

Life of a software

Activities during the construction of a software

- **Requirement analysis**
- **Domain analysis**
- **Design**
- **Implementation**
- **Verification: testing** (or formal methods)

Problem of communication



Ce que demande
l'utilisateur



Ce que l'analyste a
spécifié



Ce que prévoit le
concepteur



Ce que le programmeur a
écrit



Ce que la mise au point a
fait



Ce que l'utilisateur n'a pas
su exprimer

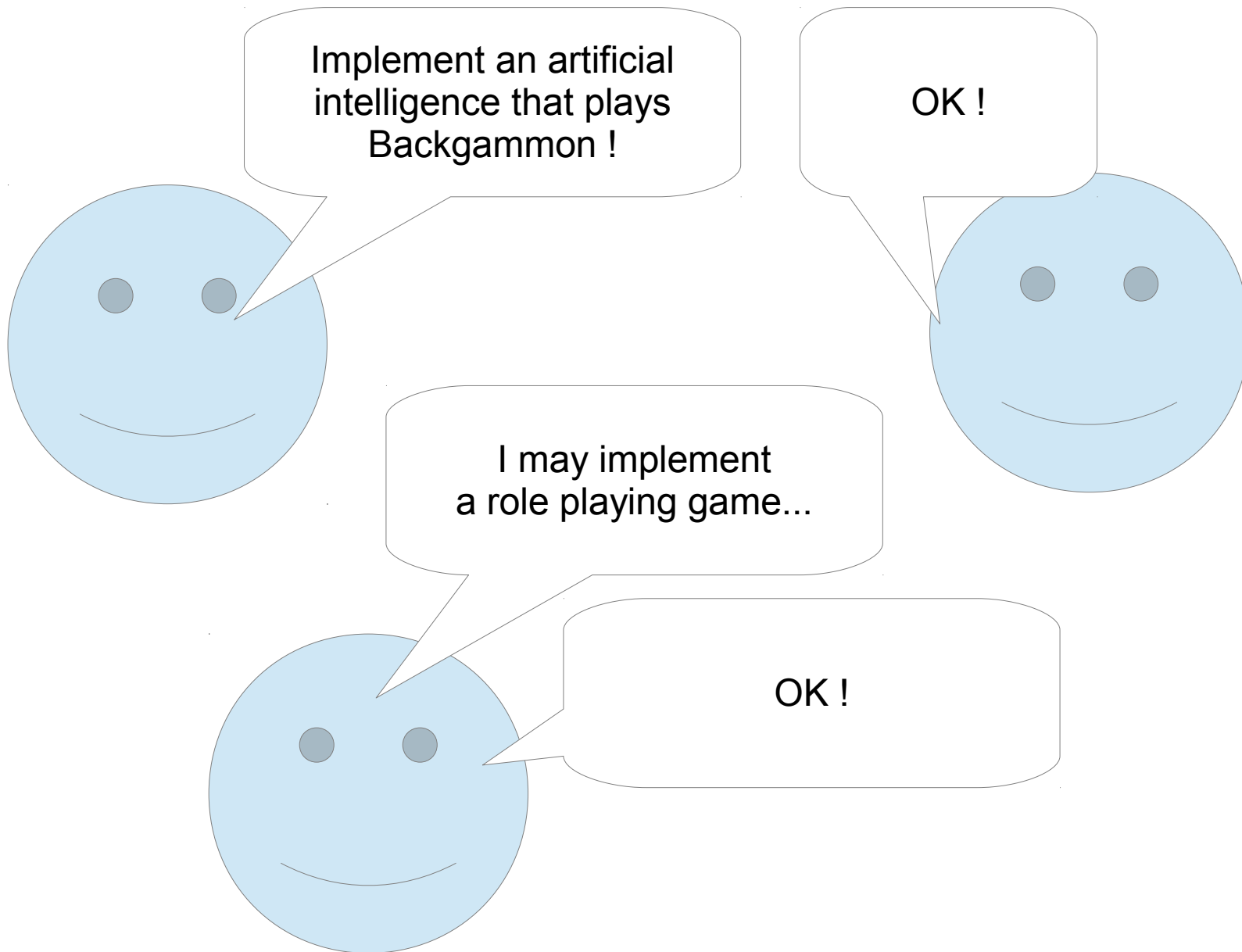
Requirement analysis



Requirement analysis

- **Non-functional** requirements
- **Functional** requirements

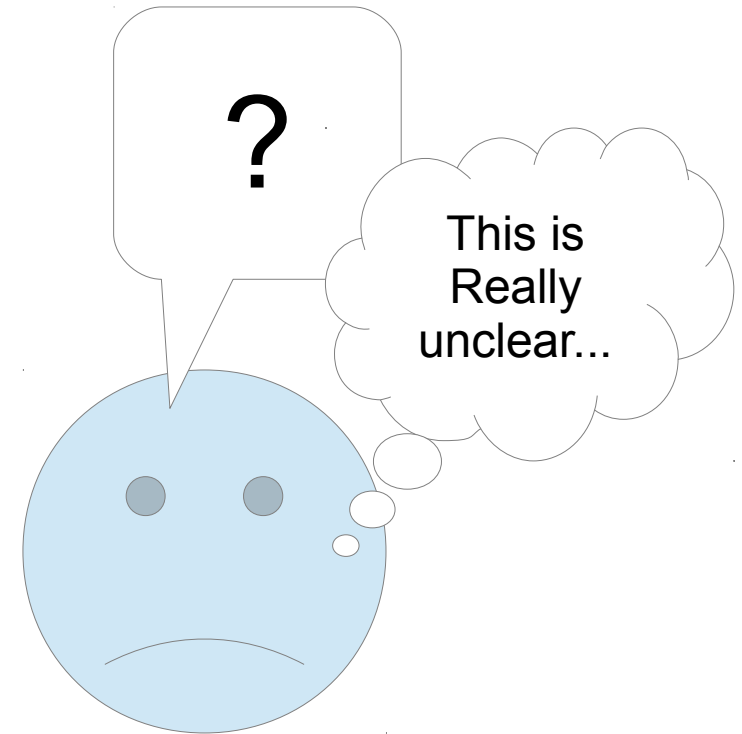
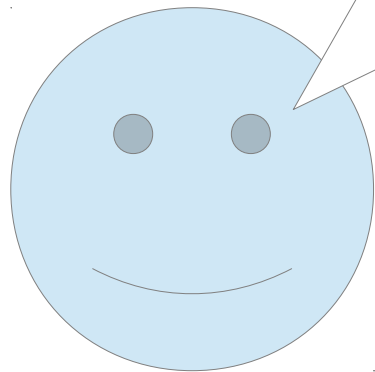
This may happen...



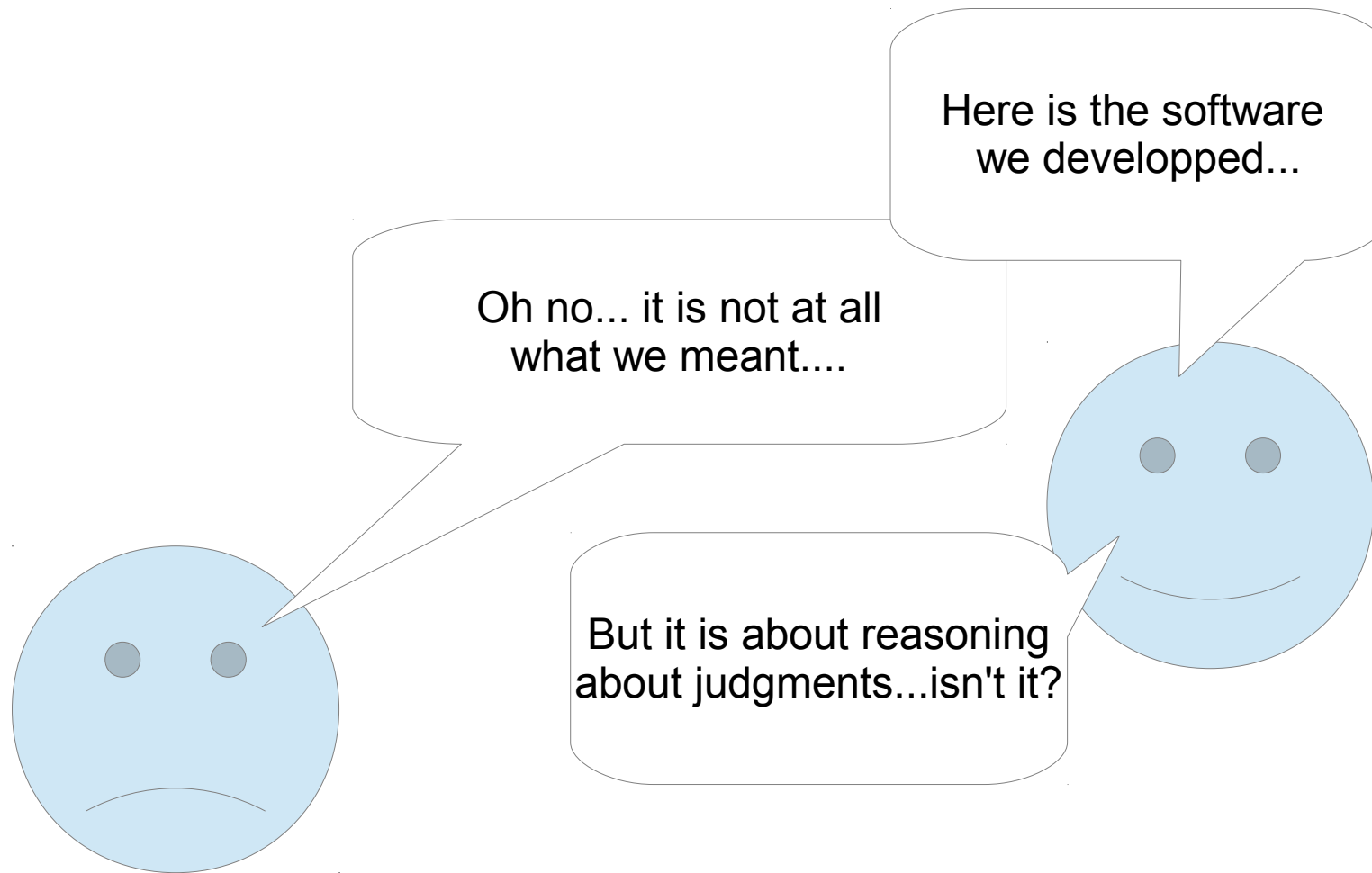
But sometimes the requirements are difficult to understand!

We need a software to help us to write down judgments. The software should reason about facts and laws.

We should be able to write down also some documents that explains how the automated reasoning should occur. Simply it should be very near from how a judge actually reasons...



The risk... after one year...



The risk...



Ce que demande
l'utilisateur



Ce que l'analyste a
spécifié



Ce que prévoit le
concepteur



Ce que le programmeur a
écrit



Ce que la mise au point a
fait



Ce que l'utilisateur n'a pas
su exprimer

Functional requirements

We identify:

- **use cases**
- and **actors** (user, outside computer program, etc.)

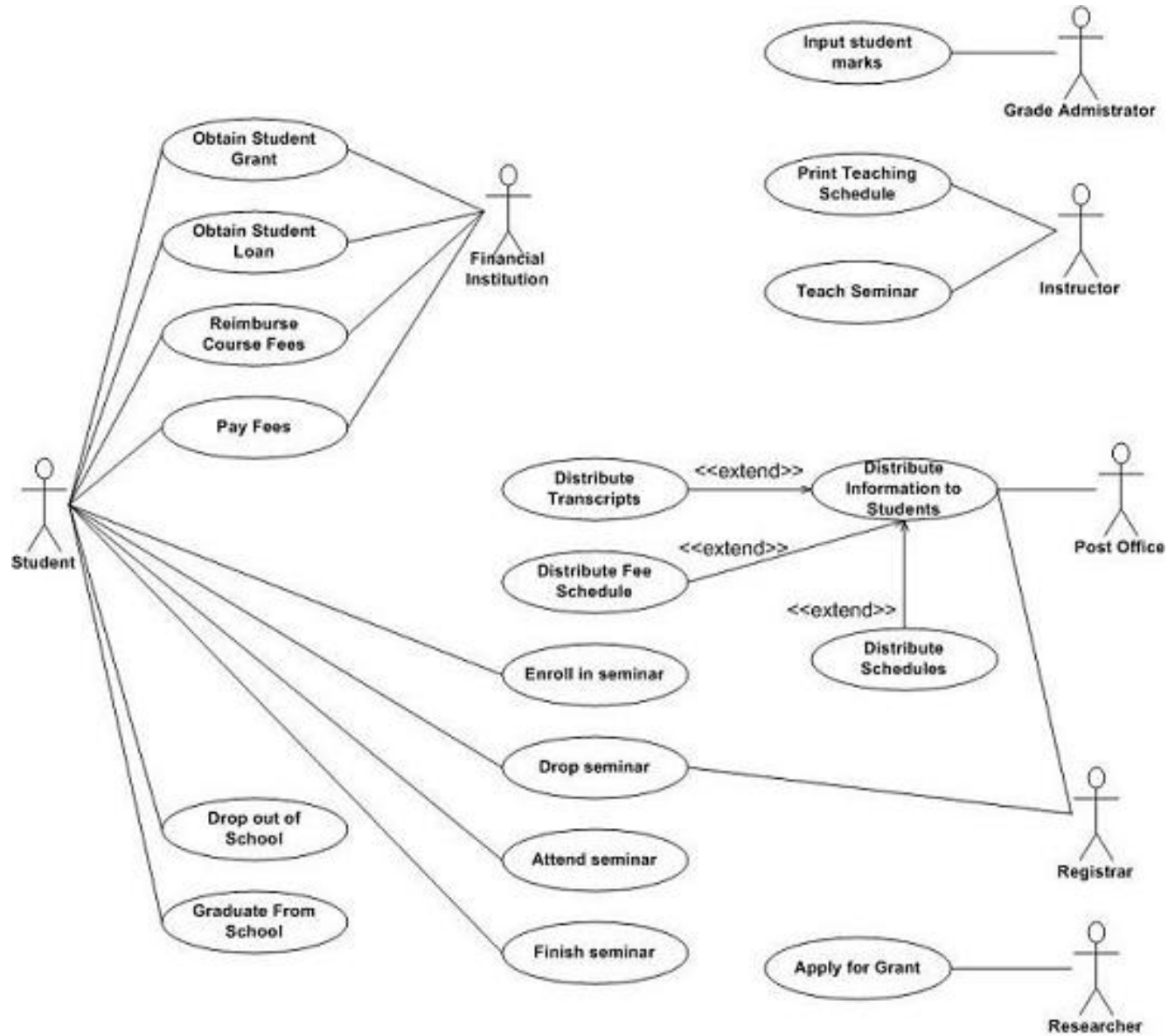
Aim:

