Techniken zur Entwicklung Korrekter Software 1

Vorlesung vom 29.10.2007: Einführung in CASL

Christoph Lüth & Lutz Schröder

WS 07/08

Axiomatic Specfication

- loose requirements, close to informal descriptions

Axiomatic Specfication

- loose requirements, close to informal descriptions
- clarification of underlying mathematical concepts

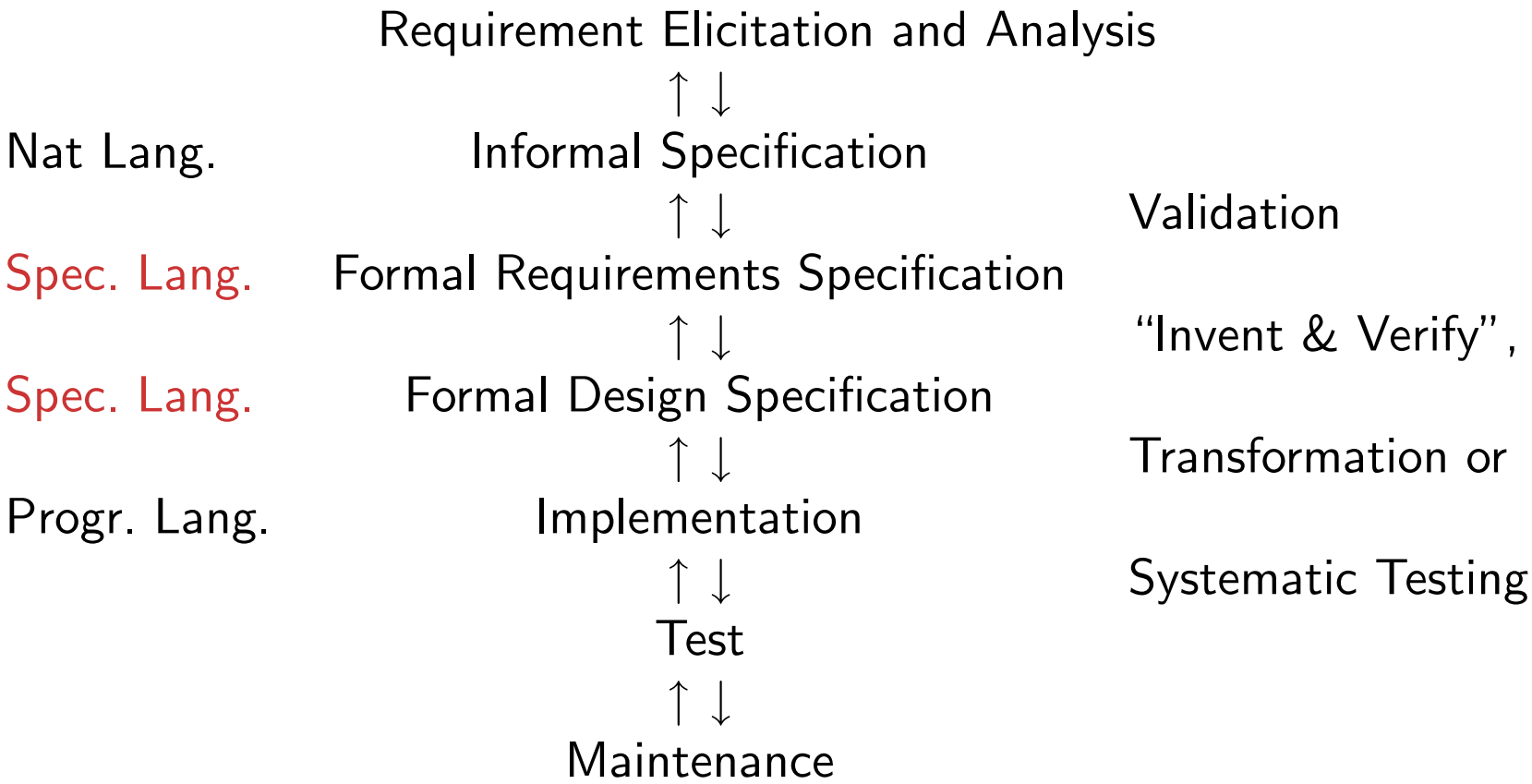
Axiomatic Specfication

- loose requirements, close to informal descriptions
- clarification of underlying mathematical concepts
- design of algorithms and data structures independently of any implementation language

