Instituts thématiques

Inserm

Institut national
de la santé et de la recherche médicale

Toward a Specific Software Development Process for High Integrity Systems

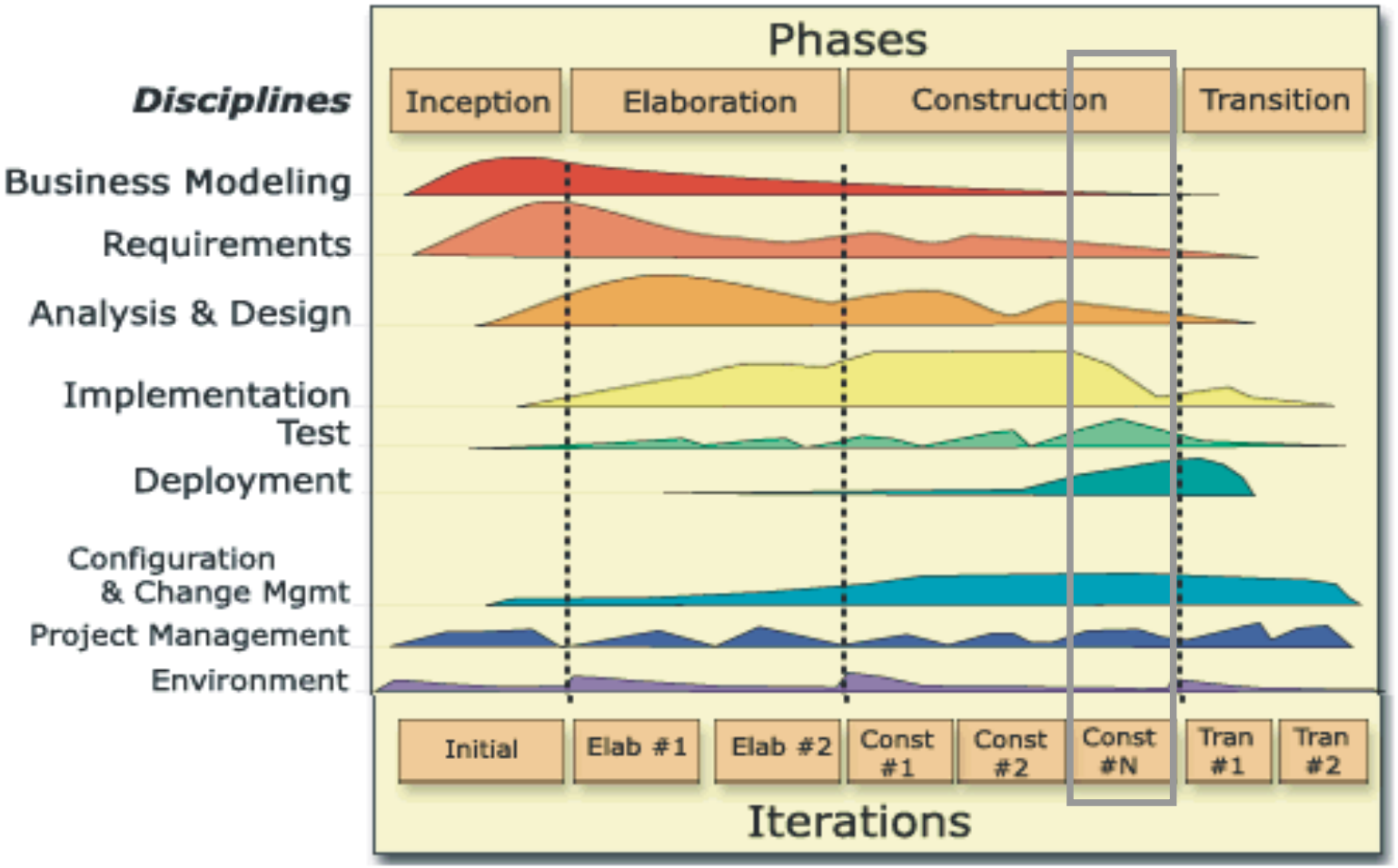
Isabelle Perseil (INSERM)