- Understand the requirements clearly
- Role hierarchy
- Permissions
- Authentication

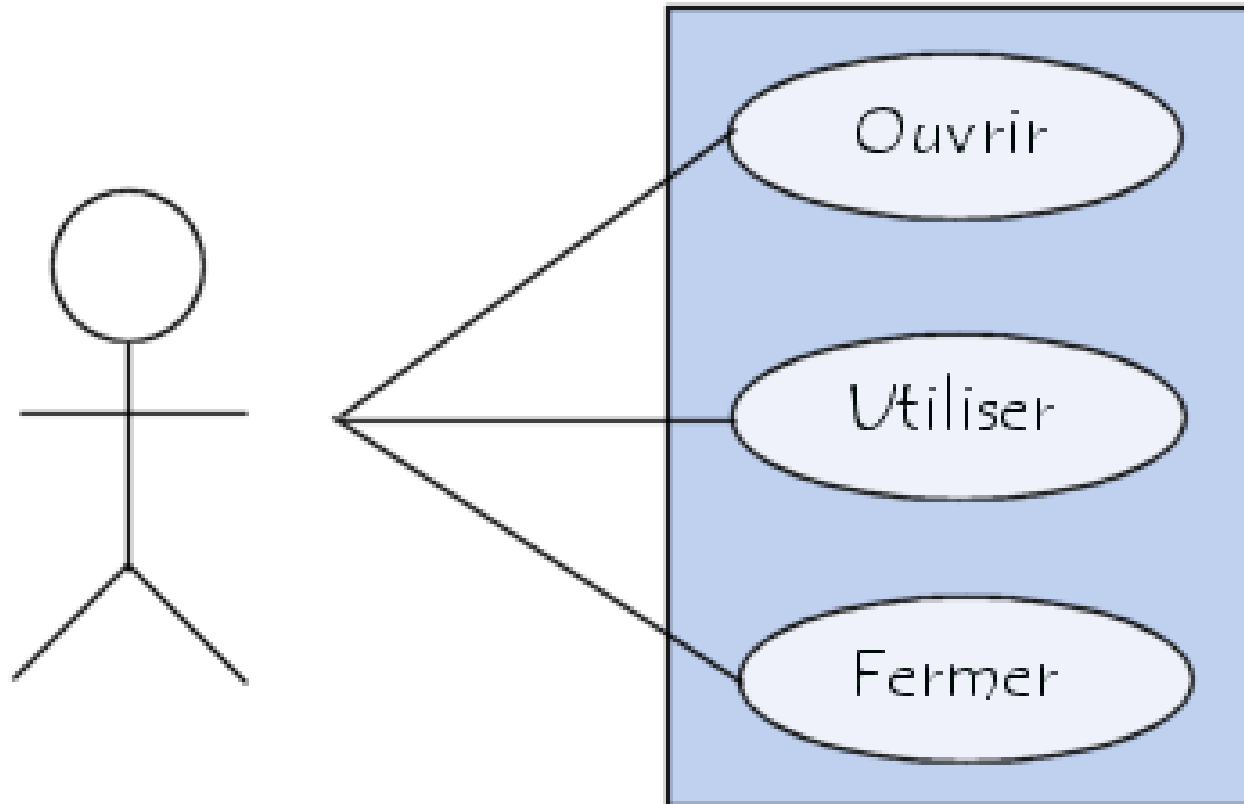
No overview over role hierarchy etc. with textual documents

- A student can obtain a grant, loan from the financial institution.
- The financial institution can ask a student to pay fees.
- The financial institution may reimburse course fees.
- A student can be enrolled into a seminar.
- Etc.

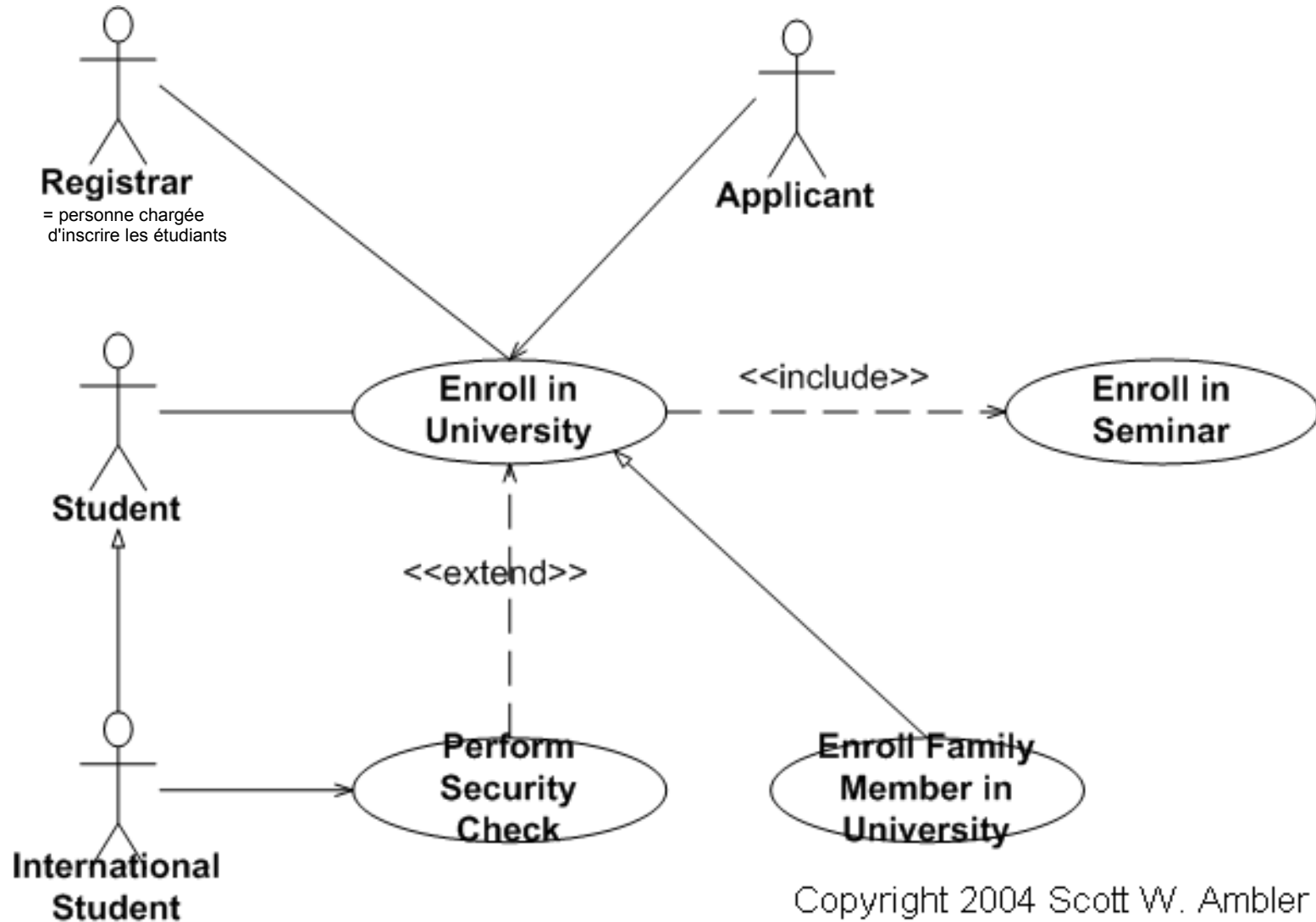
Solution: UML use case diagram



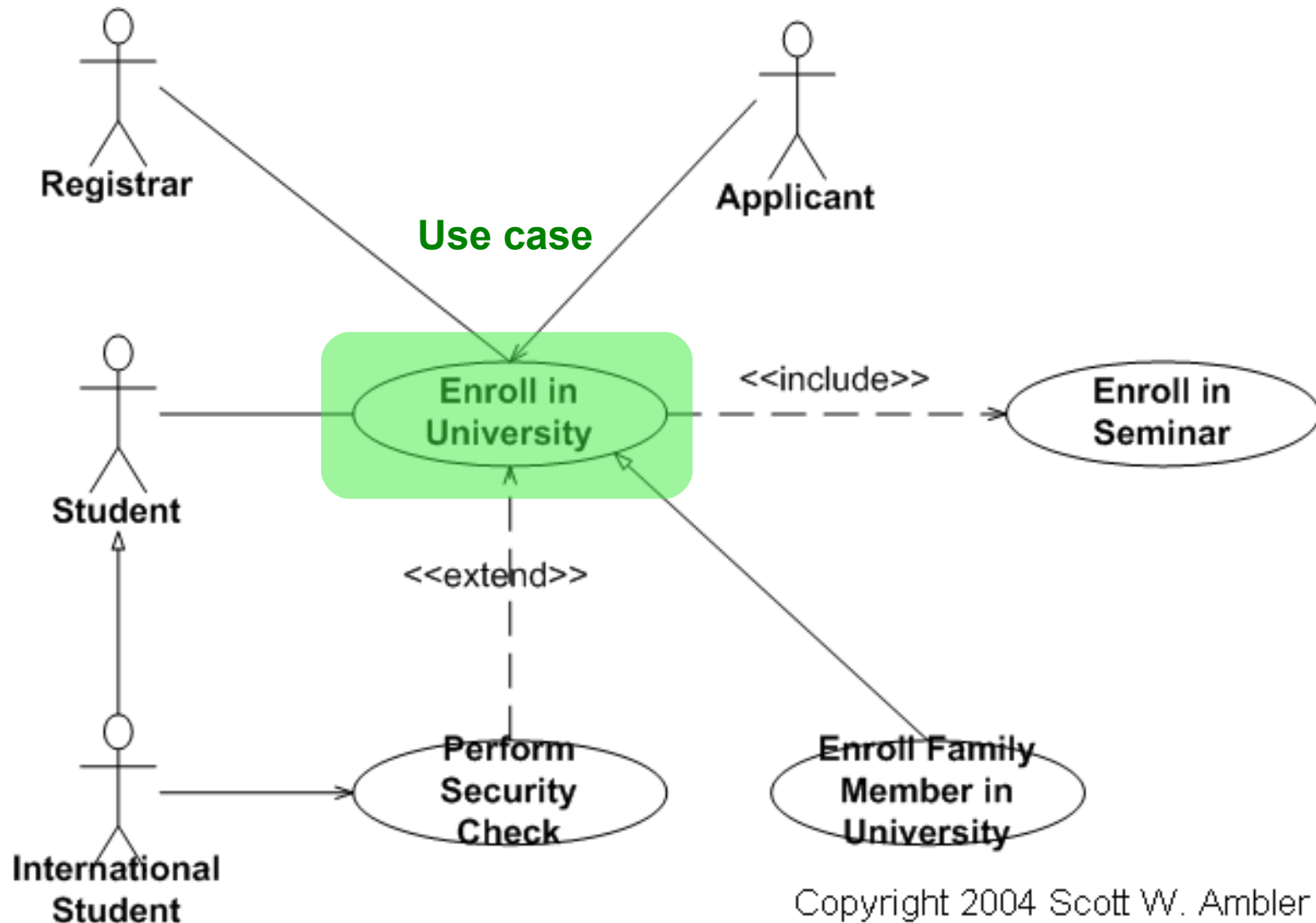
Simple UML use cases diagram (and when it is stupid to draw a diagram)