Axiomatic Specfication

- loose requirements, close to informal descriptions
- clarification of underlying mathematical concepts
- design of algorithms and data structures independently of any implementation language
- CASL is a standard for axiomatic specification

Waterfall Model (slide by M. Roggenbach)



Example: sorting

Informal specification:
To sort a list means to find a list with the same elements, which is in ascending order.

Example: sorting

Informal specification:

To sort a list means to find a list with the same elements, which is in ascending order.

Formal requirements specification:

- $is\_ordered(sorter(L))$
- $is\_ordered(L) \Leftrightarrow \forall L1, L2 : List;\ x, y : Elem\,.$
  $$L = L1 + +[x, y] + +L2 \Rightarrow x \leq y$$
- $permutation(L, sorter(L))$
- $permutation(L1, L2) \Leftrightarrow$
  $$\forall x : Elem\,.\ count(x, L1) = count(x, L2)$$

Sorting (cont'd)

We want to show insert sort to enjoy these properties.
Formal design specification:

- $insert(x, []) = [x]$
- $insert(x, y :: L) =$
  $x :: y :: L)$ *when* $x \leq y$
  else $y :: insert(x, L)$
- $insert\_sort([]) = []$
- $insert\_sort(x :: L) = insert(x, insert\_sort(L))$

Implementation (in Haskell)

```
insert :: Ord a => (a,[a]) -> [a]
insert(x,[]) = [x]
insert(x,y:l) = if x <= y then x:y:l
                          else y:insert(x,l)

insertsort :: Ord a => [a] -> [a]
insertsort([]) = []
insertsort(x:l) = insert(x,insertsort(l))
```

CASL – the Common Algebraic Specification Language

- de facto standard for specification of functional requirements

CASL – the Common Algebraic Specification Language

- de facto standard for specification of functional requirements
- developed by the "Common Framework Initiative"
  (an open international collaboration)

CASL – the Common Algebraic Specification Language

- de facto standard for specification of functional requirements
- developed by the "Common Framework Initiative"
  (an open international collaboration)
- approved by IFIP WG 1.3
  "Foundations of Systems Specifications"

CASL – the Common Algebraic Specification Language

- de facto standard for specification of functional requirements
- developed by the "Common Framework Initiative"
  (an open international collaboration)
- approved by IFIP WG 1.3
  "Foundations of Systems Specifications"
- CASL User Manual (Lecture Notes in Computer Science 2900) and
  Reference Manual (Lecture Notes in Computer Science 2960)

Foundations of CASL

- detailed language summary, with informal explantation

Foundations of CASL

- detailed language summary, with informal explantation
- formal definition of abstract and concrete syntax

Foundations of CASL

- detailed language summary, with informal explantation
- formal definition of abstract and concrete syntax
- complete formal semantics

Foundations of CASL

- detailed language summary, with informal explantation
- formal definition of abstract and concrete syntax
- complete formal semantics
- proof systems

Foundations of CASL

- detailed language summary, with informal explantation
- formal definition of abstract and concrete syntax
- complete formal semantics
- proof systems
- libraries of basic datatypes

Foundations of CASL

- detailed language summary, with informal explantation
- formal definition of abstract and concrete syntax
- complete formal semantics
- proof systems
- libraries of basic datatypes

All this is contained in the Reference Manual
— here, we will largely follow the User Manual

CASL has rock-solid foundations

- the complete formal semantics maps the syntax to underlying mathematical concepts

