Formal Software Engineering
(génie logiciel avec l'approche formelle)

J. Christian Attiogbé

Master Alma, Septembre 2016

Two parts course, shared with Pr. Claude Jard and M. Benoît Delahaye

About me: Teaching at IUT & UFR Sciences - Research at LINA

Dpt Info IUT

- Modélisation de données, Base de données,
- Algorithme et Programmation
- Modélisation et programmation de systèmes répartis
- Programmation Objet, Sécurité des réseaux

Dpt Info UFR Sciences

- Formal Software Engineering (*Construction formelle de logiciels*)

LINA - UMR 6241 / Université de Nantes - CNRS - Mines de Nantes

- Topic ALD: AeLoS, Ascola, GDD, GRIMM
- Topic SAD: COD, COMBI, Contraintes/TASC, OPTI, TALN

AeLoS Team - LINA

Research Topics

- Construction of Correct Architecture and Software
- Modelling, Verification, Refinement, Semantics
- Distributed Systems (services, components, architectures, properties)

Contact the team members for various internship projects, PhD projects, ...

AeLoS Team Members

P. André, G. Ardourel, C. Attiogbé, B. Delahaye, C. Jard, A. Lanoix, J-M. Mottu, M. Oussalah, D. Tamzalit + PhD Students

Presentation of this Course (48h)

Formal modelling and verification of software
(the only way amenable to prove software correctness)

- Part 1 - by Christian Attiogbé (˜ 24 hours)
  Correct Construction with B Method, Event-B
  Atelier B/Rodin (theorem-proving).
  xs
- Part 2 - by Claude Jard & Benoît Delahaye(˜ 24 hours)
  Concurrency and Semantic Models ;
  Petri Nets/Romeo (model-checking), Timed Models/Uppaal (model-checking).

Presentation of this Course (48h)

Forecast Agenda

Dates	Part 1 C. Attiogbé	Part 2 C. Jard , B. Delahaye
07/9	Introduction FSE	
14/9	CA	
21/9	CA	
05/10	CA	
10/10	CA	
19/10	CA	
26/10	Claude Jard	
2/11	Congés Toussaint	
09/11		CJ
16/11		CJ
23/11		CJ
30/11		BD
07/12		BD

About you - Motivations for this course

MASTER level ⇒ Managing industrial projects (computer systems)
in various domains,
with variable size (small or big)
Complex CS projects ⇒ Methods, Techniques, Tools

- Analysis Methods,
- Design & Verification Methods,
- Development/Implementation Methods.

You probably already know some programming languages, semi-formal methods, [FM?]

Are you confortable with large CS projects, difficult problems, (what about the future)... ?

Categories/Natures of Software Systems

Nature	Features?	Methods?
sequential		
autonomous (transformational)		
centralised		
reactive		
real-time		
parallel		
parallel and concurrent		
distributed		
embedded		
communication protocols ...		

$\Rightarrow$ various types of software systems, various methods

Introduction: Industry [already] adopts FM!

Difficulties for industries: Market Pressure, High costs, $\cdots$
BUT, there are numerous success stories

- Proof of a compiler (Coq, Xavier Leroy, 2011)
- Design of a Real-Time Operating System (TLA+)
  E. Verhulst, R.T. Boute, J.M. Sampaio Faria, B.H.C Sputh, V. Mezhuyev, Formal Development of a Network-Centric RTOS, 2011
- Proof of IEEE 1395 Firewire Protocols (Spin, PVS, B, +++ ; 2004+)
- Proof of control systems (B, Siemens)
- Proof of circuit (STMicroelectronics)
- BOS barrier protecting the harbor of Rotterdam (Z, 2001)
- Proof of microcode and software (Intel)
- Proof of Communication Protocols (IO Automata, 1993+)

The complexity of current computer systems discourages empirical methods.

Introduction: Prove the correction of a software

Build correctly a software or
Proof the correction of a software S via its model.

- The *model* of the software : M
- The *properties* : P
- $M \models P$
  proof depending on the structure of the model
  ex: prove that P is true in all the (reachable) states of M
  (if M is a state model)

Anyway, you need a formal model; and rigorous software construction methods. Do you know some?