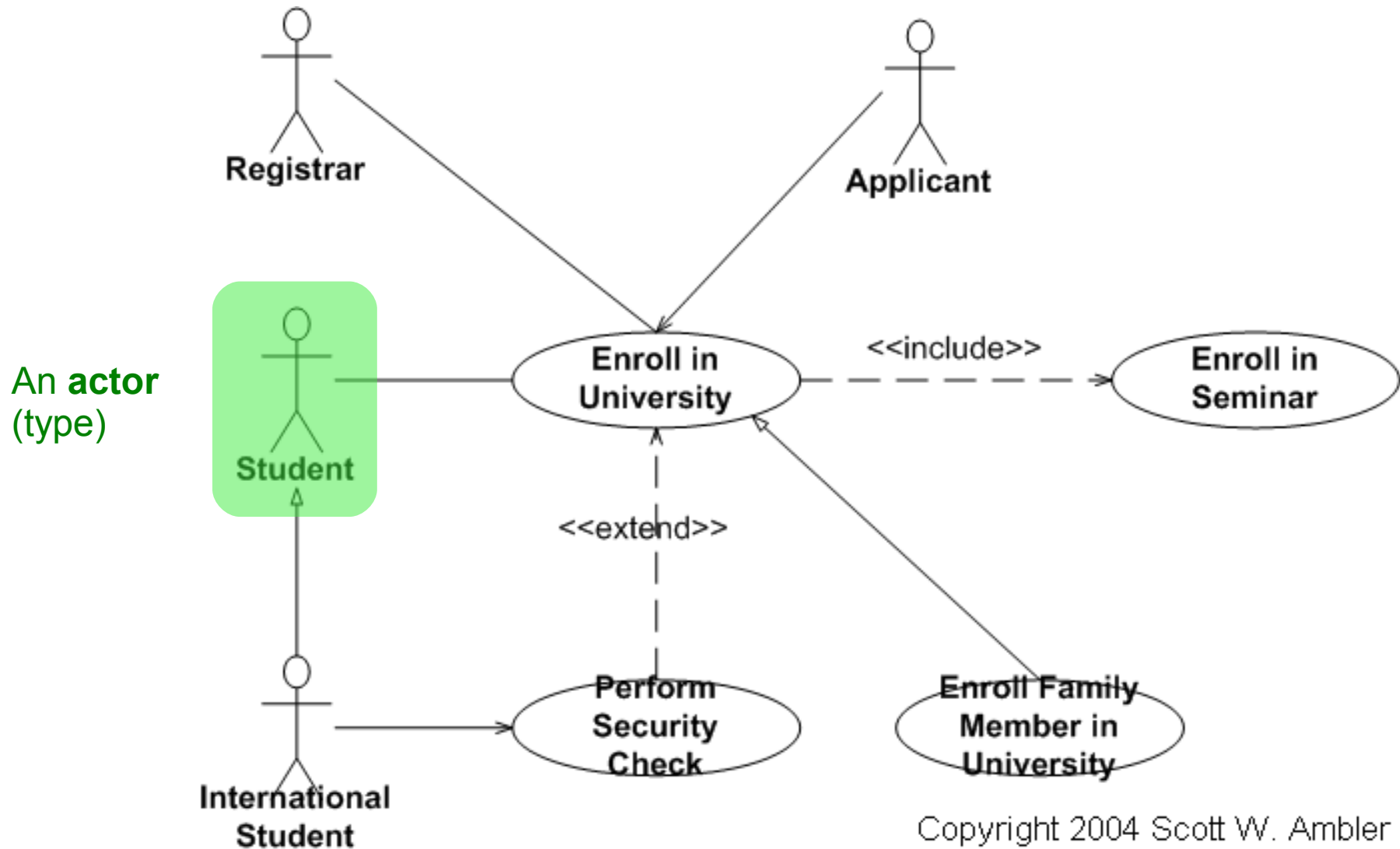
UML use case diagram notation



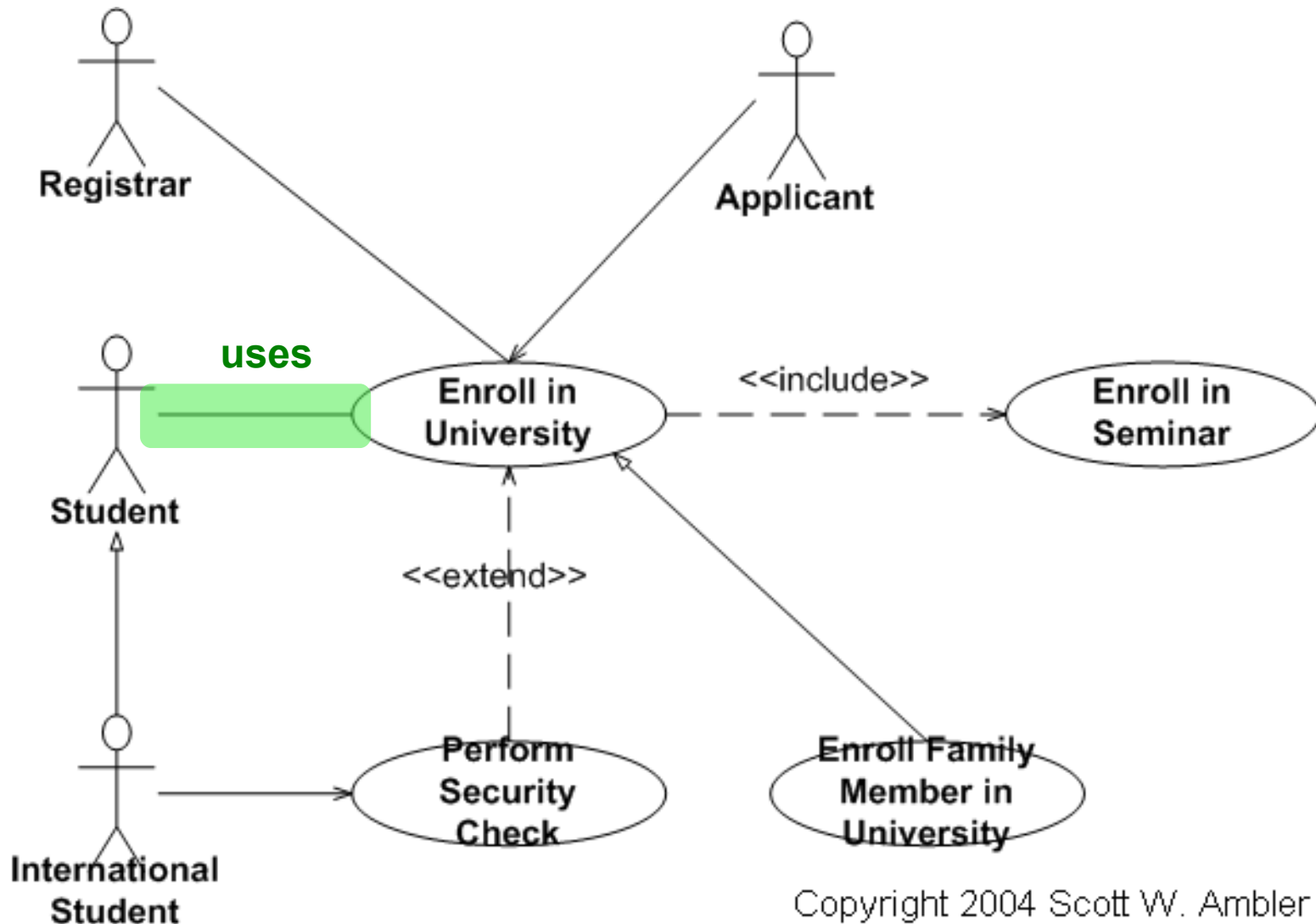
UML use case diagram notation



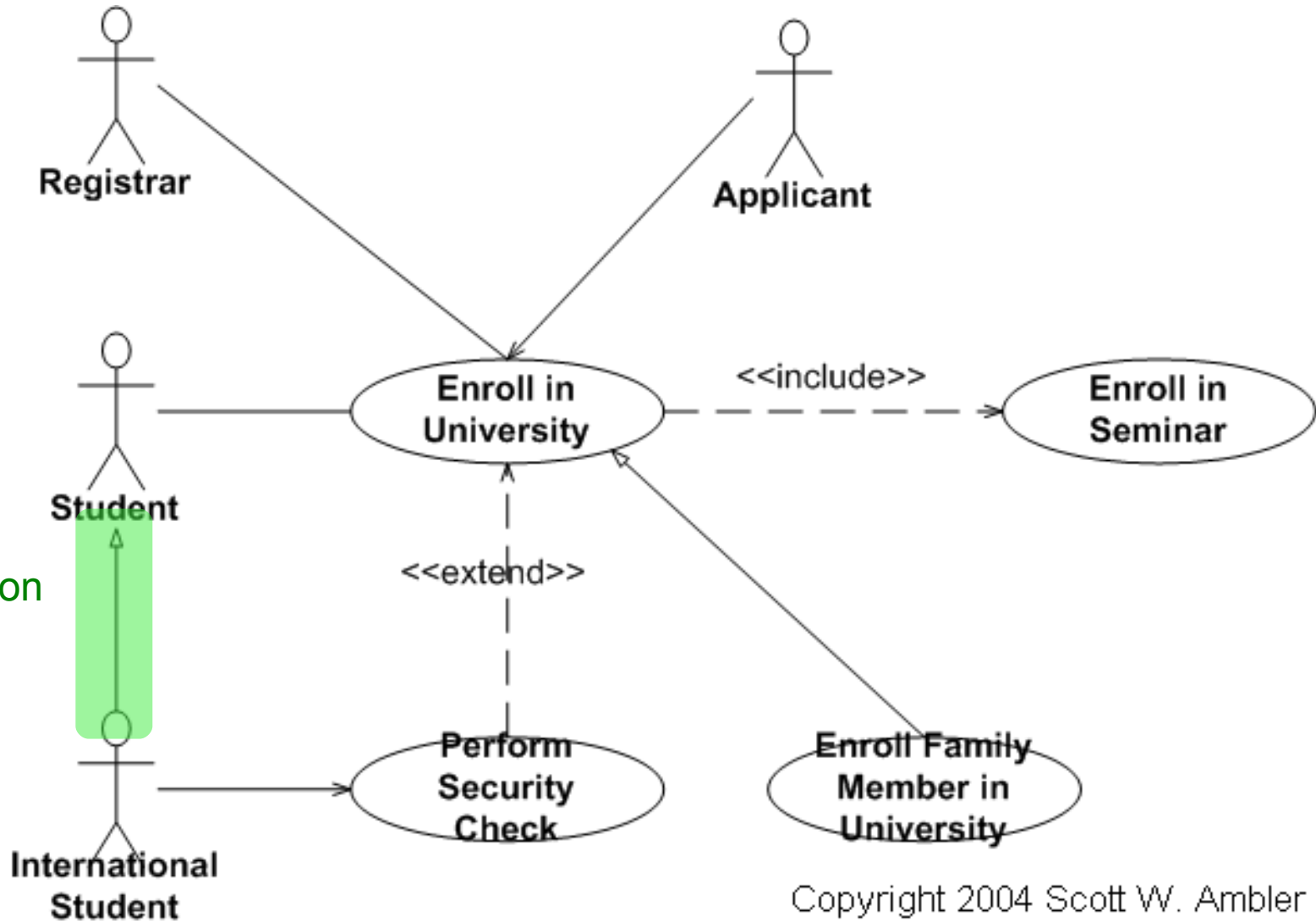
UML use case diagram notation



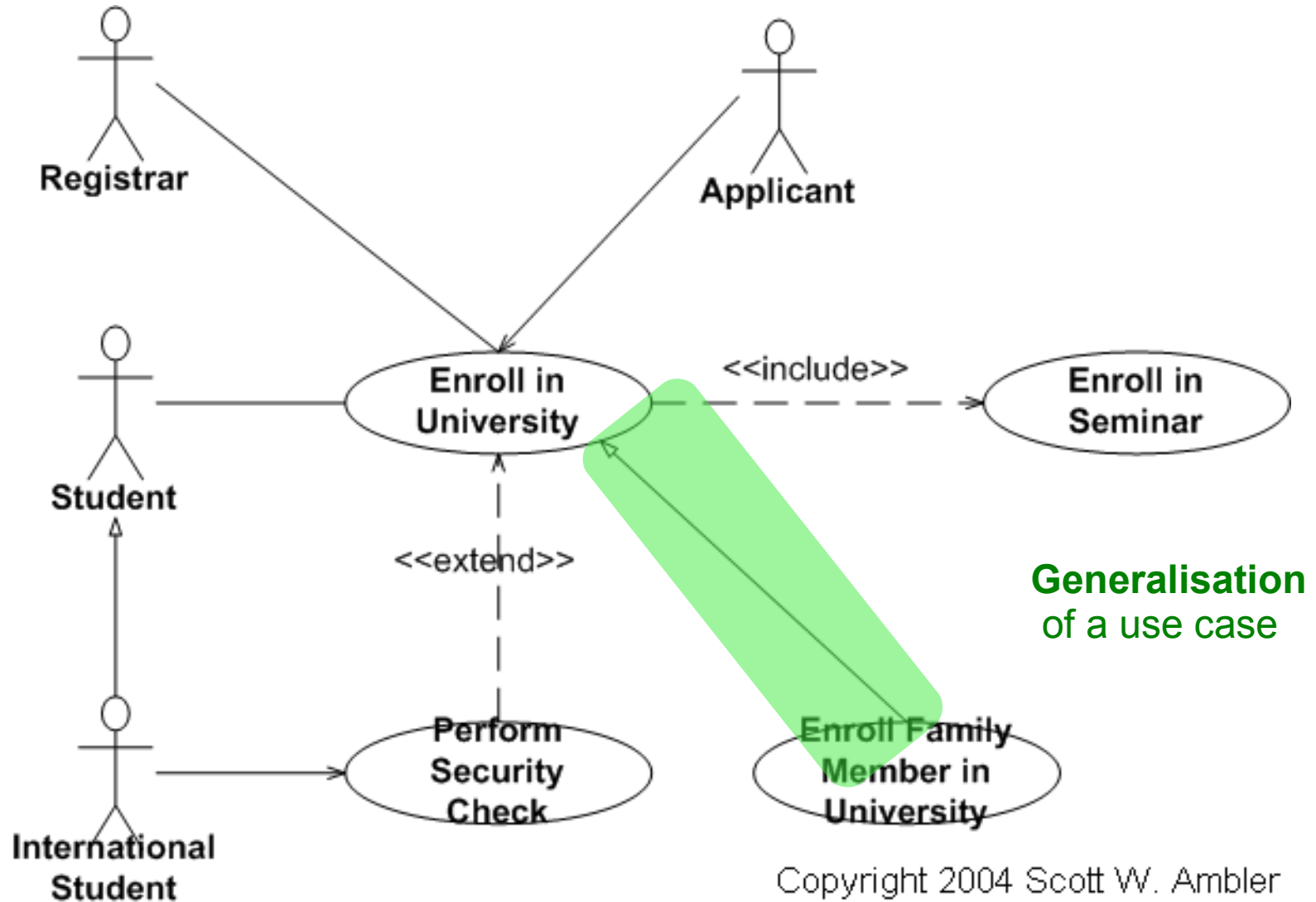
UML use case diagram notation



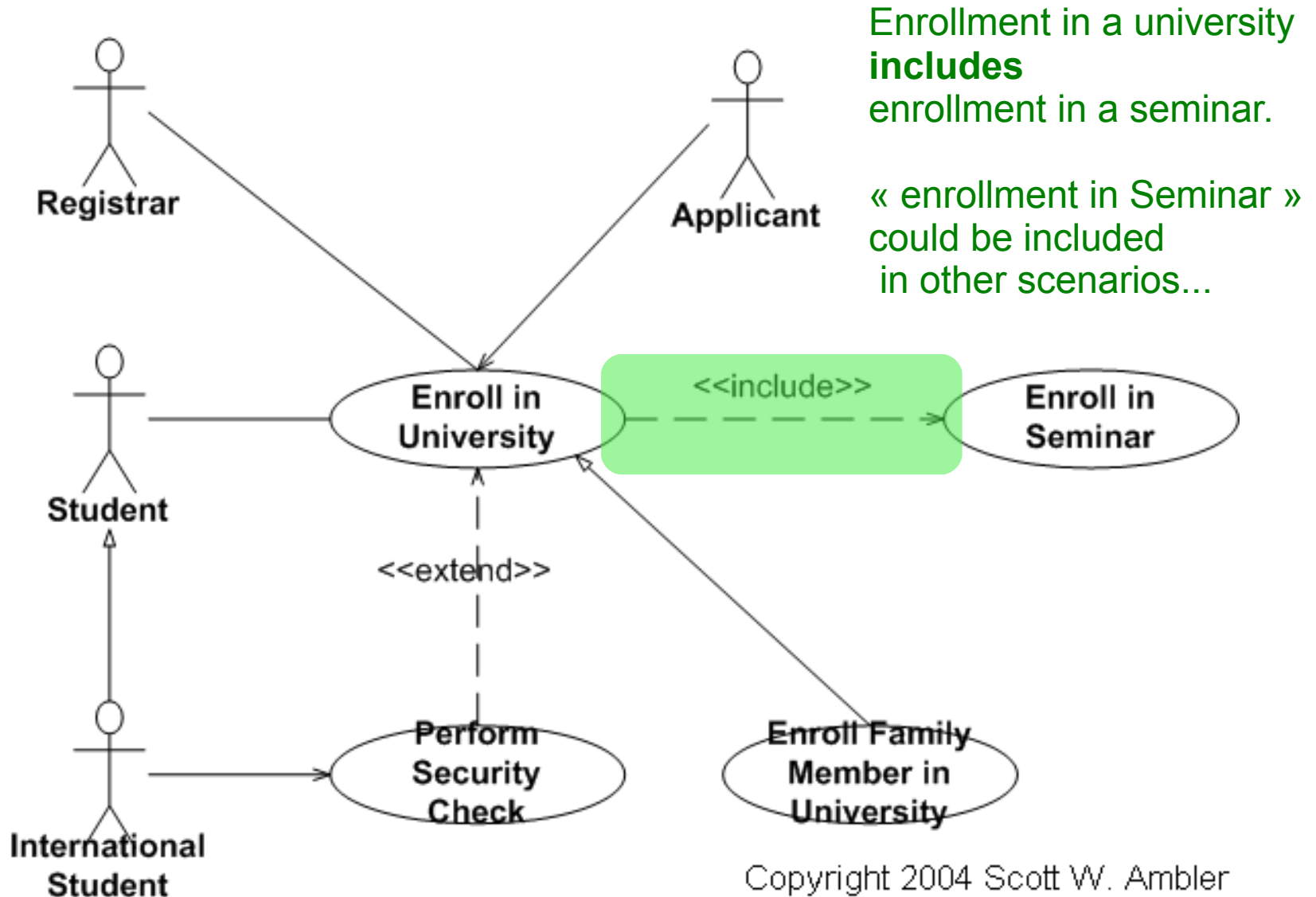
UML use case diagram notation



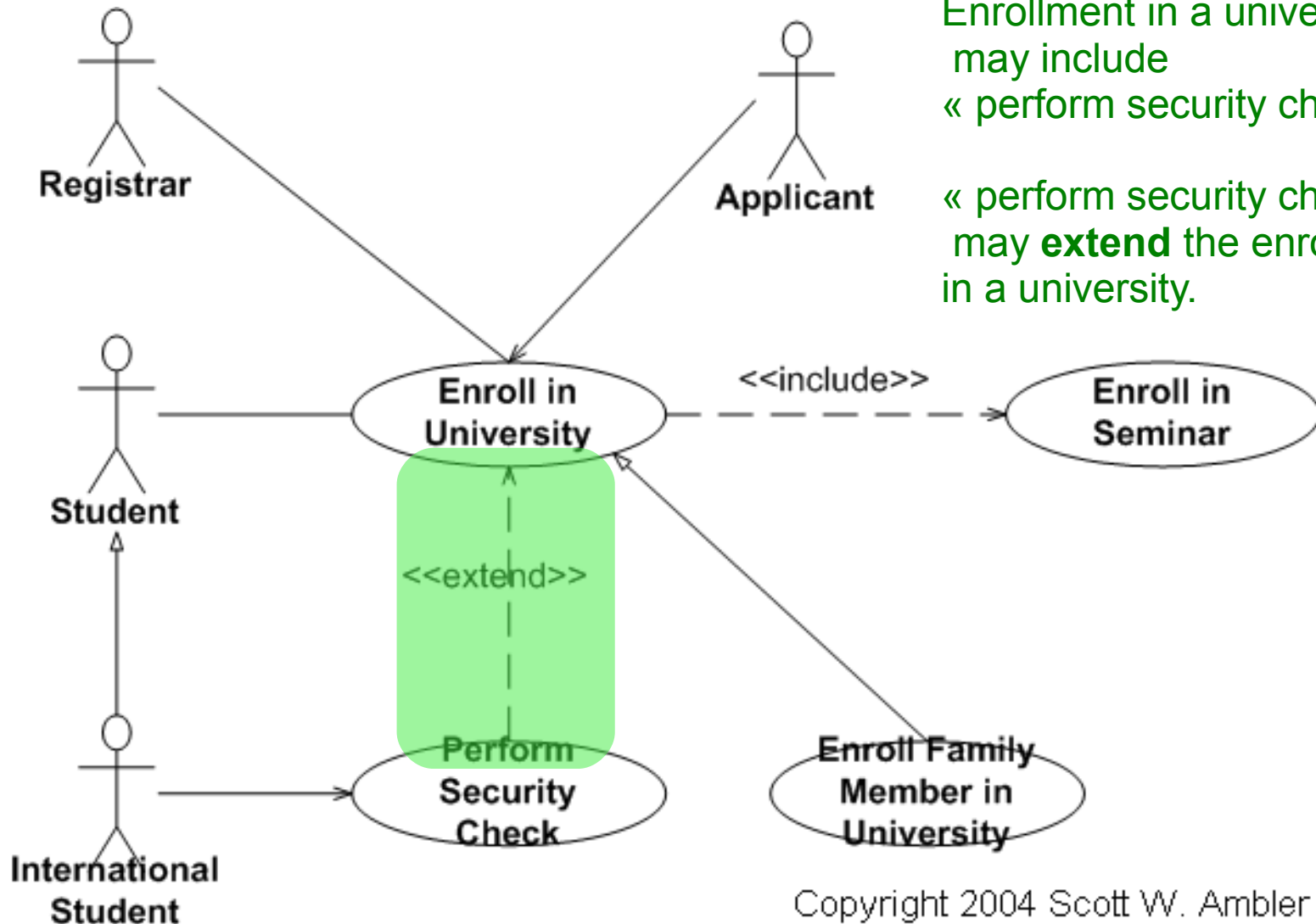
UML use case diagram notation



UML use case diagram notation



UML use case diagram notation



