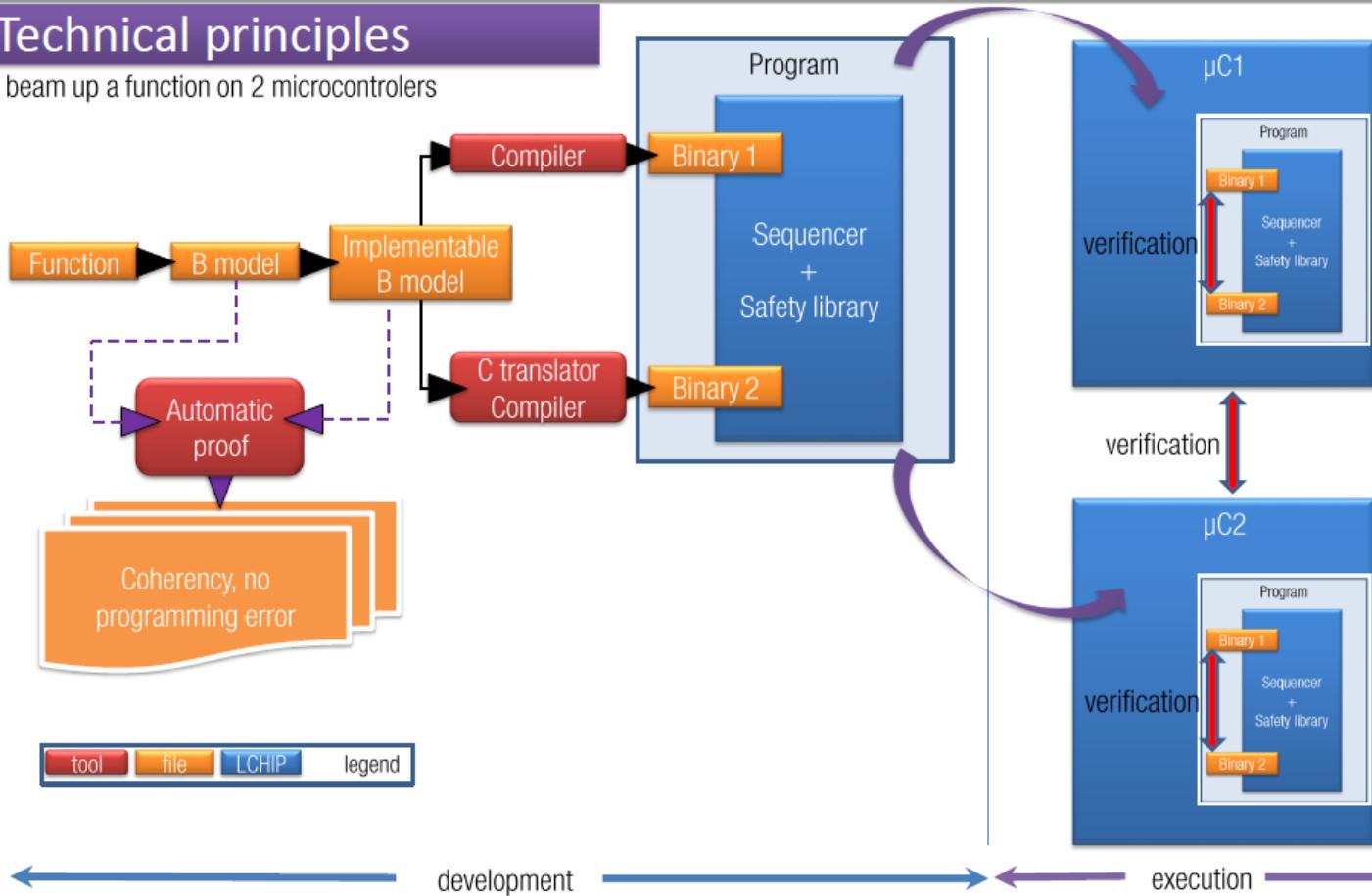
- ▷ Basé sur du 2002 PIC32 microcontrollers
- ▷ Offre jusqu'à 80 MIPS pour les applications
- ▷ Toutes les interfaces de PIC32 sont disponibles pour adresser les E/S, analogiques, bus de communication , ...