CASL has rock-solid foundations

- the complete formal semantics maps the syntax to underlying mathematical concepts
- CASL specifications denote classes of models

CASL has rock-solid foundations

- the complete formal semantics maps the syntax to underlying mathematical concepts
- CASL specifications denote classes of models
- The semantics is largely indepdendent of the details of the logic (institution)

CASL has rock-solid foundations

- the complete formal semantics maps the syntax to underlying mathematical concepts
- CASL specifications denote classes of models
- The semantics is largely indepdendent of the details of the logic (institution)
- The semantics is the ultimative reference for the meaning of CASL

CASL on the web

- CASL in general: `http://www.cofi.info`
- CASL tools: `http://www.tzi.de/hets`
- CASL libraries: `http://www.cofi.info/Libraries`

Layers of CASL

CASL consists of several major layers, which are quite independent and may be understood (and used) separately:

Basic specifications many-sorted first-order logic, subsorting, partial functions, induction, datatypes.

Layers of CASL

CASL consists of several major layers, which are quite independent and may be understood (and used) separately:

Basic specifications many-sorted first-order logic, subsorting, partial functions, induction, datatypes.

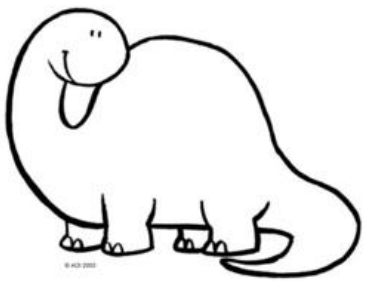
Structured specifications translation, reduction, union, and extension of specifications; generic (parametrized) and named specifications

Why Modular Decomposition?

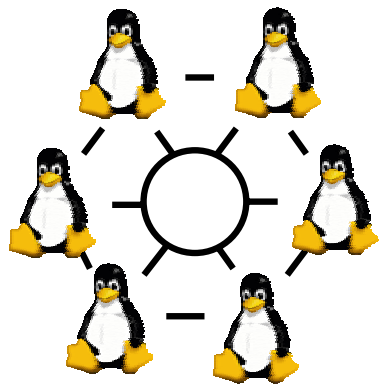
- reduction of complexity
- better understanding of specification and code
  (small pieces, well-defined interfaces)
- better distribution of work

Why Modular Decomposition?

- reduction of complexity
- better understanding of specification and code
  (small pieces, well-defined interfaces)
- better distribution of work

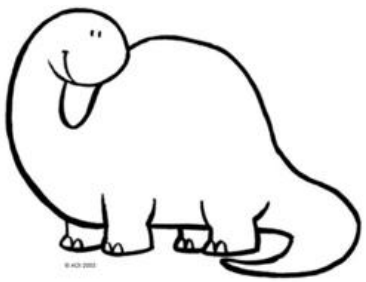


vs.

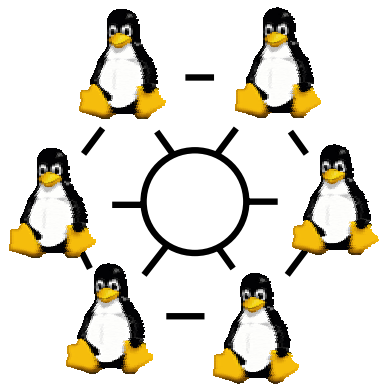


Why Modular Decomposition?

- reduction of complexity
- better understanding of specification and code
  (small pieces, well-defined interfaces)
- better distribution of work



vs.



- better maintenance and possibilities of re-use

Layers of CASL (cont'd)

Architectural specifications serve to structure implementations: define how models of a specification may be constructed out of models of simpler specifications.

Layers of CASL (cont'd)

Architectural specifications serve to structure implementations: define how models of a specification may be constructed out of models of simpler specifications.

Libraries allow the distributed (over the Internet) storage and retrieval of (particular versions of) named specifications.