Enrollment in a university may include « perform security check ».

« perform security check » may **extend** the enrollment in a university.

Qui utilise des use cases ?

- À l'Observatoire de Bordeaux

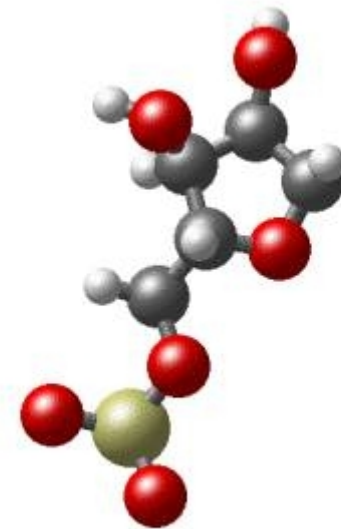
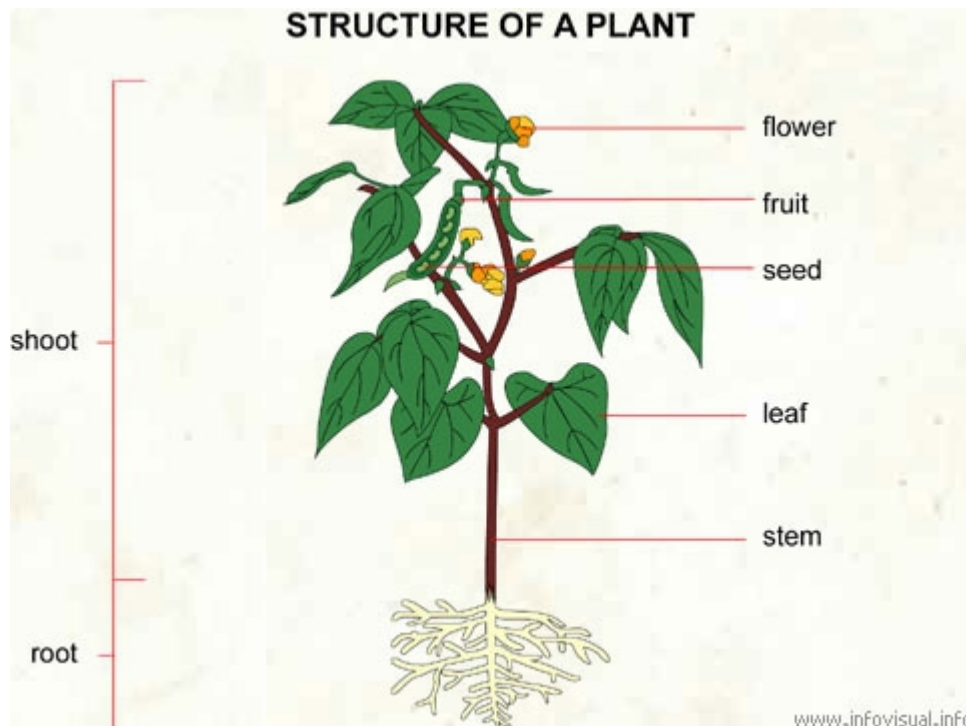
http://www.obs.u-bordeaux1.fr/amor/VWakelam/kida/kida_atelier0209_pdf/use-case.pdf

Un cours de `use case' pour astrochimistes !

Analysis of the domain

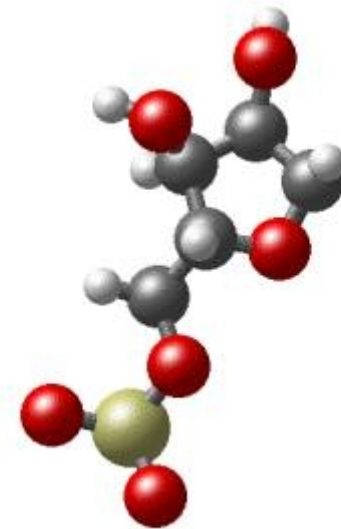
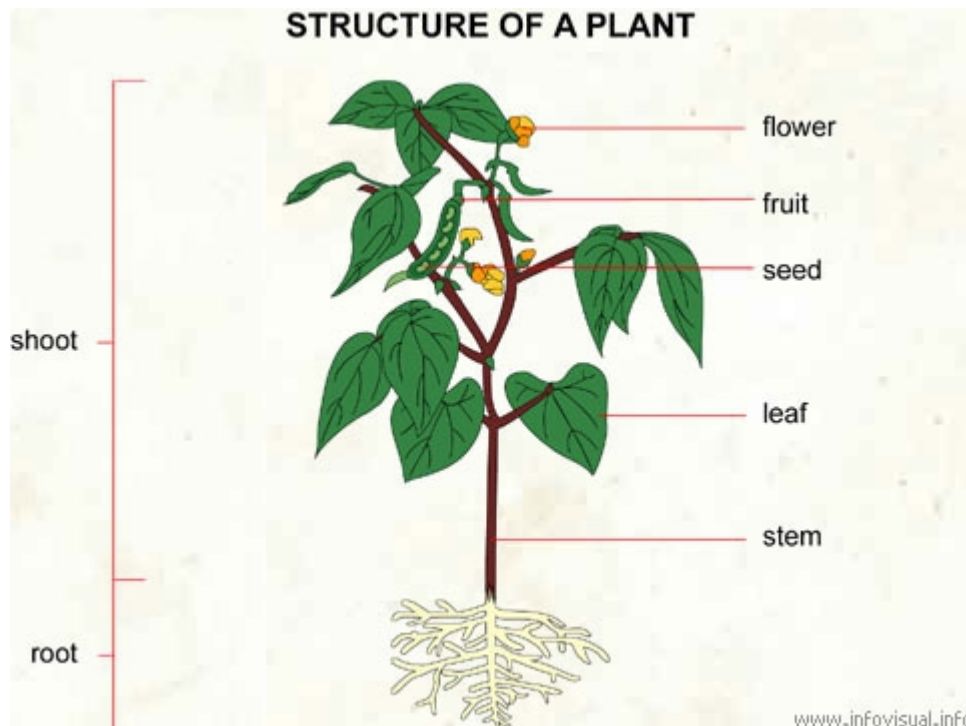
- What are the entities and their relations?