Roadmap

- -1- Introduction
- -2- RUP advantages / shortcomings
- -3- Real-time languages and best practices
- -4- Formal methods integration in real-time software development
- -5- From RUP to the C-Method software development process
- -6- Conclusions

- The main tool of project management
- An integrated process in a methodological approach
- State-of-the-art of good practices in software process development techniques for DRES
- RUP and DRES
- Discrepancy between
  - Evolution of modeling languages, practices of model transformation and verification AND
  - Evolution of the processes which use them during the phases of requirements specification, analysis, design and certified code generation
- Very rare integration approaches in industrial environments
  - B-Method at RATP
  - Esterel at Dassault
  - Intensive use of PVS at NASA
- **Our approach**
  - To enrich the current process with **other phases**
  - Consider that the requirements of strategic type must first be completely identified, specified, verified
  - Parallelization of sub-processes
  - A seamless development involving intermediate languages

1 / RUP iterations



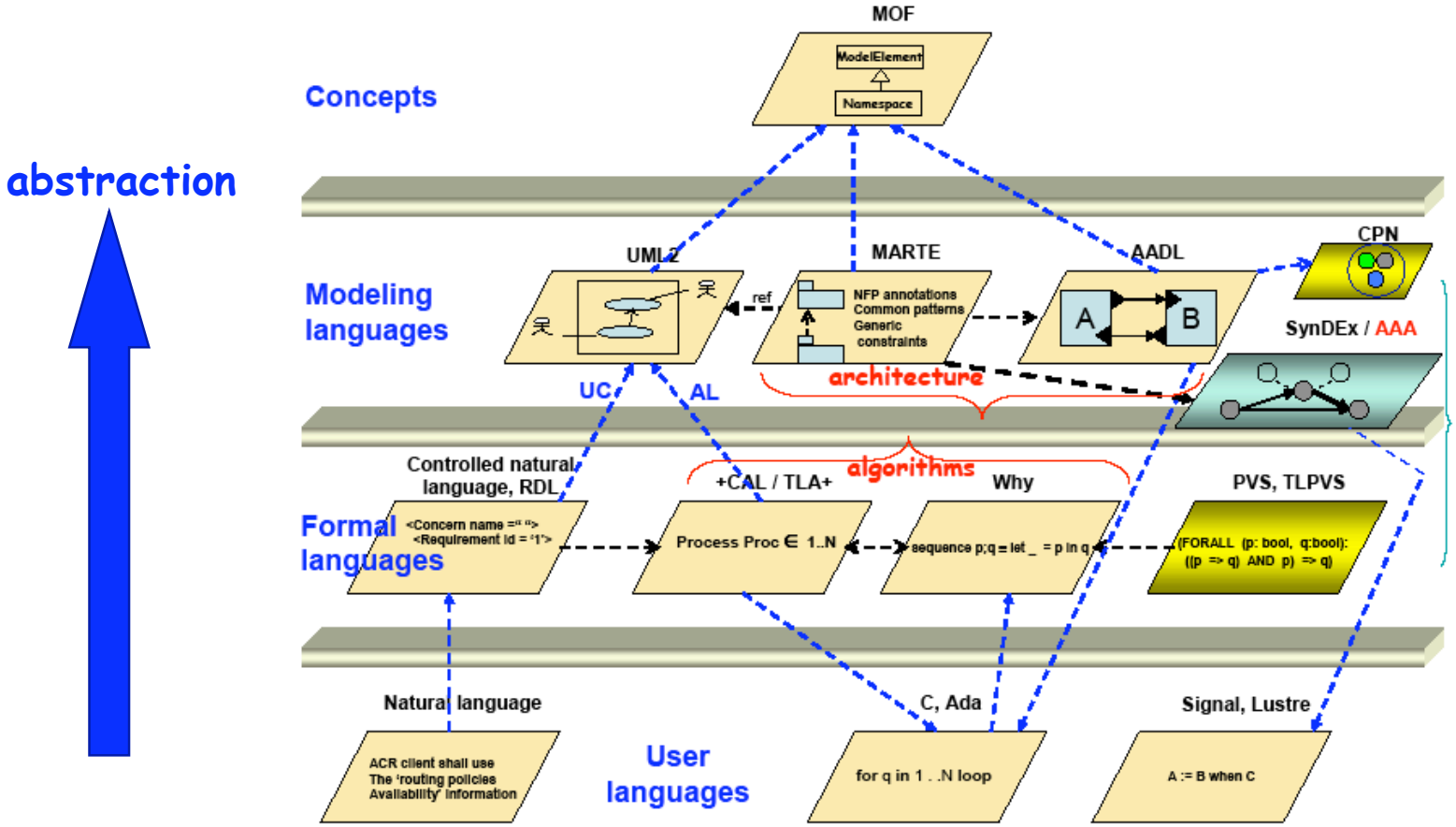
2/ RUP advantages

- IBM Rational definitely banished the waterfall process
  - Unfortunately this good resolution have not been followed by everyone, even in the research field.
- The ``use case driven'' approach is definitively a very good approach that is even kept in the Agile methods
  - allows the requirements to be traced
- The ``architecture-centric'' process is adopted for all complex and large systems
- The possible customization enables an adaptable process framework in which each company may choose the most convenient elements.