☞ Learn how to build M, P and how to prove
(using dedicated tools or not).

Examples of models (you already know)

- Logic models
- Axiomatic/algebraic models (equation systems)
- State-based models (automata, lts, graphs)
- + various aspects: time, data, signals

Various classes of models

- Synchronous models
- Asynchronous models
- ...

☞ We will learn some aspects in this course.

Some references

- Jan Van Leeuwen, *Handbooks of Formal Models and Semantics*, 1990
- J. Wing, *A Case Study in Model Checking Software Systems*, SCP, 1997
- Mana & Pnueli; de Roever et Al.;
- E. Clarke, J. Wing, *Formal Methods: State of the Art and Future Directions*, CMU, 2006
- L. Lamport, numerous documents!
- André Arnod, *sémantique des processus communicants*
- J-F. Monin, *Introduction aux méthodes formelles*. Hermès, 2000
- *Success Stories*
  `www.fm4industry.org/index.php/DEPLOY_Success_Stories`,
  `www.fm4industry.org/index.php/Deploying_Event-B_in_an_Industrial_Mic`
- and Dijkstra, Hoare,...

Wing, Hehner, Monin, Holloway, ···

Introduction

- Computer Systems (*Systèmes informatiques*)?
- Construction?
- What issues?
- What concepts, theories, methods, techniques, tools?
- State of the art?
- The needs?

You can try: direct programming + tests, semi-formal methods, formal methods, ???

Are there some hints, well-established systematic approaches ?

Variety of computer systems

Nowadays, computers and softwares are everywhere

- Domestic devices, home automation, leisures
- Medical domains
- Transportation (airplanes, trains, cars, ...)
- Administration (integrated databases + decision systems)
- Various services (booking, health control, home control,)
- Bank and Financial systems
- Politics, electronic votes, analysis
- etc

What about the quality of software with respect to its critical place?
Do we know how to build it well?
Do we know how to maintain it well?
With What methods, techniques, tools?

Example: Web services interoperability (WS-AT)

Interoperability of services in distributed aplications.
In distributed applications several services cooperate to achieve common goals.
Pbm: How to build such interoperable, distributed applications with coordinated joint works? in an asynchronous context.
Web services tie together a large number of participants (they are services) forming large distributed computational units called activities. These activities are complex due to many parameters: interaction between participants, they can take long time...
To face the complexity, a framework to coordinate the activities is needed (it is the objective of WSCOOR, oasis). It enables participants to reach a consistent agreement on the outcome of distributed activities.
Several protocols have been proposed as basis for the interaction between Web services.
For example WS-Atomic Transaction (WS AT) contains protocols which are mechanisms to create activities, join into them, and reach common agreement on the outcome of joint operations.

Example: Web services interoperability (WS-AT)

Def: An activity is a set of actions spaning multiple services but with a common goal (classical ex: resa).
The activities that require the ACID (atomic, consistent, isolated, and durable) properties of transactions are users of WS-Atomic Transaction.
An initiator creates/initiates an activity, and communicates its context to other applications. The other applications can register to participate in the activity. A coordinator manages all the participants of an activity. The coordinator at some point can decide to abort or to try to commit the transaction. Therefore it initiates (preparation phase) a vote to which all the participants participate. When there is a common positive agreement, it can commit the outcome (commit phase) of the trasaction (all or nothing).
Required Safety Property: to guarantee that the initiator and the participants agree on whether the transaction is committed or aborted.

http://docs.oasis-open.org/ws-tx/wstx-wsat-1.1-spec-os/wstx-wsat-1.1-spec-os.html

Example of formal models or specifications

```
MACHINE /* Sorting: a set of naturals -> seq. of natural */
  Tri
CONSTANTS
  tride /* defining a function */
PROPERTIES
  tride : FIN(NAT) ---> seq(NAT) &
   (ran(tride(ss)) = ss &
     %(ii,jj).(ii : dom(tride(ss)) &
    jj : dom(tride(ss)) &
     ii < jj =>
       (tride(ss))(ii) < (tride(ss))(jj) ) ) )
END
```