Example: students, courses, airport traffic, biology, plants, chemical products, etc.


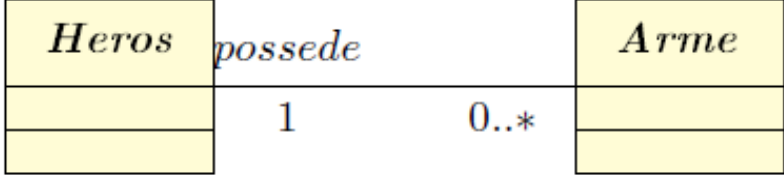


Aim: to understand the domain

- The developer is not familiar with the domain;
→ she needs to understand the client

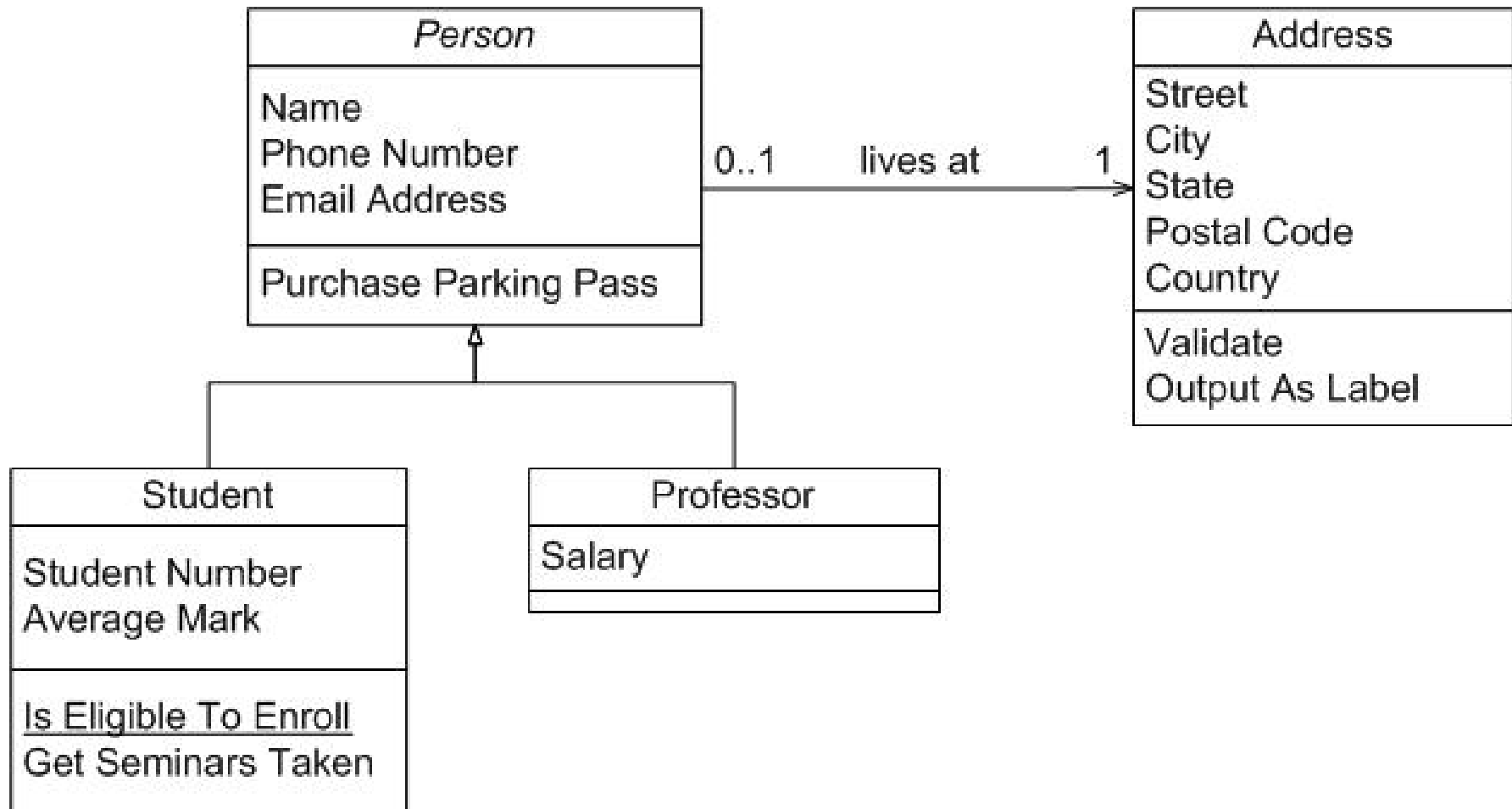


Informal/formal, textual/graphical

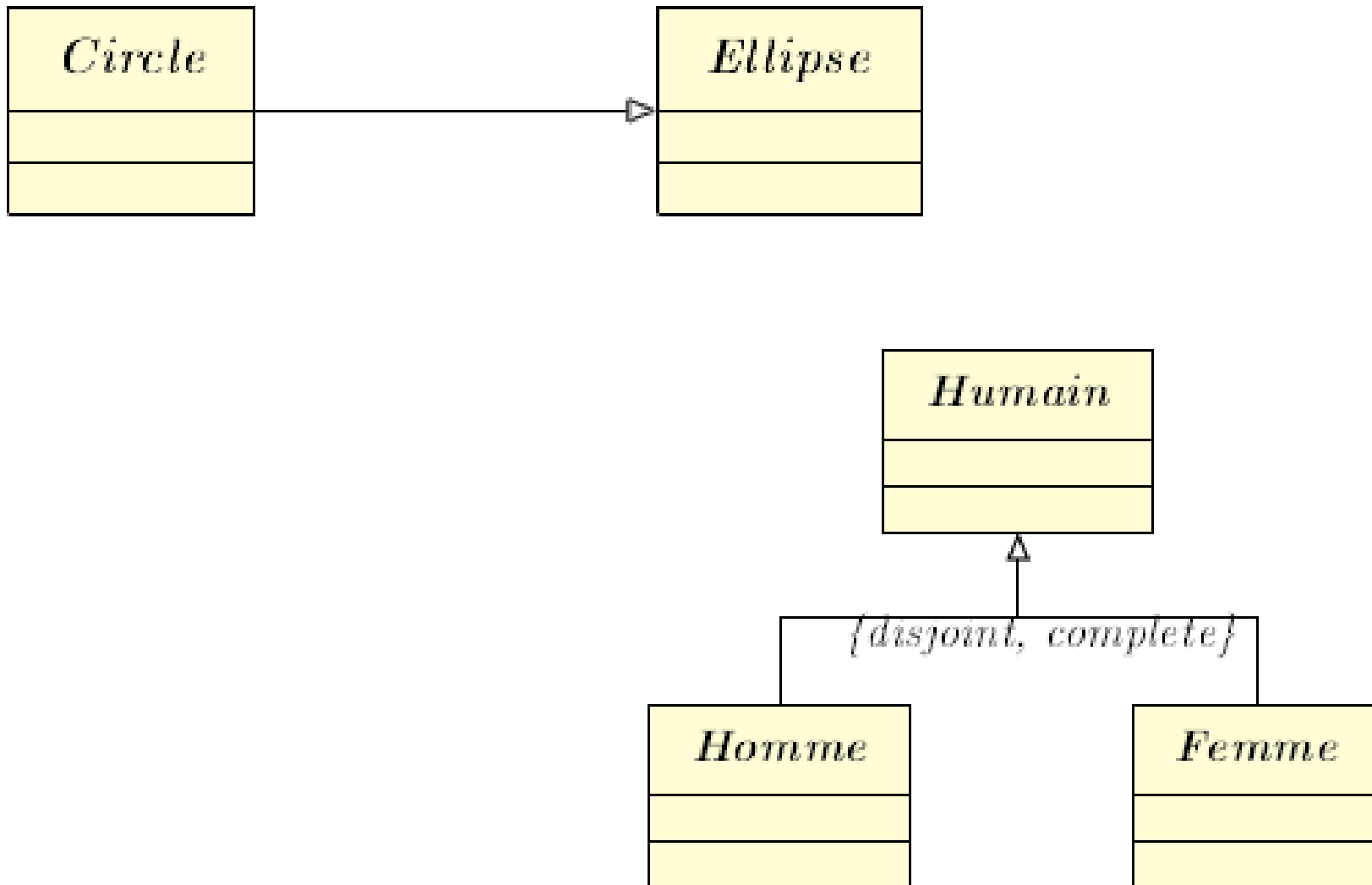
	Textual (boring)	Graphical (fun, understandable)
Informal (only for humans)	<p>Un héros possède des armes. Une arme n'est possédée par qu'un seul héros.</p>	
formal	$\forall x, \forall y, possede(x, y) \rightarrow heros(x) \wedge arme(y)$ $\forall y, card(\{x \mid possede(x, y)\}) = 1$	