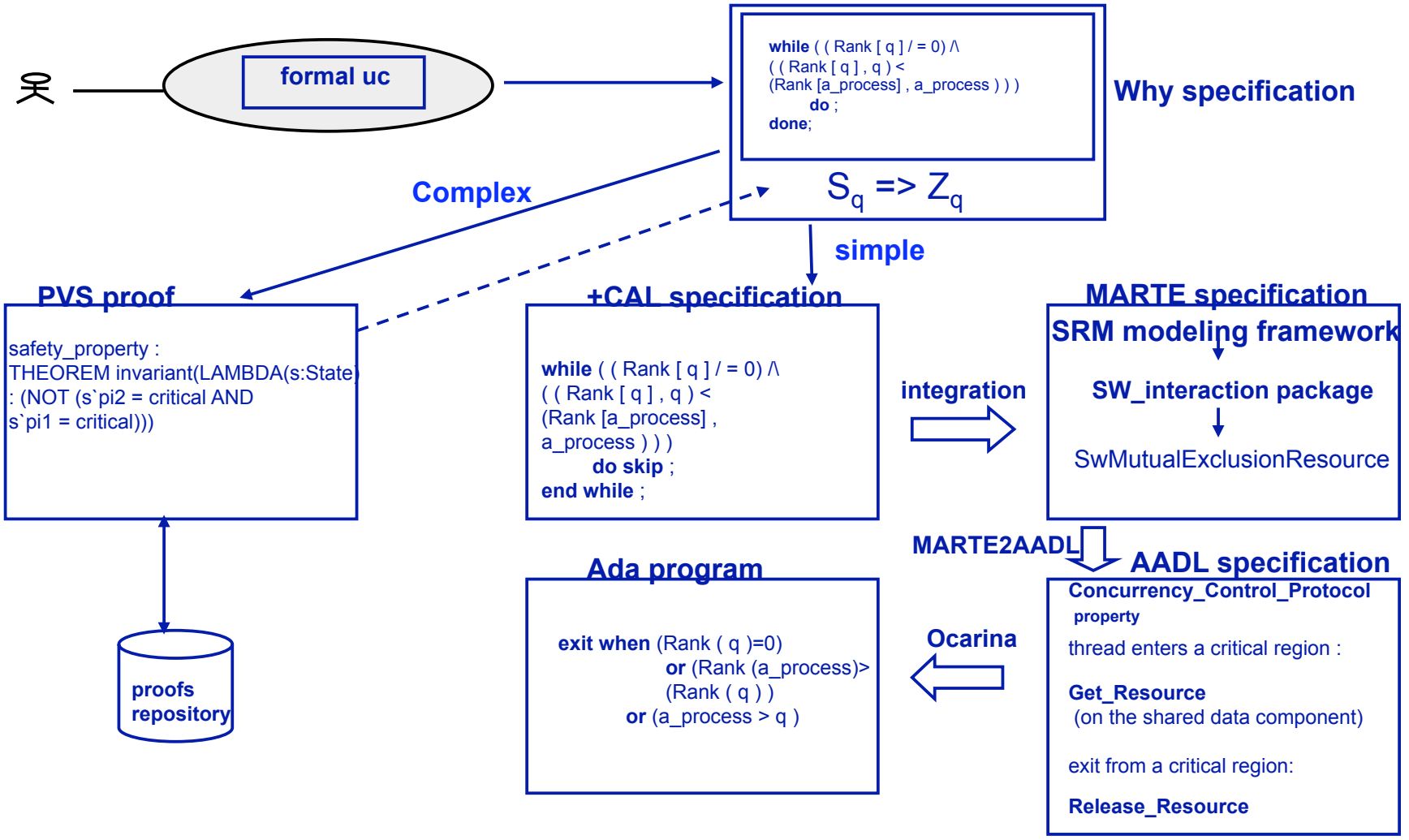
3/ RUP shortcomings

- The homogeneous decomposition between Inception Elaboration and Construction is too much simplistic
  - Because depending on activities types, cycles are more or less complex, therefore not homogeneous
- The RUP is supported by a very heavy tool, which is not intuitive
  - The learning period is long and requires significant investments
- Depending on the environment, the parameterization may also be very long
  - the parameterization gives the impression of genericity,
  - but the process is not fundamentally different for a any kind of project (telecommunications, automotive, aeronautics, financial, etc) : the phases and activities are the very same.
- The RUP is only suitable for very big projects
  - its intrinsic logic is so much linked to the IBM Rational world that it is mostly applied with the entire tool suite.
- The entire process is rather a set of good recipes than the result of a rigorous ``rationale'' as the name should suggest
  - The inception phase should be global with respect to a systemic approach.

