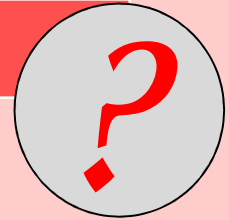


Lecture 3: Software languages and development basics

- Review: programming paradigms
- Comparing software languages (C, C++ and Python)
- Software development method
- Software development process



Review: programming



What are the four operations of John-von-Neumann-CPU-workflow and what do they do?

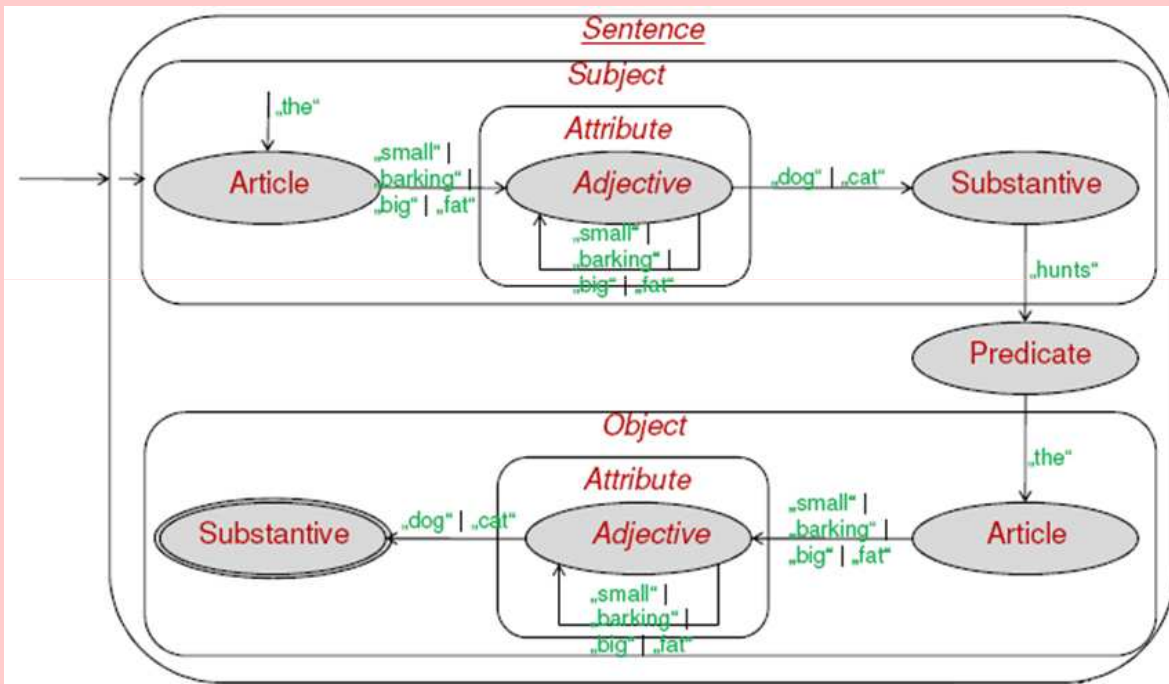
What is an Assembler? Give 3 advantages and 3 disadvantages of Assembler



Review: language theory



Check the following sentences with the grammar given by graph if they can be produced



“the dog hunts the big fat cat”
“the small dog hunts the small cat”
“the small barking dog hunts the big fat cat”
“the small small small small dog hunts the barking fat dog”

Change the above graph so that it also accepts the sentences “the small dog hunts the fat cat” and “the small cat hunts”.