- Automated generation of code
- Automated consistency checking

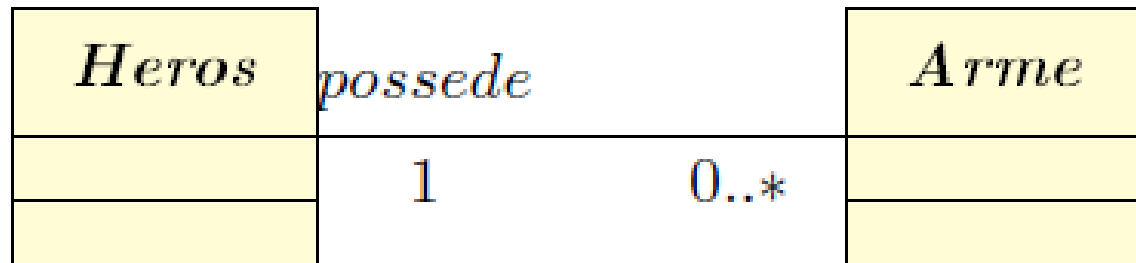
UML class diagram



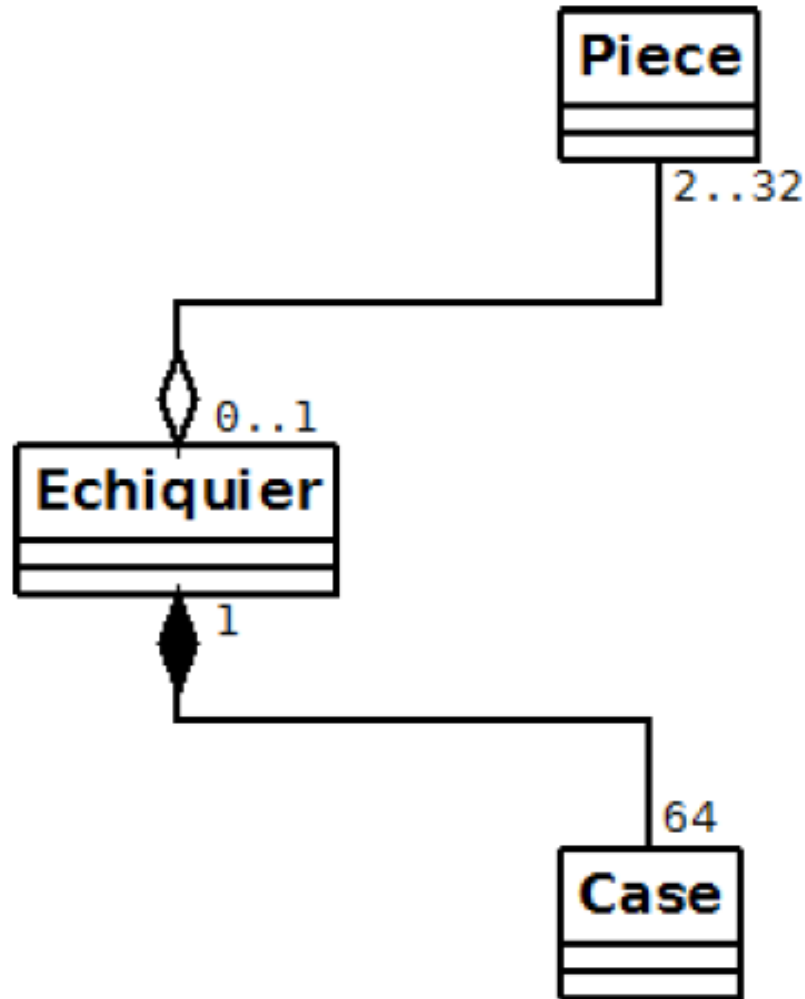
Inheritance



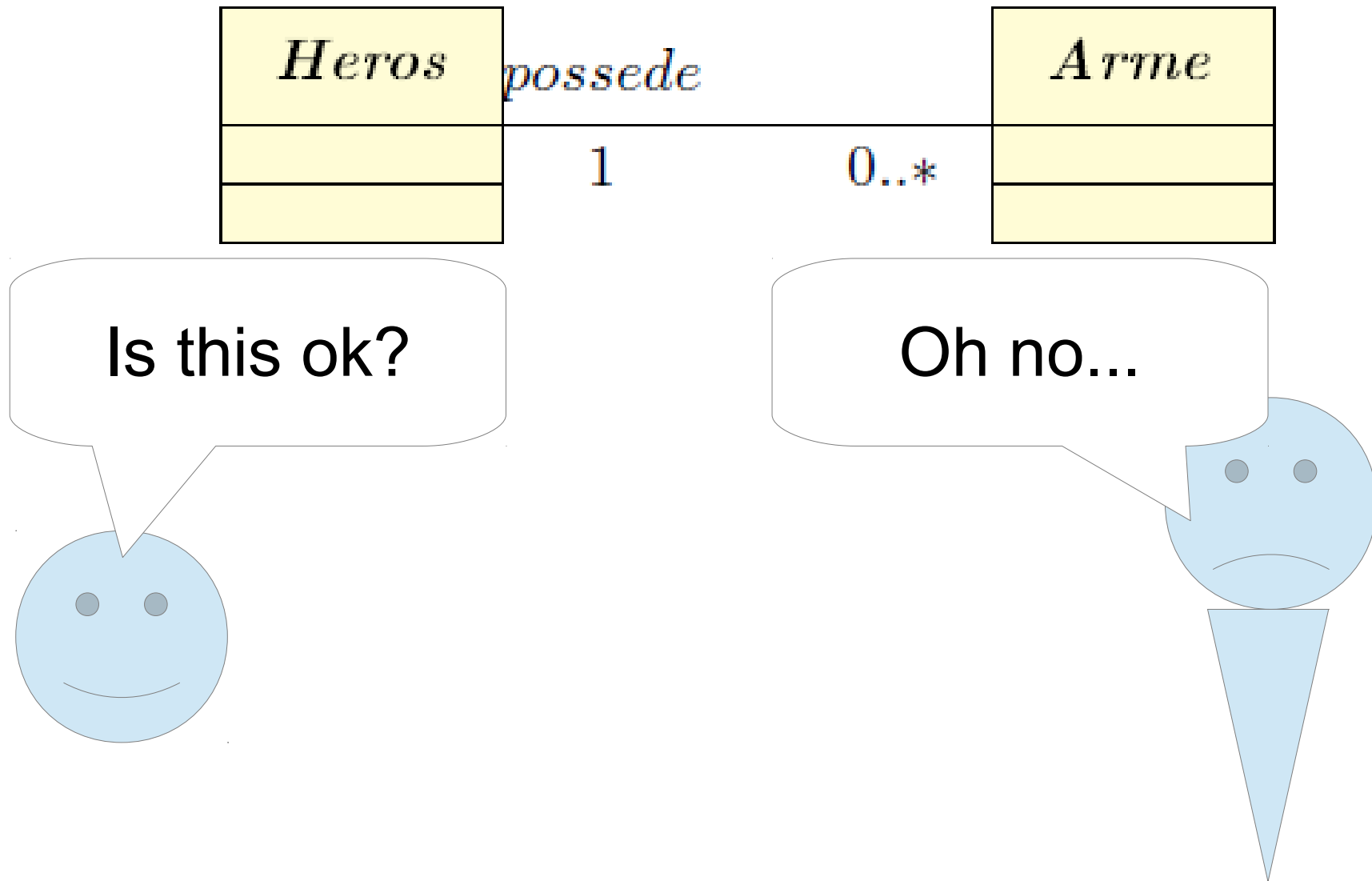
Associations



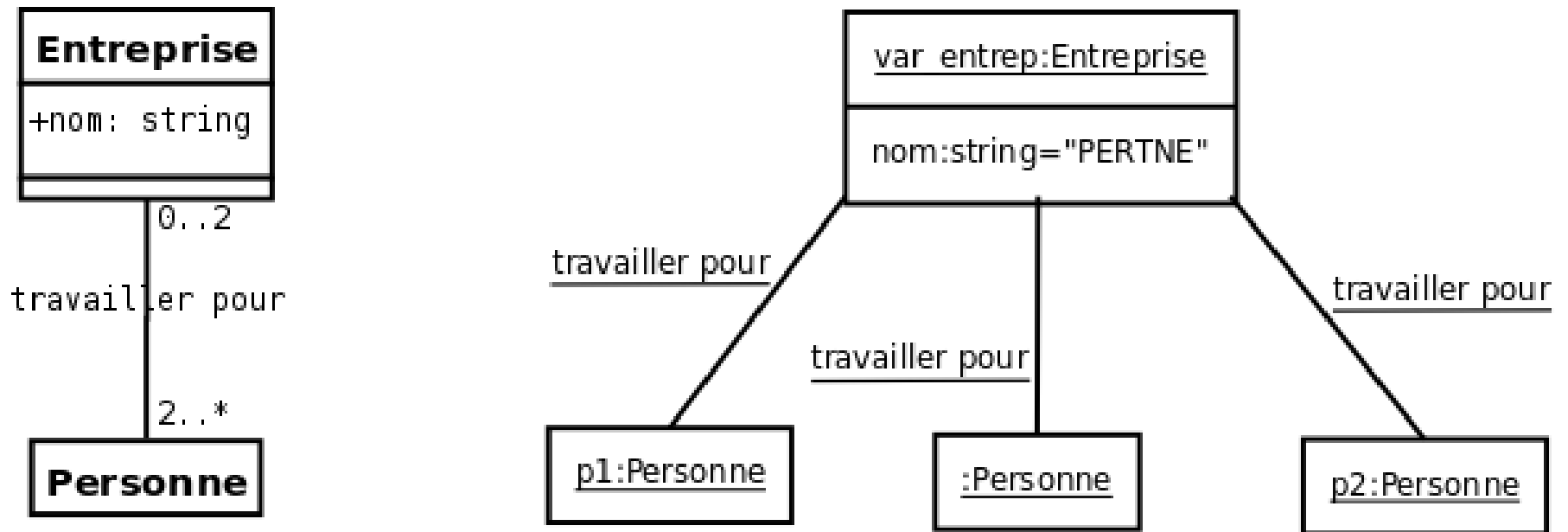
Aggregation and composition



Analysis of the domain: ping-pong with the client

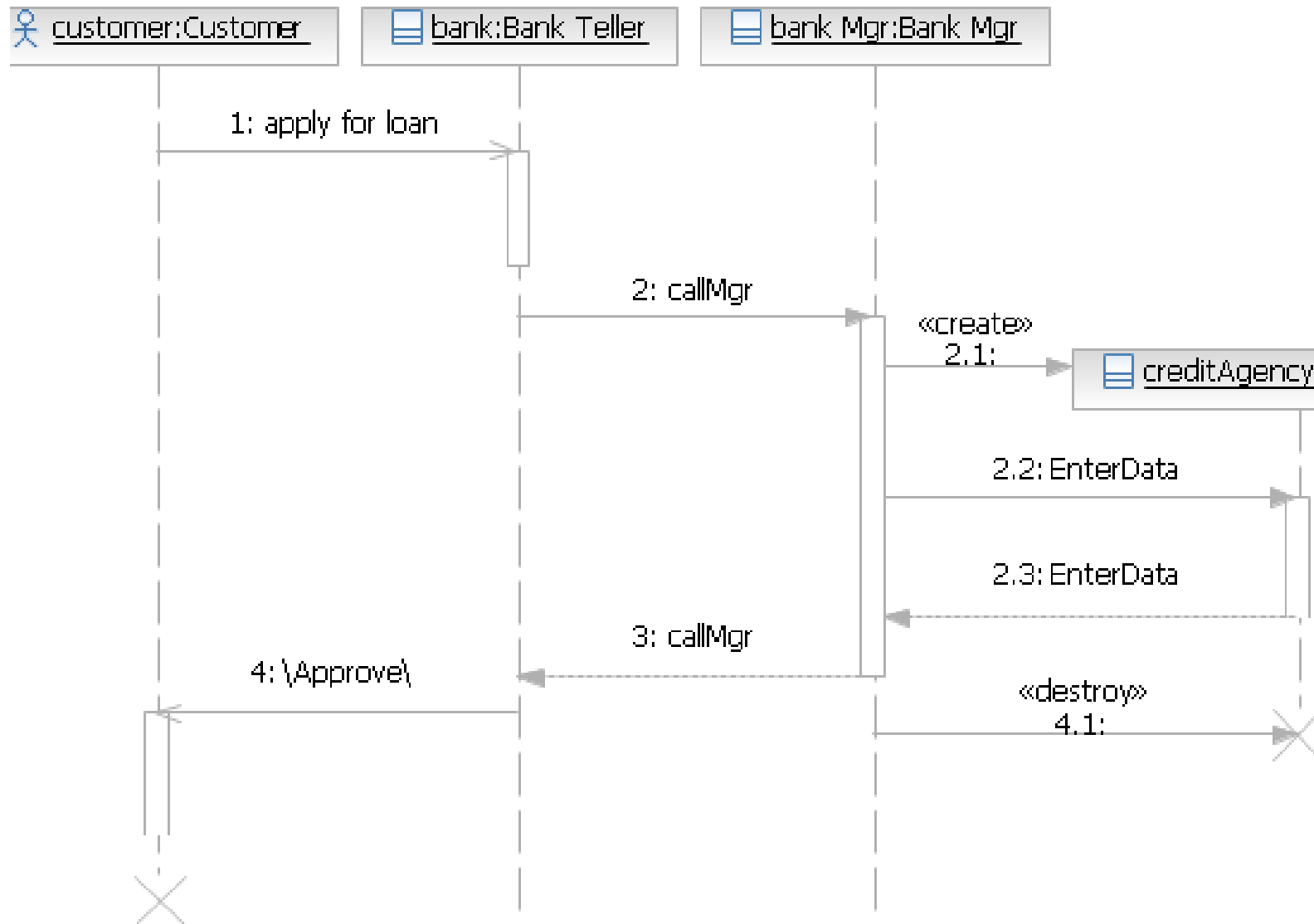


UML Object diagram




UML sequence diagram (to detail the use cases)

Interaction1



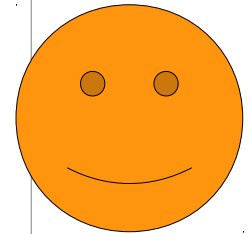
Design

- Share the work
 - decomposing in packages
- Enable to change the technology and add new features
 - design patterns



Not interesting for
the client !