Emphasize abstraction = what (not how)

Example of formal specifications

```
system ProdCons /* Model */
sets
   DATA ;
   STATE = {empty, full}
variables
   buffer, bufferstate, bufferc
invariant
   bufferstate ∈ STATE
∧ buffer ∈ DATA ∧ bufferc ∈ DATA
initialization
   bufferstate := empty
||  buffer :∈ DATA
||  bufferc :∈ DATA
end
```

Emphasize abstraction

Example of formal specifications

ProdCons (continued)...

```
events
   produce ≙ /* when buffer empty */
      any dd where
         dd ∈ DATA ∧ bufferstate = empty
      then
         buffer := dd ||
         bufferstate := full
      end ;
   consume ≙ /* when buffer is full */
      select   bufferstate = full
      then
           bufferc := buffer ||
           bufferstate := empty
      end
end
```

Emphasize abstraction = what (not how)

Example of properties

Always an unique process in CS	$card(activeProc) = 1$
A process cannot be simultaneously active and blocked	$activeProc \cap clockedProc = \emptyset$
...	...

☞ The use of invariant properties

- Safety properties: *nothing bad should happen*
- Liveness properties: *something good eventually happens*
  More generally, one uses Modal Logics.

Semi-Formal Methods

Examples of semi-formal methods

- Functional Analysis (SA..., SADT),
- Structured Analysis (SA, SSADM), SA-RT,
- Entity-Relationship (*Entités/Associations*): Merise, Axiale,
- JSD/JSP,
- Object-Oriented Analysis, OMT, UML,
- Software Architecture (System Level ; Top-Down approach),
- etc

Pros and Cons

Need of Formal Methods

Need of rigorous methods for some specific domains:

- Security, Certification, Cost, Maintenance
- ITSEC (Information Technology Security Evaluation Criteria) requires the use of **formal methods**
- Failure of (one flight) of ARIANE!, failure of a Pentium series, etc
- Environments which are dangerous for humans (nuclear, chemistry, marine, etc)
- Embedded Systems (vehicles, home equipments, etc)
- Automata (medical domain, etc)
- etc

Pros and cons

Examples

Examples of formal methods

- Logics (First Order, Higher Order, Modal)
- Finite State Machines (Mealy, Moore, ...)
- Transition systems (Automata, Petri nets, Communicating processes,...)
- Algebraic Methods
- Timed Systems (extension of Tansition systems)
- ...

Features of Formal Methods

Formal methods ⇒ use rigorous approach to

- guaranty of software correction with respect to specifications,
- decrease/remove errors, and disfunctionning,
- make it easy the maintenance and the evolution.

Introduction to Formal Methods

Construction methods of computer systems
A few analogies:
Building Engineering (*Génie civil*)
→ Architecture, schemes/blueprints (design), computings, construction (implementation)
Physics
→ Observations, modelling/study of models, implementation
Computer Science (Informatique)
⇒

Requirement Analysis (observations?)

Modelling - study of models,

Design and Implementation of systems.

Preliminaries

Various **approaches of formal methods**:

- à postériori : First, one implements (programming paradigm) and then one verifies that the produced program is correct
  → proof systems, testing, model-checking
- à priori : One builds correctly the system
  → Development methods (refinement, synthesis), proof systems

Several formal methods (languages, proof systems, methods)

Building Computer Systems | Formal Methods: Introduction

Preliminaries

- Top-down approach: by decomposition
  - Global analysis (system study, system engineering)
  - Software Architecture
    ↓
  - Implementation of components
    - Direct Programming or
    - Formal Development
- Bottom-up approach: composition of elementary components.
  - Study of available components,
  - Composition, reuse.

J. Christian Attiogbé (Université de Nantes) Formal Software Engineering (génie logiciel av Master Alma, Septembre 2016 Two parts course, shared / 61

Need of formal methods

In all cases (approaches), make use of formal methods for

- Study of systems
- Study and construction of components
- Formal framework for reasoning, analysis, development.

What are inside formal methods?