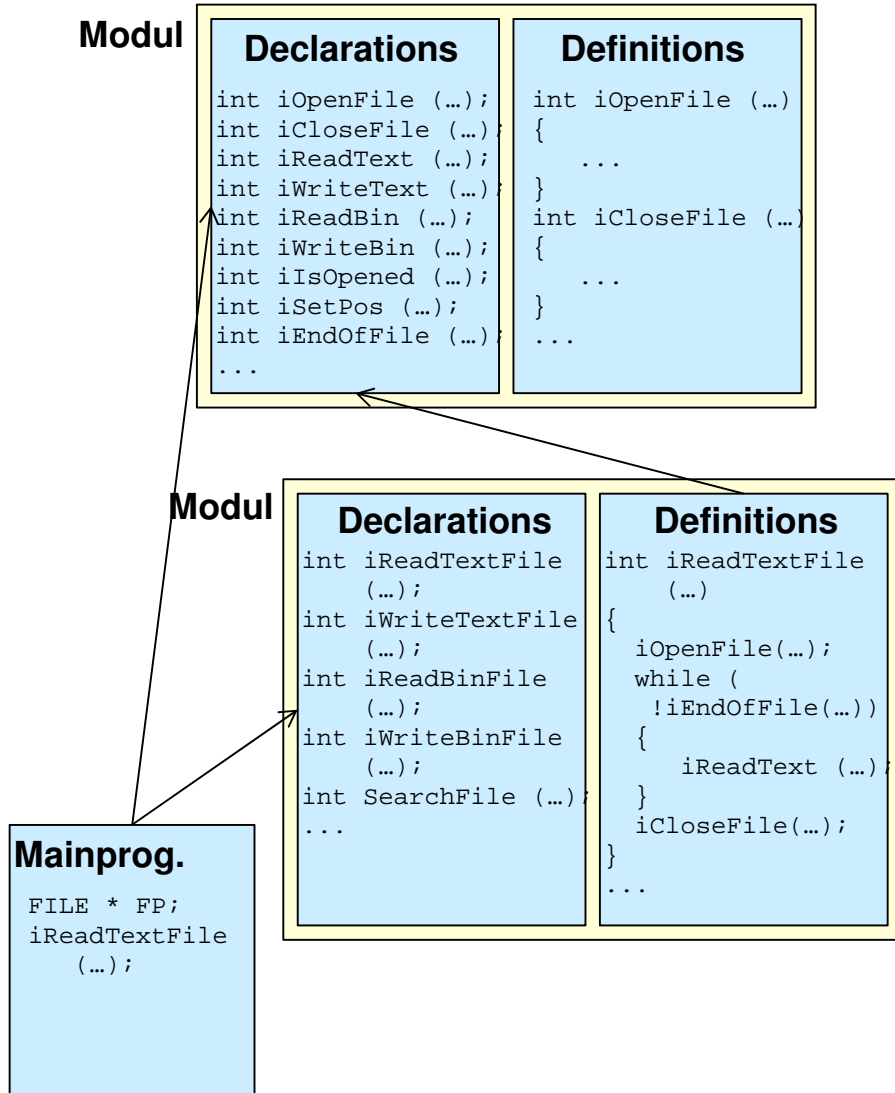
Review: Programming paradigms

Programming paradigms for problem oriented computer languages

- **Sequential, unstructured programming**
 - *Sequence of instructions with jumps to dedicated code positions*
 - *E.g. Assembler*
- **Procedural programming with structured programming as subset**
 - *Code is splitted into several, reusable sections called procedures or functions with own scopes, which can be called at given code positions*
 - *Logical procedure units are combined to modules*
 - *Jumps (like goto) are not allowed*
 - *E.g. Pascal, C*
- **Object oriented programming**
 - *Using objects (defined with class structures) and messages (interface method calls) to design applications*
 - *Extended techniques, like inheritance, modularity, polymorphism and encapsulation*
 - *E.g. Smalltalk, C++, Java*
- **Aspect oriented programming**
 - *Cross cutting concern as additional functionality which is not immediatly relevant for the functionality of a software itself but very important for developement, like error prevention, simulation and code investigation*
 - *Working with aspects as additional descriptions to the classes*
 - *E.g. extensions to C++*
- **Dataflow driven programming**
 - *State changes in data flows cause the execution of functionality*
 - *Message passing*
 - *E.g. NI LabView*