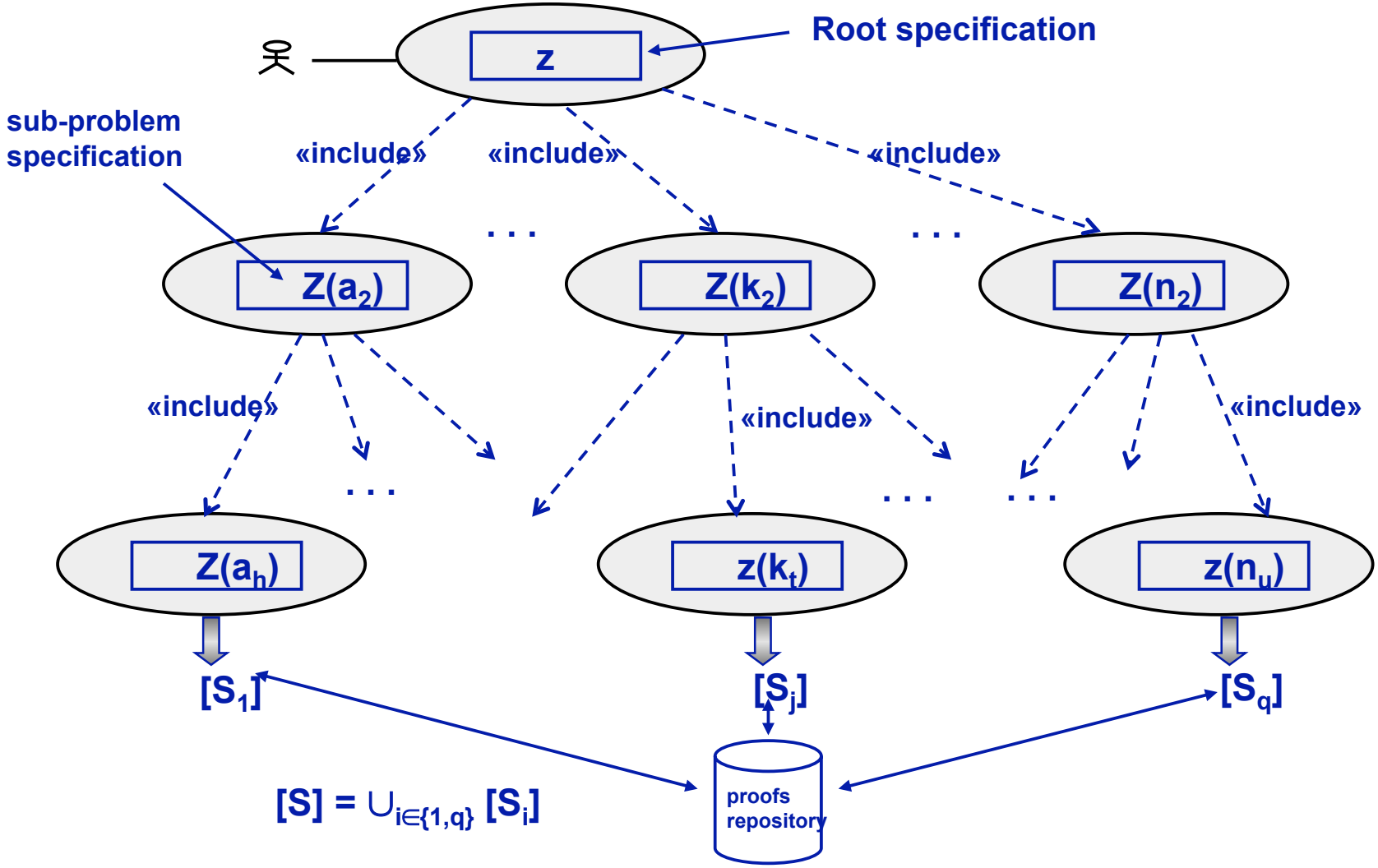
Languages and their abstraction levels



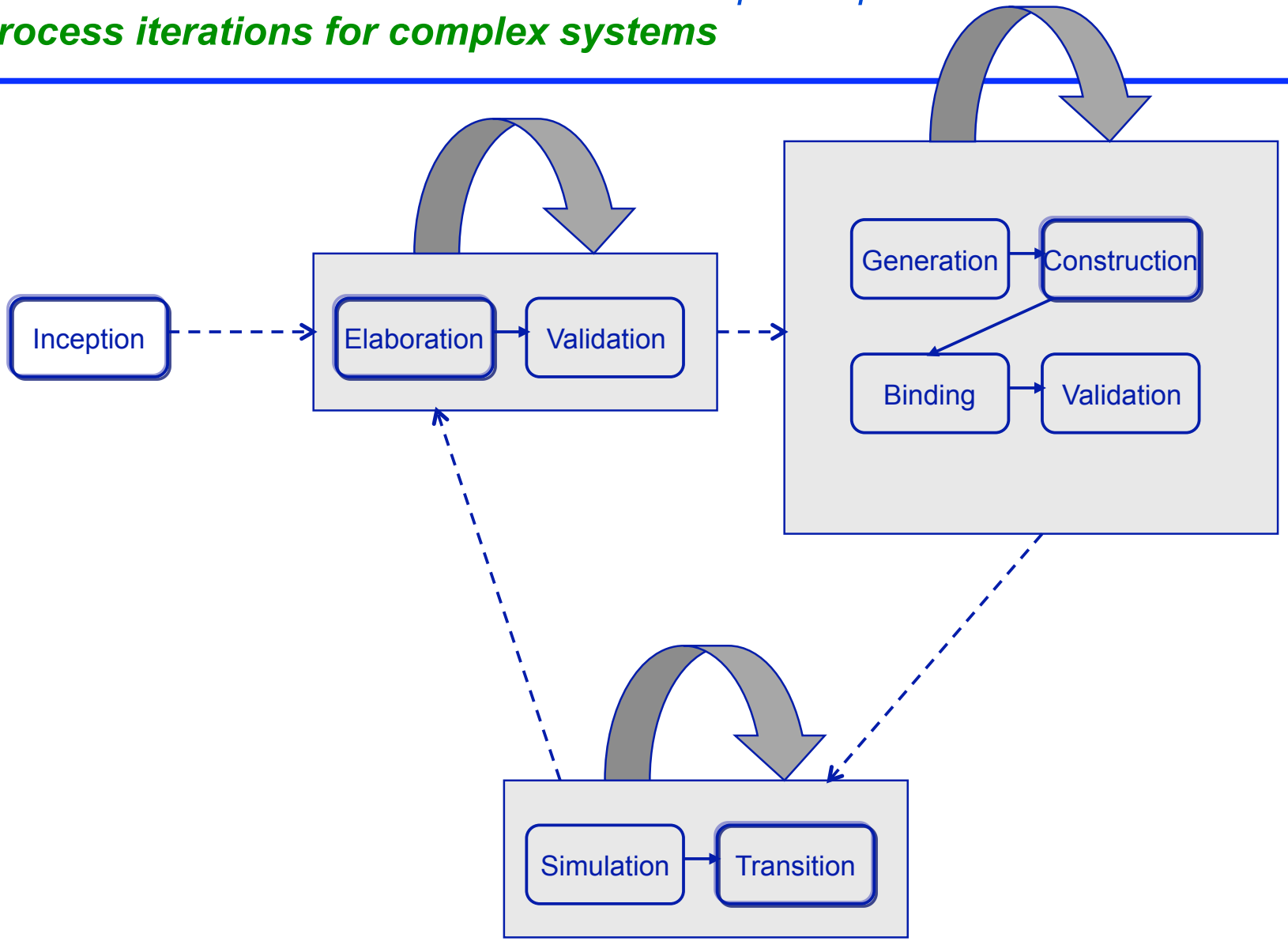