- Logics
- Algebra
- Discrete Mathematics
- Set Theory
- Automata Theory
- Type Theory
- Refinement Theory
- ...

Examples of a few industrial applications

with Z, VDM, CSP

- IBM, INMOS, ...
- CICS: IBM interactive transactional system (1983, Z)
- Integrated Circuit Design, Transputer (Z, CSP)
- etc

Examples of a few industrial applications

with the B Method (J-R. Abrial)
GEC ALSTHOM, SNCF and MATRA Transport (now Siemens)

- Railway Speed Control System (KVS for SNCF)
- Line A of the Paris RER - SACEM (signaling, speed control)
- Calcutta Metro (CTDC)
- Montreal Metro(CTDC), Marseille, Bel horizonte
- Météor (line 14, of Paris Metro, without human driver)
- Landing doors (*portes pallières*) in Metro stations
- Old people insurance, in French Sécurité Sociale
- CICS of IBM (major restructuring of a transactional, about 800000 lines of code)
- B and VDM are used in financial domain softwares, BULL UK

Many other systems with Petri Nets, Lotos, etc

Example of the Railway Speed Control System (Metro)

- Data acquisition (sensors, converters, etc),
- Computation/decision,
- Orders sent to physical devices (speed slowing system, braking system),
- Embedding of the software in the global system of the train.

Other used approaches

Some of them are equipped with tools and adopted by industries

RAISE (Result of an european project - ESPRIT)
algebric approach + *communicating processes*

LOTOS, SDL (Standard européen)
algebric approach + *communicating processes*

PVS (USA)

MEC, AltaRica (Université de Bordeaux + industries)

Classical Logics: First order Logic, Hoare Logic, etc
(Why, Frama-C, Krakatoa (Java), Key,...)

Non-classical Logics, modal logics

Coq, High-Order Logics, type theory

Foundation of formal approaches (proof)

Interpretation of the Curry-Howard's Isomorphism:

$$\text{Proof} \quad \frac{Axioms}{Theorems} \qquad \text{equiv. to} \qquad \text{Development} \quad \frac{Specifications}{Programs}$$