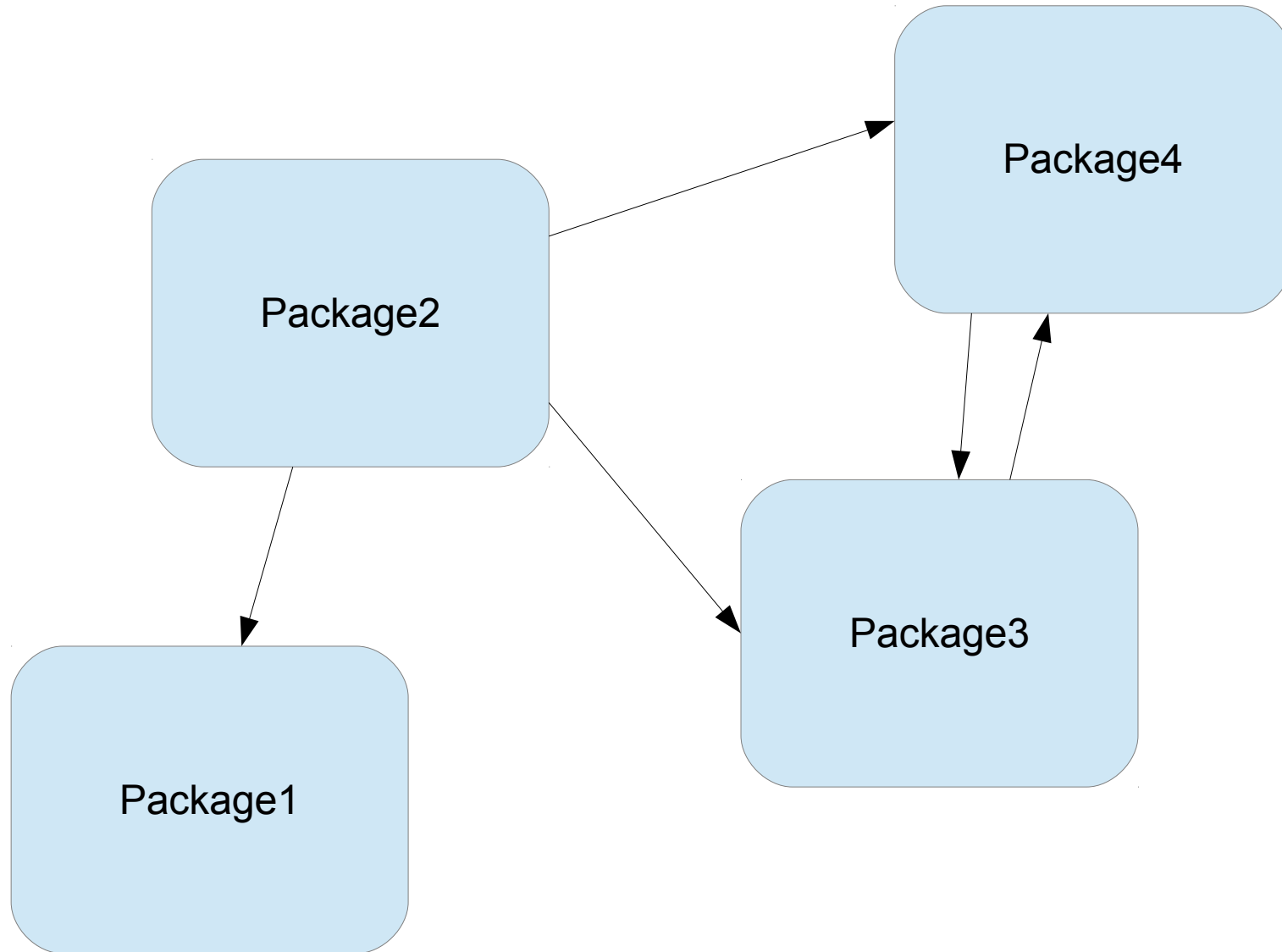
Design by contract



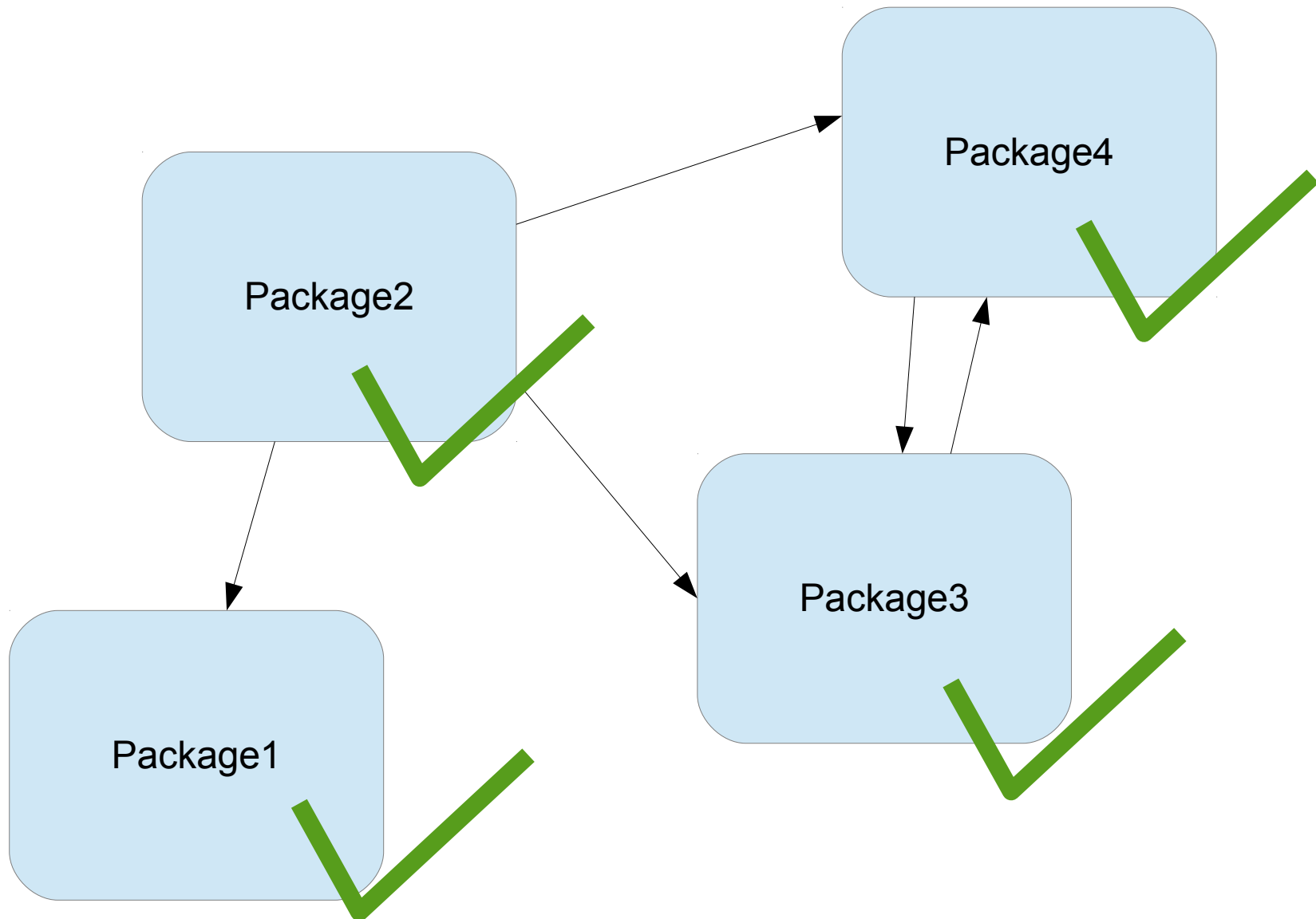
Design by contract

- **OCL**
- Eiffel
- JML

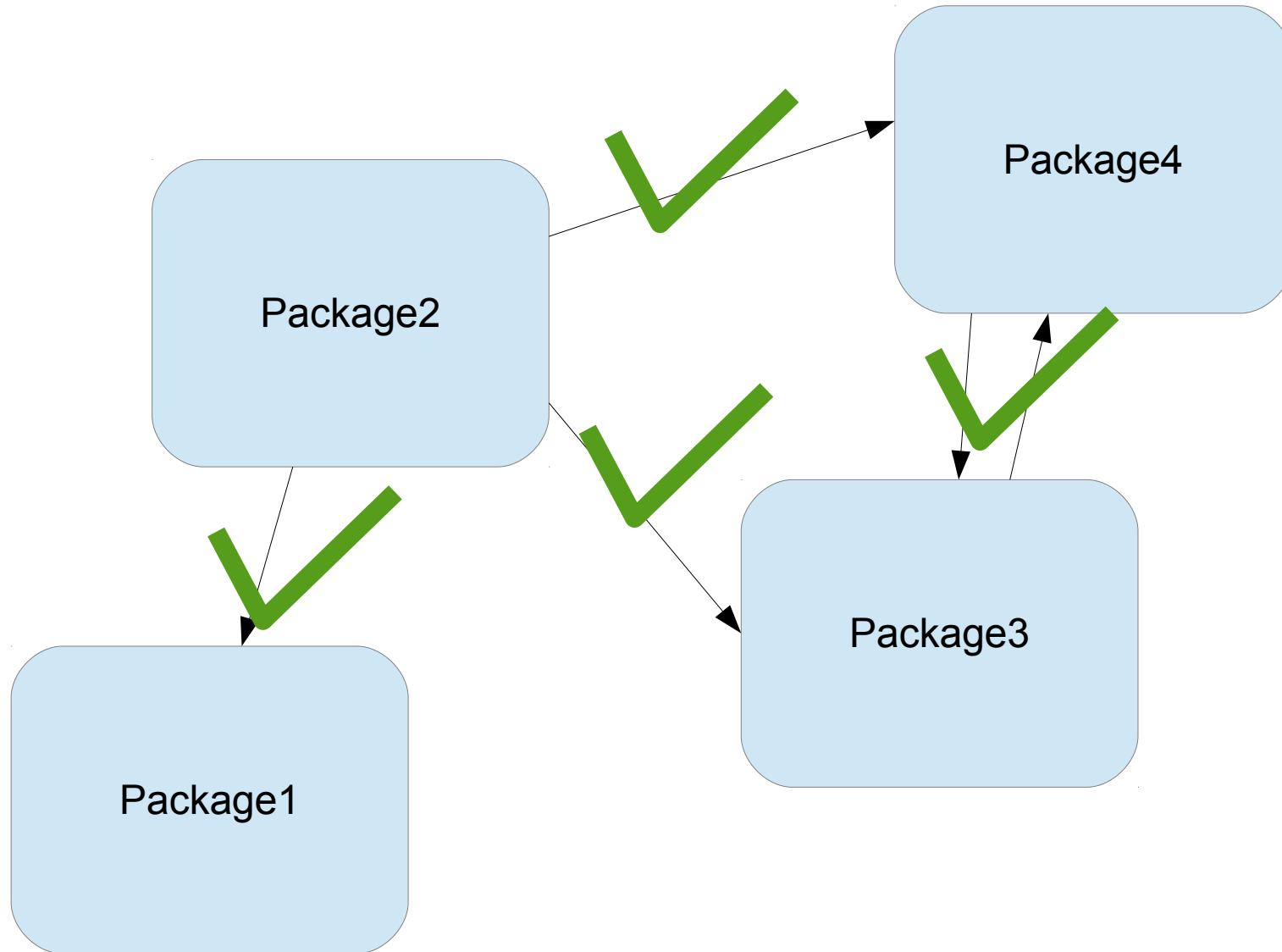
Implementation...



Unit tests



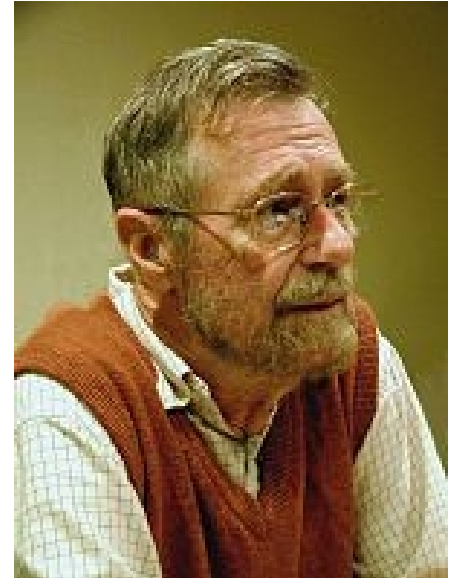
Integration tests



Test

- Program testing can be used to show the presence of bugs, but never to show their absence!

(Dijkstra's Turing Award Lecture in 1972)



Formal methods

- Proof by hands
- Proof assistant
- Model checking techniques

Proof assistant

Precondition: T

```
insertionSort(A)  
{  
  ...  
}
```

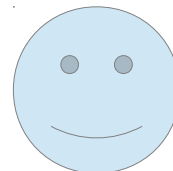


Proof

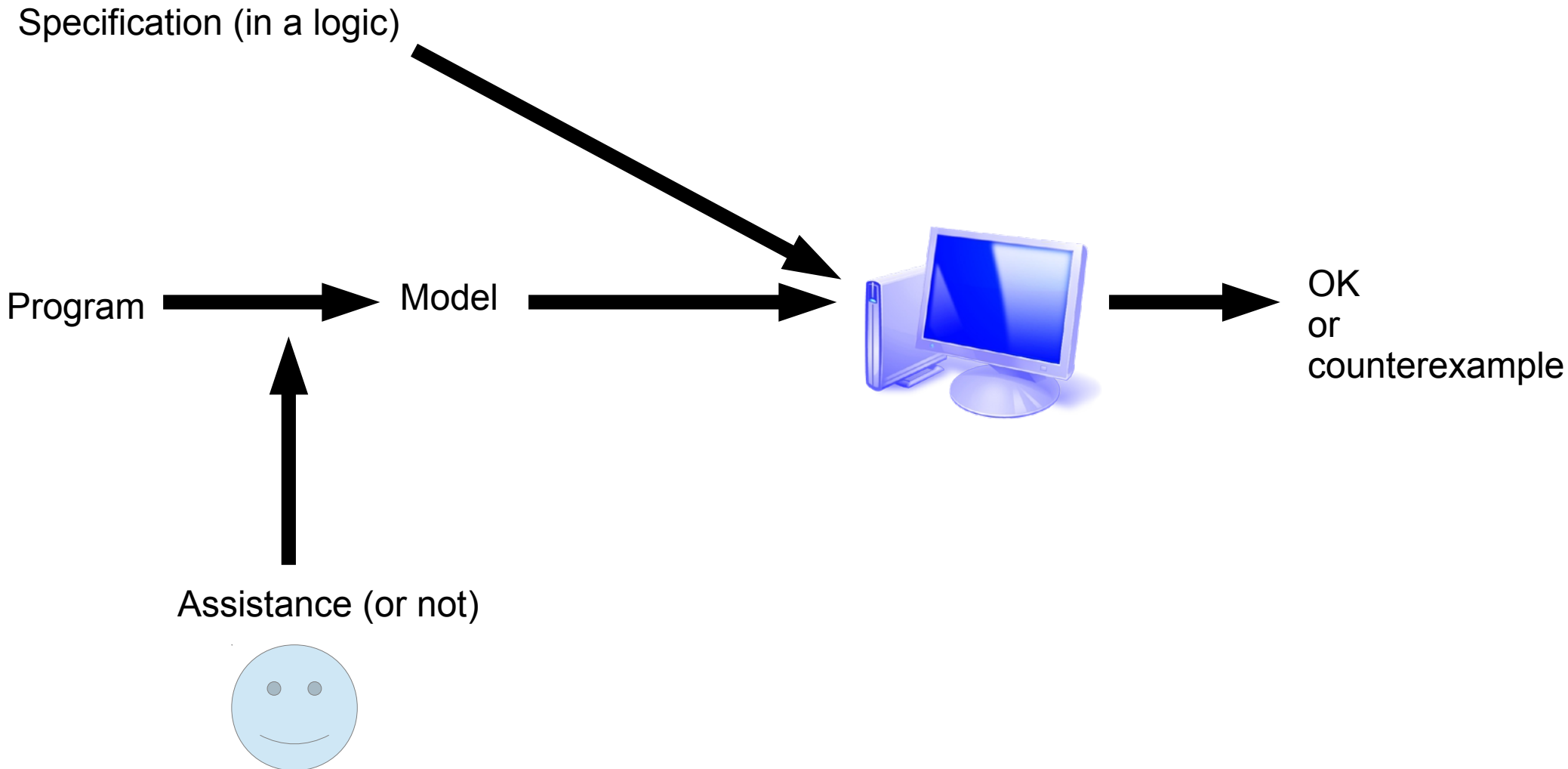
Postcondition: A is sorted



Assistance (or not)



Model checking



Tests VS formal methods

Tests

- Fastidious to create
- Easy to run
- Not complete

Formal methods

- Fastidious to use
- A HUGE cost of \$\$
 - OS kernel
\$500/lines of code
(10K lines of code)
 - NASA software
\$80/lines of code
- Complete

Who uses formal methods?