1/ a formal use-case driven method



2/ Proof-based use cases: a sub-objectives technique



Process iterations for complex systems



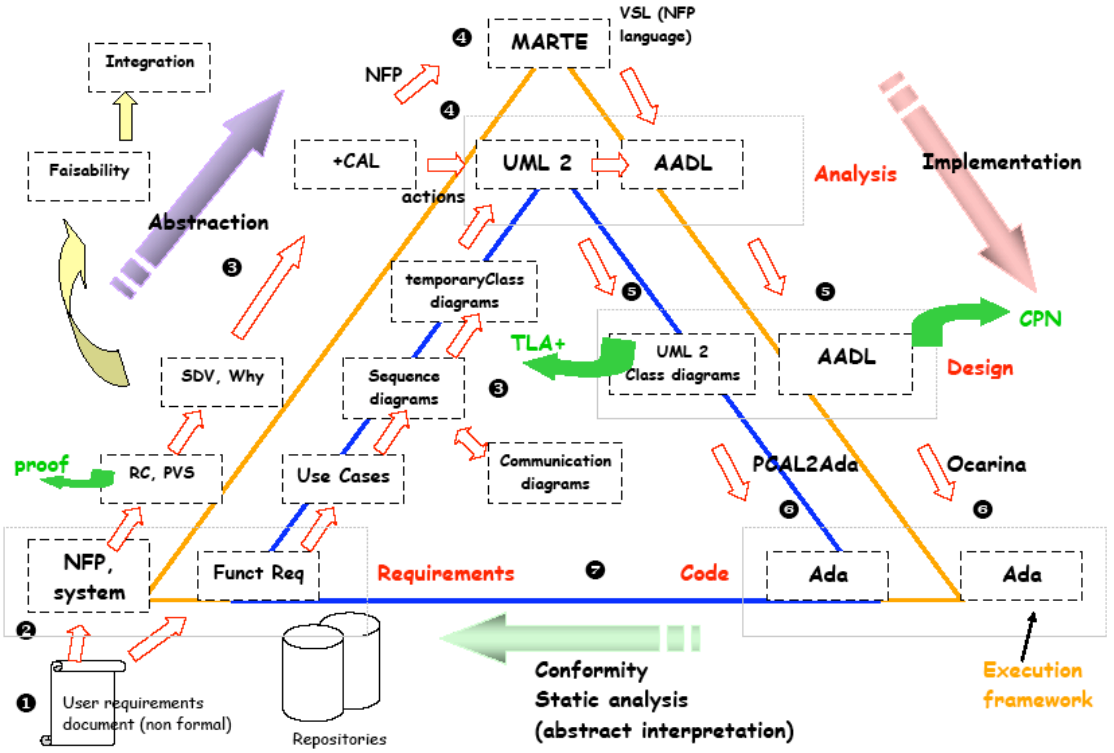
C-Method and its lifecycle guided by the abstraction levels