♡

Approach of Formal Construction

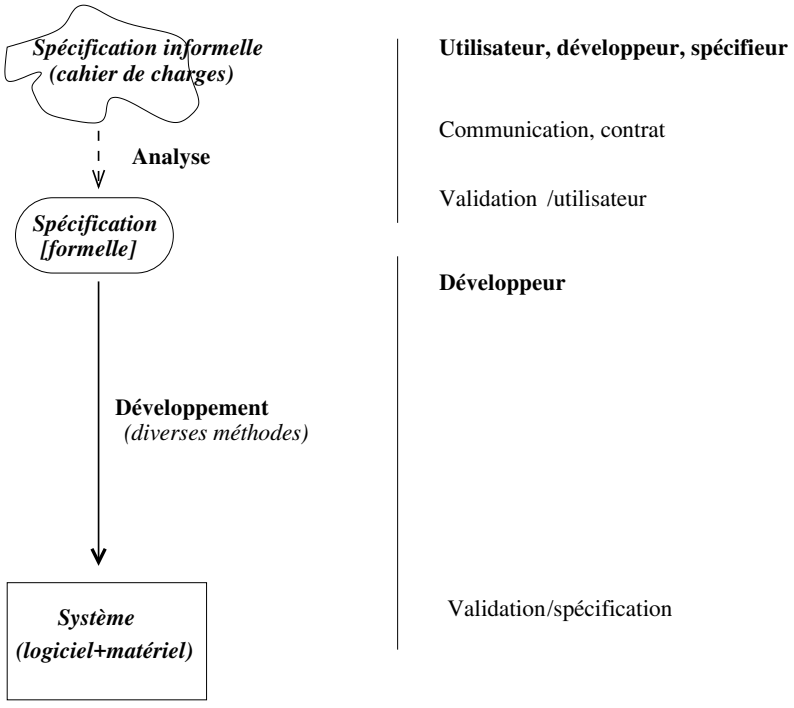

Figure: Issue of system development de system

Overview of Software Construction

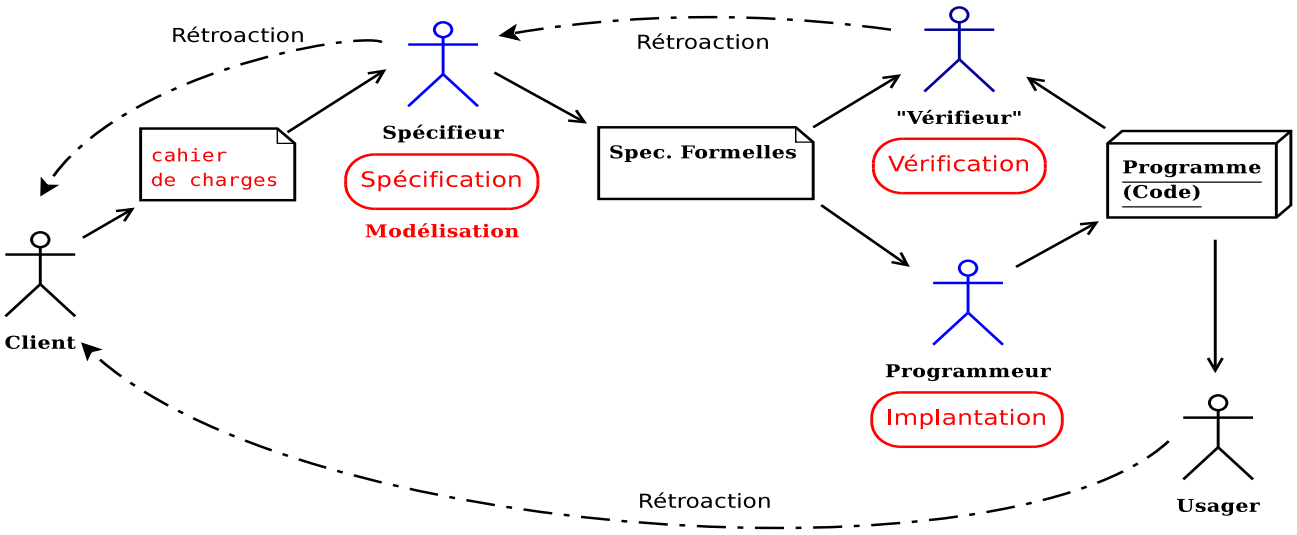

Figure: A life cycle of formal software construction

Use of Formal Methods

→ Not always approriate:

- A hammer to kill a fly
  - depending on the needs
- Professional environment
  - Available experiments?
- Industrial context
  - Delay, costs, productivity
- Certification
  - Requirements of clients.

Which approach to use?

→ Several parameters:

- Designer/implementor of big systems,
- Designer/implementor of small (home) systems,
- Features of systems to be implemented,
- Available experiments,
- ...

Categories/Natures of Software Systems

Nature	Features?	Methods?
sequential		
autonomous (transformational)		
centralised		
reactive		
real-time		
parallel		
parallel and concurrent		
distributed		
embedded		
communication protocols ...		

⇒ various types of software systems, various methods

A few difficult points

- To describe precisely the intended system specification
- To build correctly the sofware development
- To be sure that the constructed software is correct with respect to the needs
- Maintenance/Evolution of the system.

Example of the B Method

The built system is correct by construction : stepwise refinement from abstract model

- Initially (< 1996), sequential execution semantics,
  for sequential systems; no behavioural indeterminism
- Then, extended (Event-B, 1998)
  for autonomous and reactive systems, concurrent and distributed systems.

Example of LOTOS

Analysis and verification of systems (already developped or not)

- Sequential and concurrent execution models
  Concurrent and sequential systems
- Possibility of behavioural indeterminism
- Autonomous and reactive systems
- Centralised and distributed systems.

Steps of the software life cycle

From requirement document → computer system
requirement document = informal specification
Several steps:

- Analysis
- Specification, Modelling
- Design
- Implementation
- Maintenance

Practice

Each project is unique

- Nature of complex systems → multifacets, modular
- Several methods, including:
  - Semi-formal methods
  - Formal Methods (integrated) → to deal with complex systems.