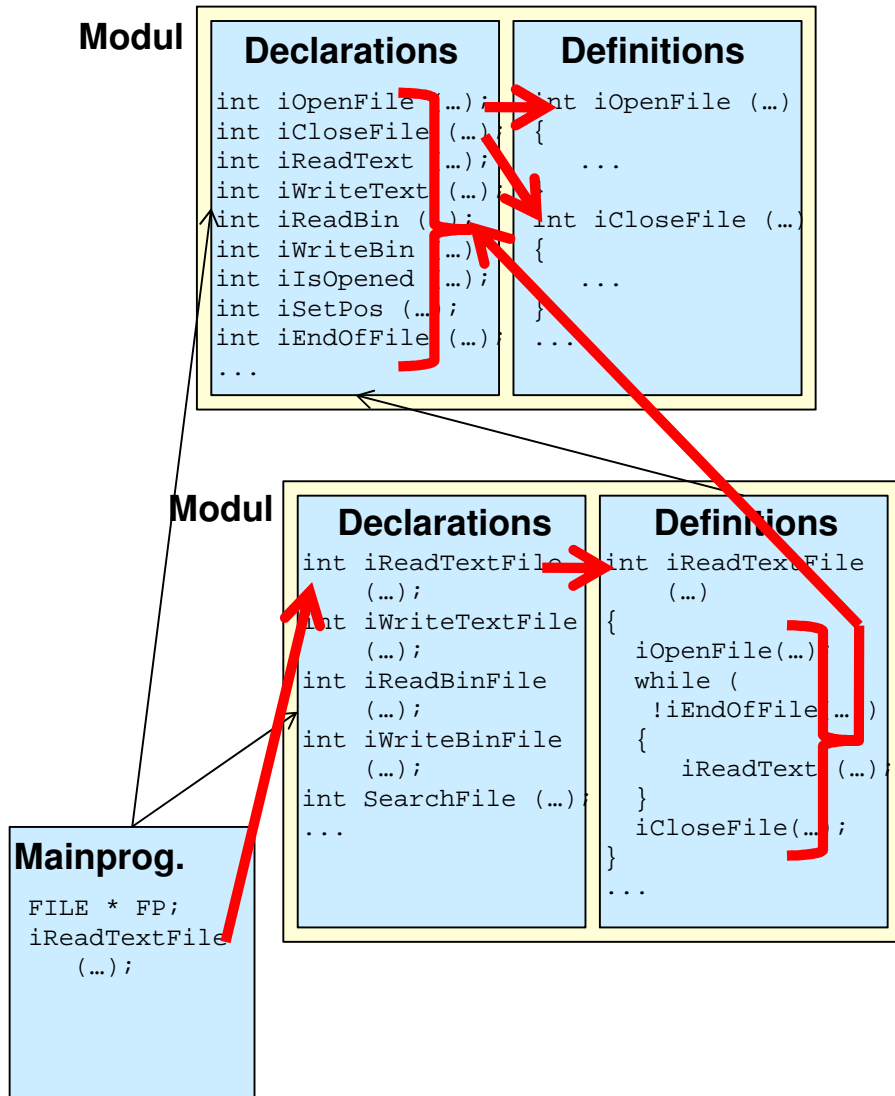
Review: Programming paradigms

Structured programming vs. Object oriented programming



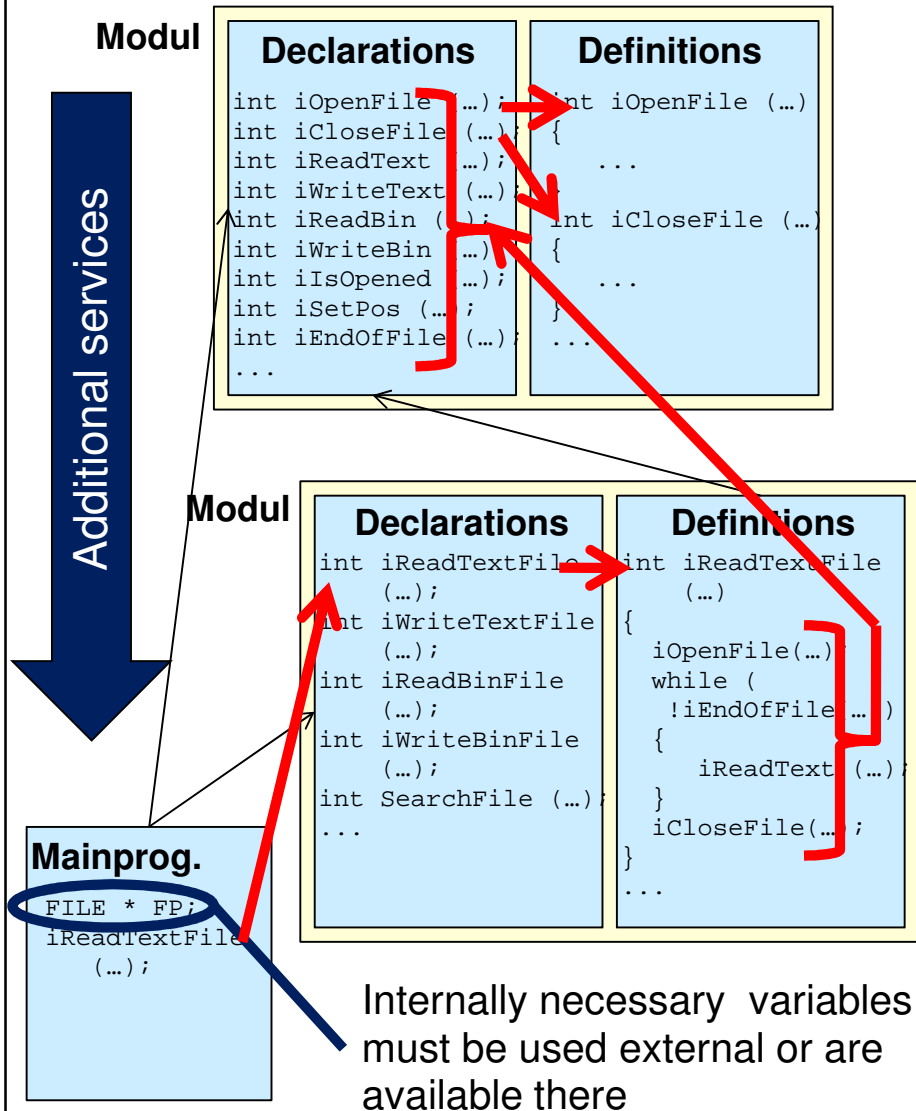
Review: Programming paradigms