► Sécurité sur le logiciel

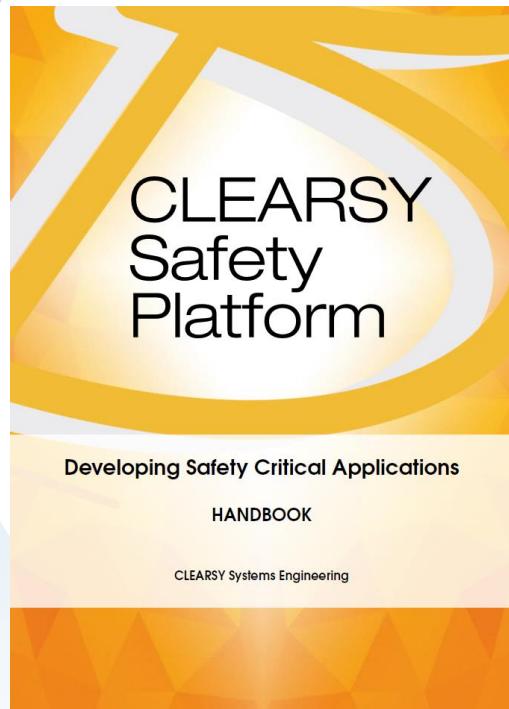
- ▷ Basé sur du logiciel 4oo4
- ▷ L'exactitude est assurée par la preuve mathématique
- ▷ Contrôles croisés entre les instances logicielles et entre les microcontrôleurs

Technical principles

To beam up a function on 2 microcontrollers



Dissémination



► Cours/recherches en cours

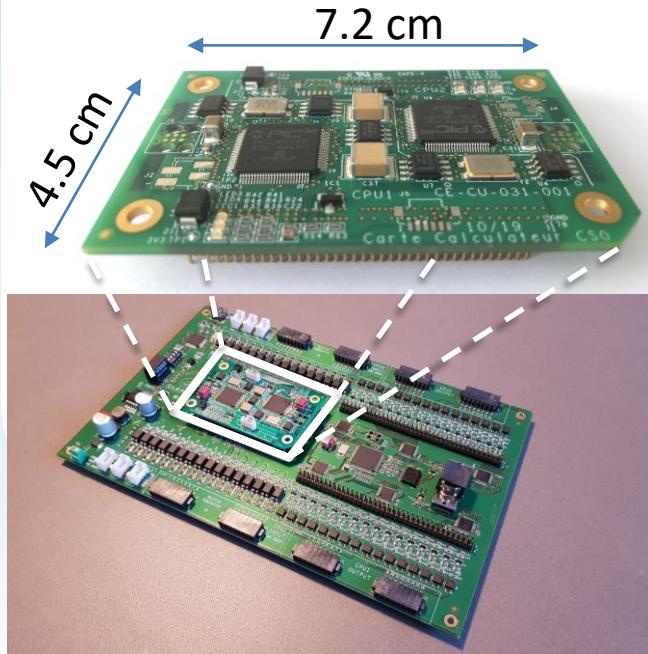
- | | |
|--------------------------------|--------|
| ▷ CentraleSupélec Saclay | France |
| ▷ Université Paris-Est Créteil | France |
| ▷ LORIA Nancy | France |
| ▷ Sherbrooke University | Canada |
| ▷ University of Firenze | Italy |
| ▷ UFRJ Rio | Brazil |
| ▷ UFF Niteroi | Brazil |
| ▷ UFRN Natal | Brazil |
| ▷ IFRN Parnamirim | Brazil |

► Documentation collaborative

- ▷ Livre, projets pratiques, maquettes
- ▷ <https://github.com/CLEARSY/CSSP-Programming-Handbook>

Industriel

- Secure gateways
- Autonomous shuttles
- Connected Medical devices
- Remotely controlled robots



► Safety Computer 0

- ▷ Core safety computer
- ▷ SIL4 certificate, certification kit (exported constraints)
- ▷ Smartcard format
- ▷ Daughter board (no power supply, no I/O)

► Industry-ready release

- ▷ Plugged on a motherboard (64x I/O)
- ▷ Similar safety principles
- ▷ More freedom to the developer:
functional C code
B0 code for safety parts
code for main loop or for interrupt

Clearsy Safety Platform

- ▶ La CLEARSY Safety Platform a été certifier SIL4 par CERTIFER
 - ▷ Certificate n°9594/0262



Certificat de type
Par examen de la conception
Design examination type certificate

N° 9594/0262 édition 1

Délivré à
Attributed to
CLEARSY
320 Av. Archimède - Pléiades III
F-13100 Aix-en-Provence
par
by
CERTIFER
18 Rue Edmond Membrée
F-59300 VALENCIENNES

qui certifie que la conception du produit suivant :
which certifies that the design of the following product:

GENERIC PRODUCT
CLEARSY SAFETY PLATFORM
(D270 REV. 01.01)

forme aux exigences SIL4 des normes CENELEC EN 50126 :2017, EN 50129 :2018,
EN 50128 :2011.

ie SIL4 requirements of the standards CENELEC EN50126:2017, EN50129:2018, EN50128:2011.