⇒ mastering several methods

A few definitions

Modelling:
Hoare: A scientific theory is formalised as a mathematical model of reality, from which can be deduced or calculated the observable properties and of a well-defined class of processes in the physical world.
There are two main notions of models in computer science.

1. Model = an approximation of the reality by a mathematical structure.
   An object O is a model of a reality R, if O allows one to answer all the questions about R.

In Mathematics, Physics, ... models are built with equation systems using quantities (masses, energy, ...) or hypothetic laws.

A few definitions (continued)

2. Logics, theory of models
   A model of a theory T is a structure in which the axioms of T are valid.
   A *structure* S is a model of a theory T, or S satisfies T if all formula of T is satisfied in S.
   The reality is a model of a theory!
   First Order Theory = any set of logic formula in a given language (precisely defined).
   Model as an interpretation of a specification - an algebra as a model of an algebraic specification (or an axiomatisation).

Practice A few definitions

A few definitions (continued)

These two notions of *model* are encountered in the model-oriented (or state-oriented) and property-oriented approaches of Soft. Eng.
In current use,

- *model* = (archetype), what serves or is used for imitation to reproduce orther instances.
- *model* = (paradigm), declination model, conjugation model, etc
- *model* = (reference), ...

J. Christian Attiogbé (Université de Nantes) Formal Software Engineering (génie logiciel av Master Alma, Septembre 2016 Two parts course, shared / 61

Examples of theory

Set theory: it is based on a set of axioms (Bourbaki, Cantor, Zermelo, ...).
The objects of this theory are called sets.
The classe of the sets is called the universe.
The axioms of the set theory (of Zermelo+Fraenkel) are the following:

Examples of theory (continued)

- Axiom of the empty set: *there exists a set which does not contain any element: it is the empty set.*
- Extensionality Axiom: *two sets are equal if and only if they contain exactly the same elements.*
- Union Axiom: *the union of sets is a set.*
- Axiom of the set of parts: *given a set E, there exists a set P suchat that a set F is member of P if F is a part of E.*
- Axiom of replacing/substitution schema(Fraenkel, 1922) : *When one defines a function with the formula of the set theory, the elements for which this function verifies a given property are also a set.*

Moreover, to these axioms is added, the axiom of infinite: *there exists an infinte ordinal.*
ZFC = ZF + axiom of choice

- Axiom of choice: *Given a family of disjoint sets, if we consider one element of each set of the family, then one builds another set.*

A few definitions (continued)

Semiformal Method =

- Graphical Language [+ formal]
  (precise syntax and unprecise semantics) and
- Various analysis tools.

→ Combination of languages/methods/techniques that do not all have a precise semantics.
Examples : JSD, OMT, OOX, UML

Interest and Limitations of Semi-formal Methods

- SADT, SA-RT, SSADM, ...
- JSD-JSP,
- Merise, Axial, ...
- OOA, OMT, UML
- ...

The problem analysis is performed.
It is a positive contribution, although insufficient.
The problem is sided.
→ impossible to reason formally on the intended system.
→ there can be ambiguities and errors.

A few definitions (continued)

Formal Method =

- Formal Language (precise syntax and precise semantics) and
- Proof or formal reasoning system.

Examples: FSM, RdP, Z, CCS, CSP, HOL, Coq, PVS, B, ISabelle

Formal Development =

- systematic transformation of specifications into programs using predefined laws/rules. **Examples: Synthesis, Refinement**

Examples: B Method, Perfect, Escher C, Coq

A few definitions (continued)

Verification: to show that a system (S) is correct with respect to some properties (P)

$$S \models P$$

Validation: to show that a system (S) is correct with respect to some informal properties (the needs)

$$S \sim S_{informal}$$

Formal reasoning : Consists in **applying a formal system to a specification**.

Examples of formal reasoning

- Refinement of specification,
- Verification of the properties of a system,
- Validation via verification,
- Proof of theorems (*theorem proving*),
- Analysis of a system (represented by a state machine) wrt some properties (*model checking*).