Inserm

Non Functional

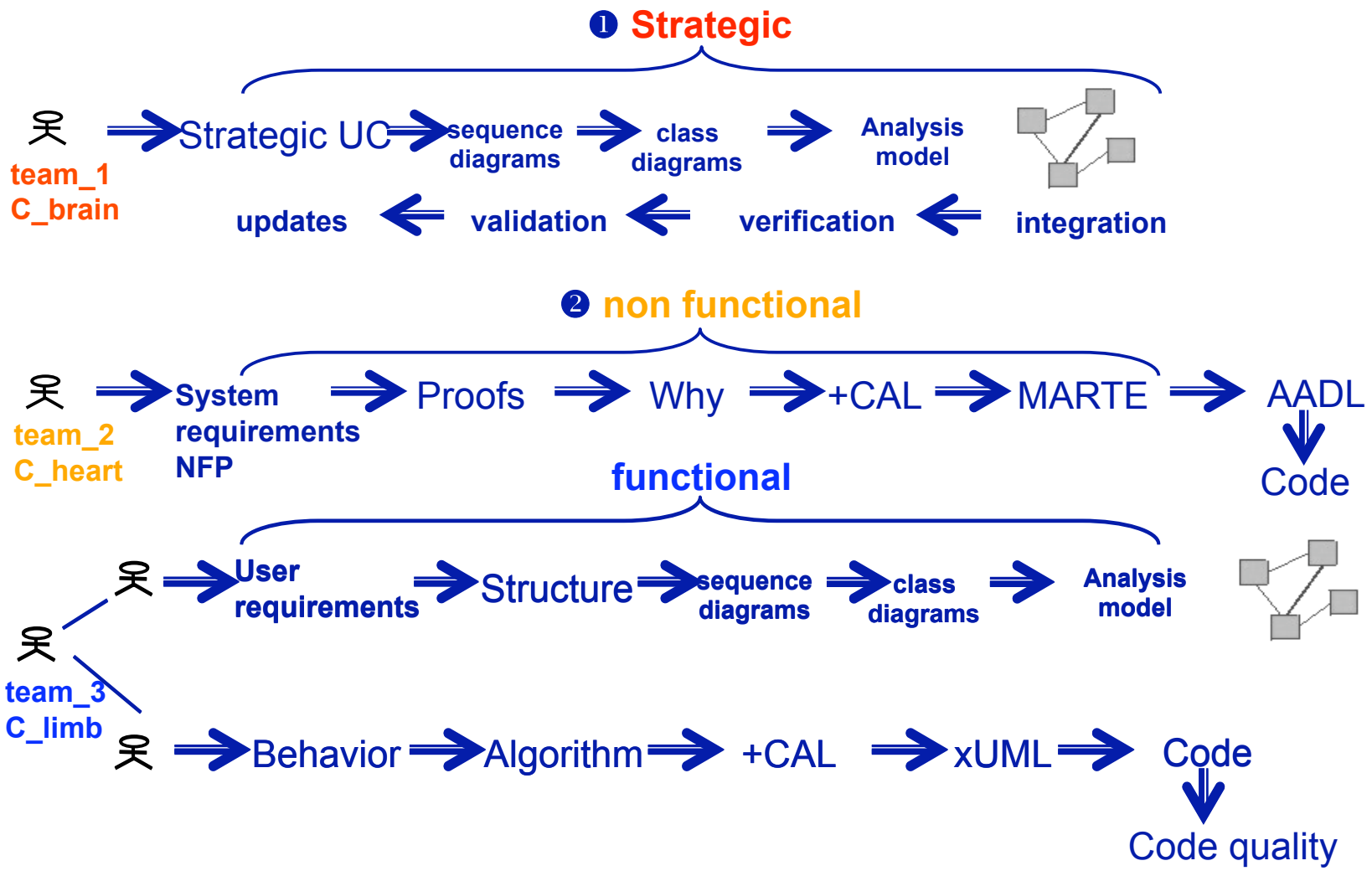
Functional

Proofs / Verification



- The strategy: multiple and // preparations of the other phases → seamless transitions
  - Activities are not sequential
  - Many strata in the requirement phases
  - 20% models generate 80% code
- 3 very large sub-processes to establish the guidelines
  - **C-brain** → software skeleton, global integration, verification
  - **C-heart** → architecture and execution framework
  - **C-limb** → functional part, final realization

The C-Method software development sub-processes



- b1- manage the whole software development process
- b2- classify the requirements (strategic, user, system)
- b3- formalize the strategic requirements
- b4- check the requirements to be conform to the non-formal requirements
- b5- drive the strategic use cases
- b6- control the Analysis process
- b7- drive the integration process
- b8- drive the verification process until the last conformance step

The c-heart activities

- h1- extract the non functional properties from the first requirement
- h2- formalize them
- h3- check they are conform to the non-formal requirements
- h4- translate them in an algorithm language
- h5- prove them
- h6- build the whole architecture analysis model (MARTE)
- h7- prepare the software binding
- h8- drive the MARTE2AADL transformation
- h9- trace the NFP during the model transformation into AADL
- h10- drive the scheduling analysis
- h11- optimize the architecture