L'annexe EC_9594_0263 version 1 fait partie intégrante du présent certificat
This certificate includes appendix EC_9594_0263 version 1

s'applique qu'à la conception du produit référencé en annexe et au dossier descriptif en résultant.
its certificate is limited to the design of the product referenced in the appendix and its description file.

La certification a été conduite en conformité avec le référentiel CERTIFER RF0015 version 3.
its certification was performed in accordance with CERTIFER repository RF0015 version 3.

Signature : 11 Janvier 2021
Signature: January 11th, 2021

Délivré à Valenciennes le 11/01/2021
Issued at Valenciennes

Le Directeur Général
The Chief Executive Officer
Pierre KADZIOLA



Platform screen doors



Platform screen doors: a safer system

≡ Installation on site



Platform screen doors controller installed
in Stockholm (Citybanan)

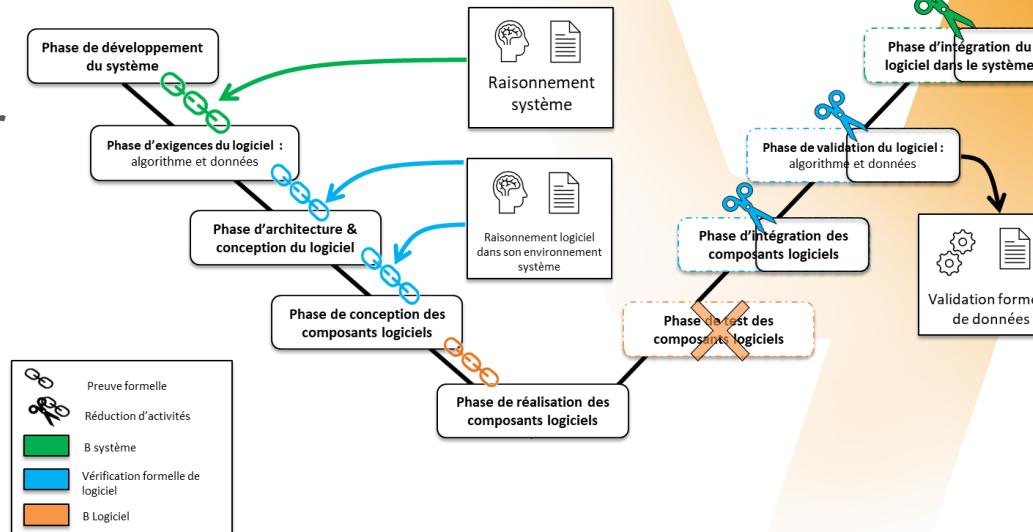
≡ Certification



Conclusion

- ▶ Utilisation rationnelle des méthodes formelles
 - ▷ Quand et où c'est utile :
Quelqu'un peut être blesser/tuer
money to be lost
 - ▷ Nous devons continuer à savoir pourquoi nous faisons les choses
 - ▷ Les méthodes formelles ne sont qu'une (petite) partie du tableau

Cycle en V système issu de la norme EN 50128



Perspectives

- ▶ Besoin croissant – notamment liées à la cybersécurité
 - ▷ Explosion de la population urbaine
 - ▷ Numérisation croissante
 - ▷ Les accidents sont de moins en moins acceptable
- ▶ « Autonomisation »
 - ▷ Les métros sont autonomes depuis les années 90 (ligne 14 depuis 1998)
 - ▷ Tous les projets ferroviaires français doivent intégrer des méthodes formelles au moins au niveau du système (y compris le Grand Paris pour 2024)
 - ▷ Trains, avions, voitures, navettes, robots, etc. restent des problèmes entiers

Remplacer l'humain par l'ordinateur n'est pas suffisant



Safety Solutions Designer

AIX
LYON

Made in France

Actualités

CLEARSY

R&D

On recrute

Contact

Offres d'emploi

Offres de stage

Nos métiers

Témoignages

CLEARSY
Safety Solutions Designer

Savoir-faire ▾

Thématisques

Offres

Composants

Outils

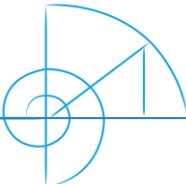
Référen-



<https://www.clearsy.com/on-recrute/offres-de-stage/>



<https://mooc.imd.ufrn.br/>



ETIENNE.PRUN@CLEARSY.COM

Séminaire Master GL
OCT 2021