$\Longrightarrow$ Logic is the foundation of the formal approaches

Software Life Cycles

- Life cycles have been for longtime, the methodological support of software development.
- the most representative life cycles are:
  - the V software life cycle,
  - the Cascade life cycle or 'Iterative Waterfall' life cycle,
  - Balzer's cycle, 1980+
  - Spiral life cycle by Barry Boehm, 1986

Software Life Cycle

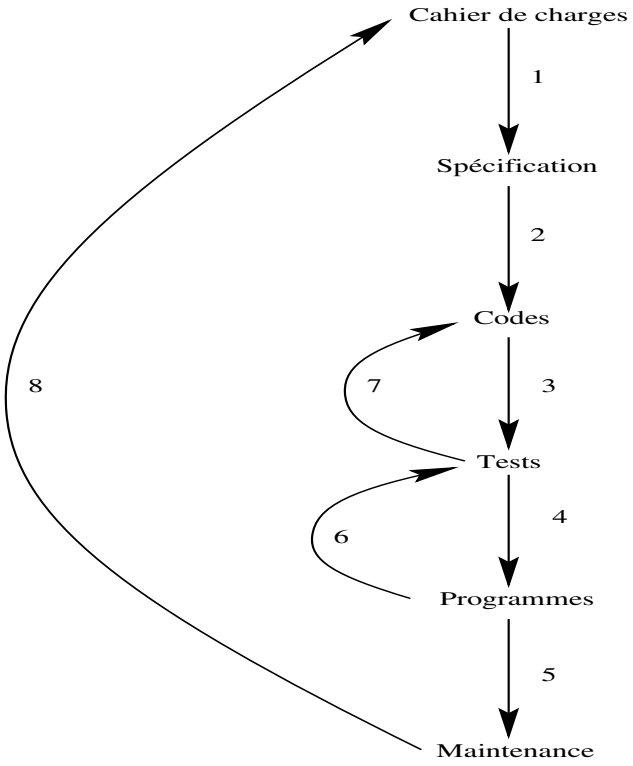

Figure: Cascade life cycles (by Boehm 1976)

Formal Specification

⇒ Description in a formal language of the 'What is' a system to be developped.

- Result of the analysis step.
- Several possible forms depending on the nature of the system.

One considers languages or formalisms of formal specification:
Logics, Z, Algebraic Specification Languages, process algebra, etc

Specification Approaches

Data or operations aspects first?

- **The data of a system permit to describe the state of the system**
- **The operations of a system enable to describe its behaviour with axioms.**

There are the data paradigm and the operation paradigm.
One should distinguish the operations expressing the behaviour of a system from the operations that characterise the data of the system.
The former work on the data of the system whereas the later are used to build or or manipulate the system data.

Specification Approaches

There are a third transversal paradigm.
It is the process paradigm (with the process algebra).
With this paradigm, the processes of a system are described with rules or equations that express the system behaviour or its states.
The main process algebra used to build others are:

- CSP (Hoare)
- CCS (Milner)
- ACP (Bergstra)

They are all based on the Transition System Approach.

Specification Approaches

You have to choose the one which is appropriate to a given problem!
Learn the main ones! and be able to move to others.
The remaining of the course has this objective.

Examples - Case studies

Preliminary examples and case studies

- Requirement analysis
- Abstraction
- Modelling
- ...

A few references

- Jan Van Leeuwen, *Handbooks of Formal Models and Semantics*, 1990
- J. Wing, *A Case Study in Model Checking Software Systems*, SCP, 1997
- Mana & Pnueli; de Roever et Al.;
- E. Clarke, J. Wing, *Formal Methods: State of the Art and Future Directions*, CMU, 2006
- L. Lamport, numerous documents!
- André Arnod, *sémantique des processus communicants*
- J-F. Monin, *Introduction aux méthodes formelles*. Hermès, 2000
- *Success Stories*
  `www.fm4industry.org/index.php/DEPLOY_Success_Stories`,
  `www.fm4industry.org/index.php/Deploying_Event-B_in_an_Industria`
- and Dijkstra, Hoare,...