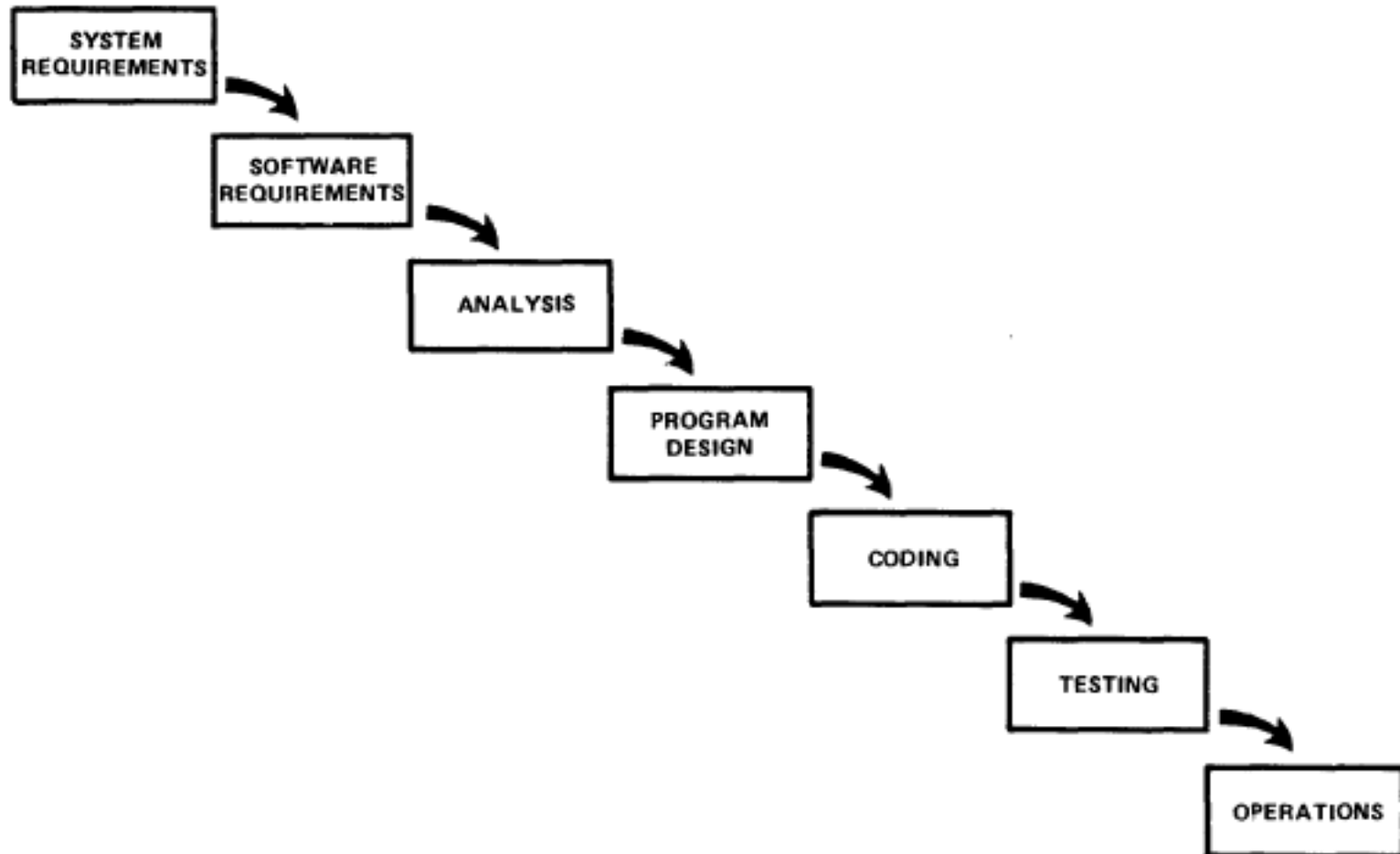
- Not critical
- Updates...

- Critical

Life cycle models

- How the activities are organized in the time?

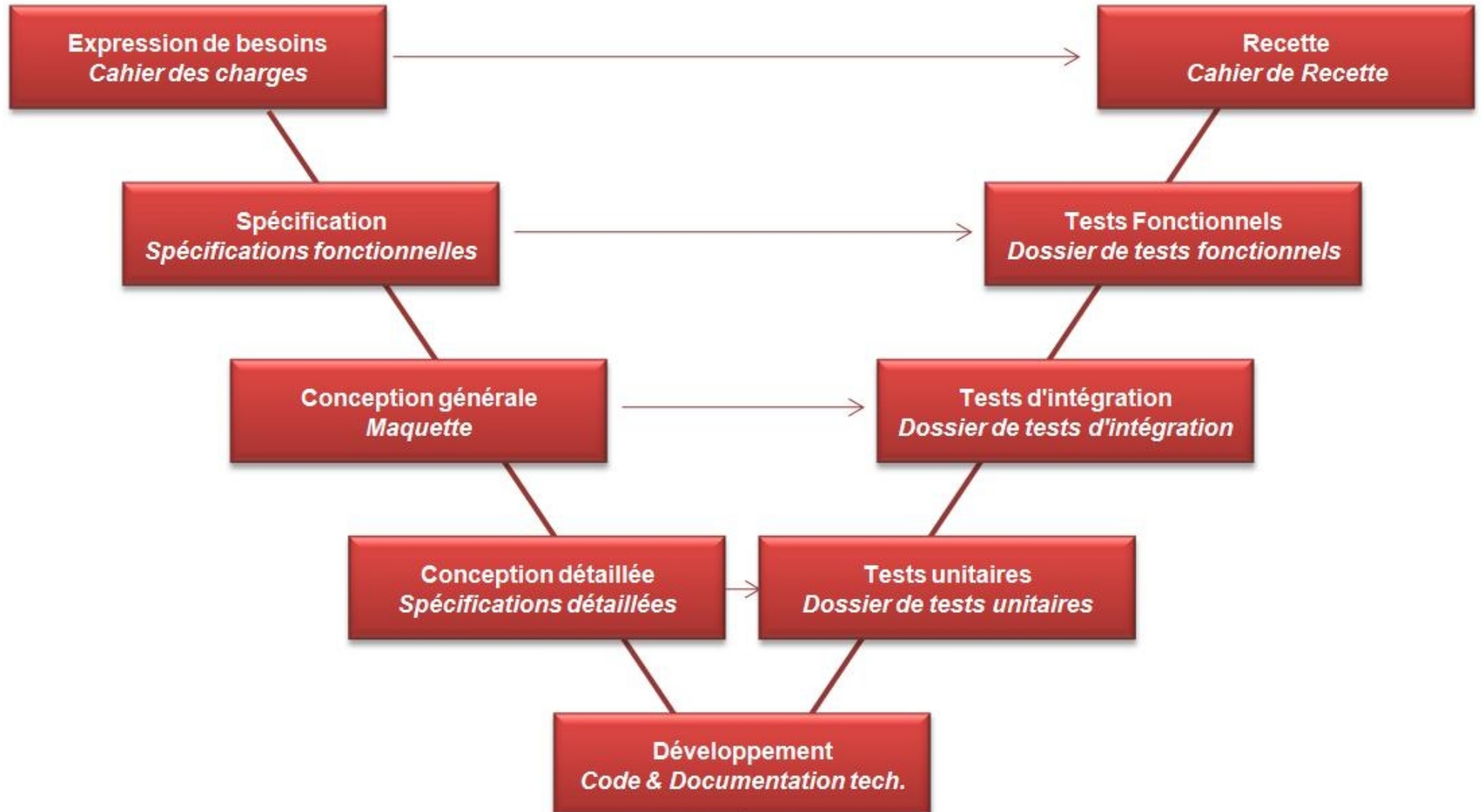
Waterfall model (does not work)



→ Distribution of the article of Royce (1970)

V-model (AFNOR)

Association française de normalisation



The risk...



Ce que demande
l'utilisateur



Ce que l'analyste a
spécifié



Ce que prévoit le
concepteur



Ce que le programmeur a
écrit

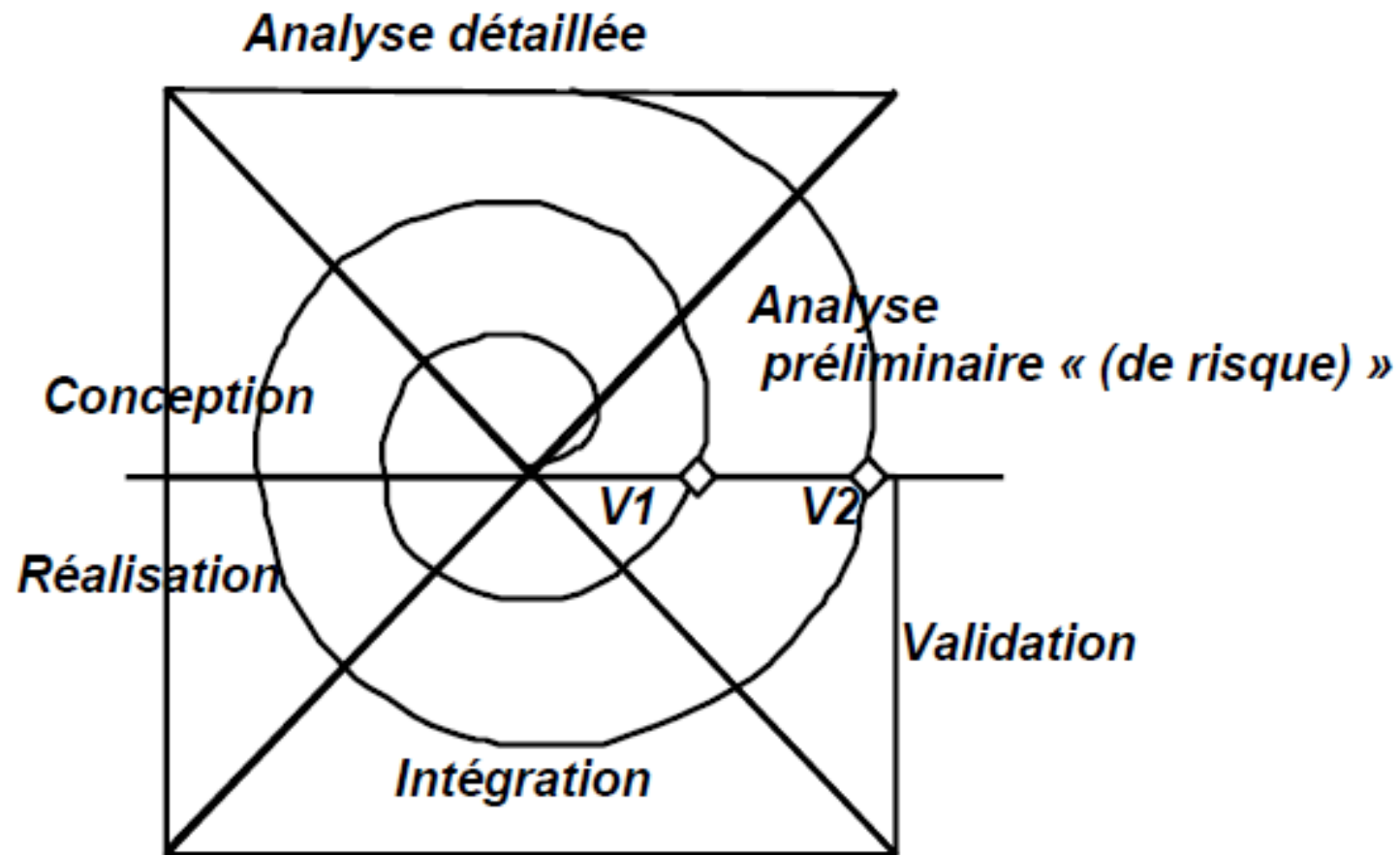


Ce que la mise au point a
fait



Ce que l'utilisateur n'a pas
su exprimer

Spiral model



Each time, we address a new use case !

Comparison

V-model

- Comfort
- Not for innovation

Spiral-model

- Disturbing
- Adapted to innovation

Agile manifesto (2001)



- Livraison fréquente
- Lien direct avec le client
- Extreme programming