- h12- bind the software on the hardware
- h13- drive the execution framework generation
- h14- drive the execution framework verification

The c-limb activities

- I1- drive the functional requirements
- I2- formalize them
- I3- check their compliance to the non-formal requirements
- I4- lead the structural analysis and design
  - I41- identify the classes
  - I42- **extract the complex behavioral parts for I5**
  - I43- integrate and complete the skeletton
  - I44- generate a preliminary code skeletton
- I5- lead the behavioral analysis and design
  - I51- formalize algorithms in +CAL
  - I52- translate them into TLA+ specifications and check them with relevant invariants
  - I53- integrate the +CAL algorithms in the actions
  - I54- **lead the +CAL2Ada code generation**
- I6- test the binding
- I7- validate the integration
- I8- lead the functional validation
- I9- verify the consistency with the requirements (abstract interpretation)
- I10- manage the upgrade process until the death of the product

Future & ongoing works

- A certified translator **PCAL2Ada (written in PVS)**
- Integration of +CAL in a professional modeling tool (as Rhapsody)
- Automate the abstraction phase of the C-Method (IA)
- Creation of user group / working group at the OMG

Conclusions

- New phases
- New activities
- Other iteration types
- New lifecycle, new method
- Intermediate languages
- Languages integration techniques
- Same overall logic: distribution of activities along the lifecycle