Structured programming vs. Object oriented programming



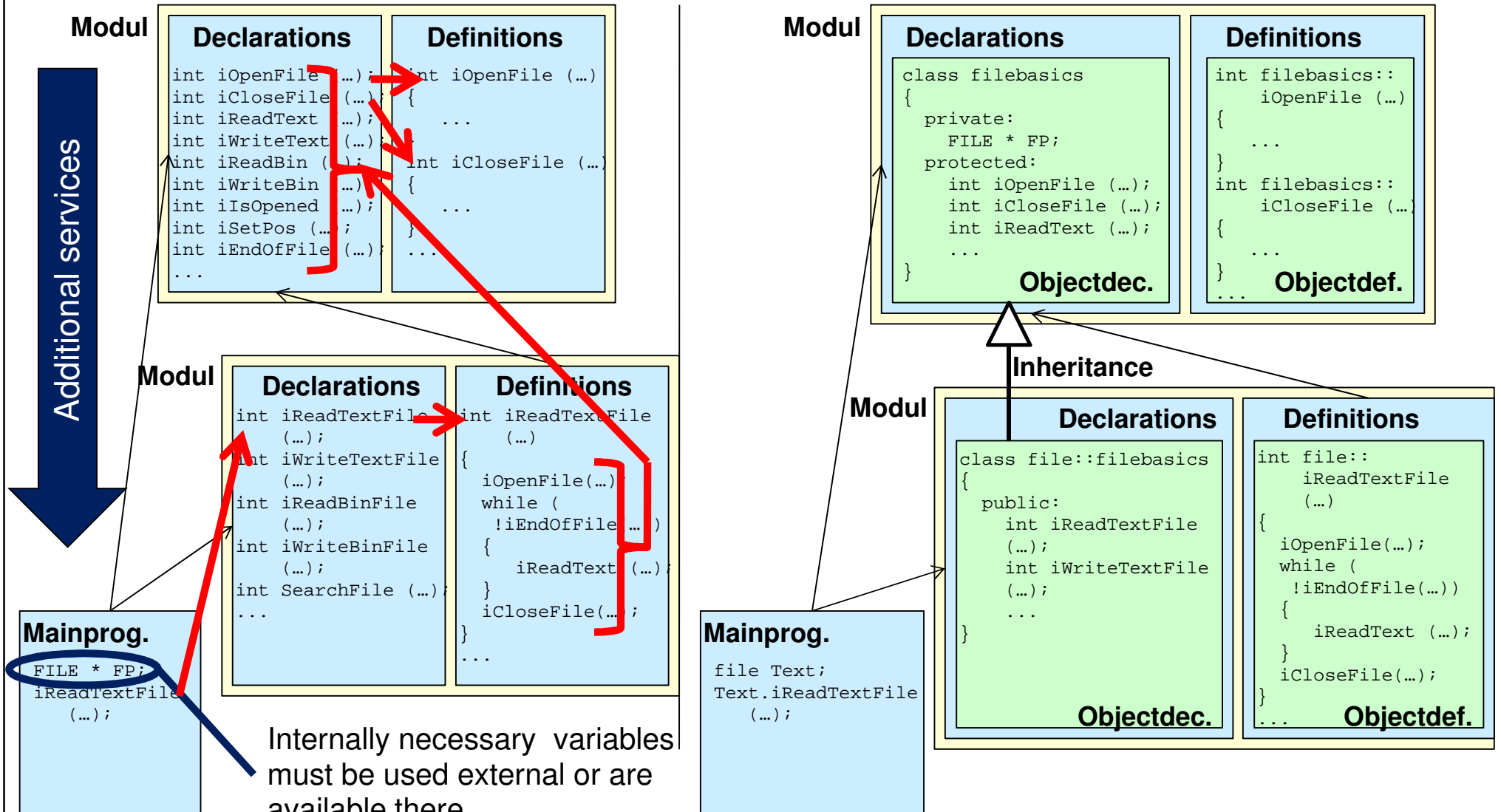
Review: Programming paradigms

Structured programming vs. Object oriented programming



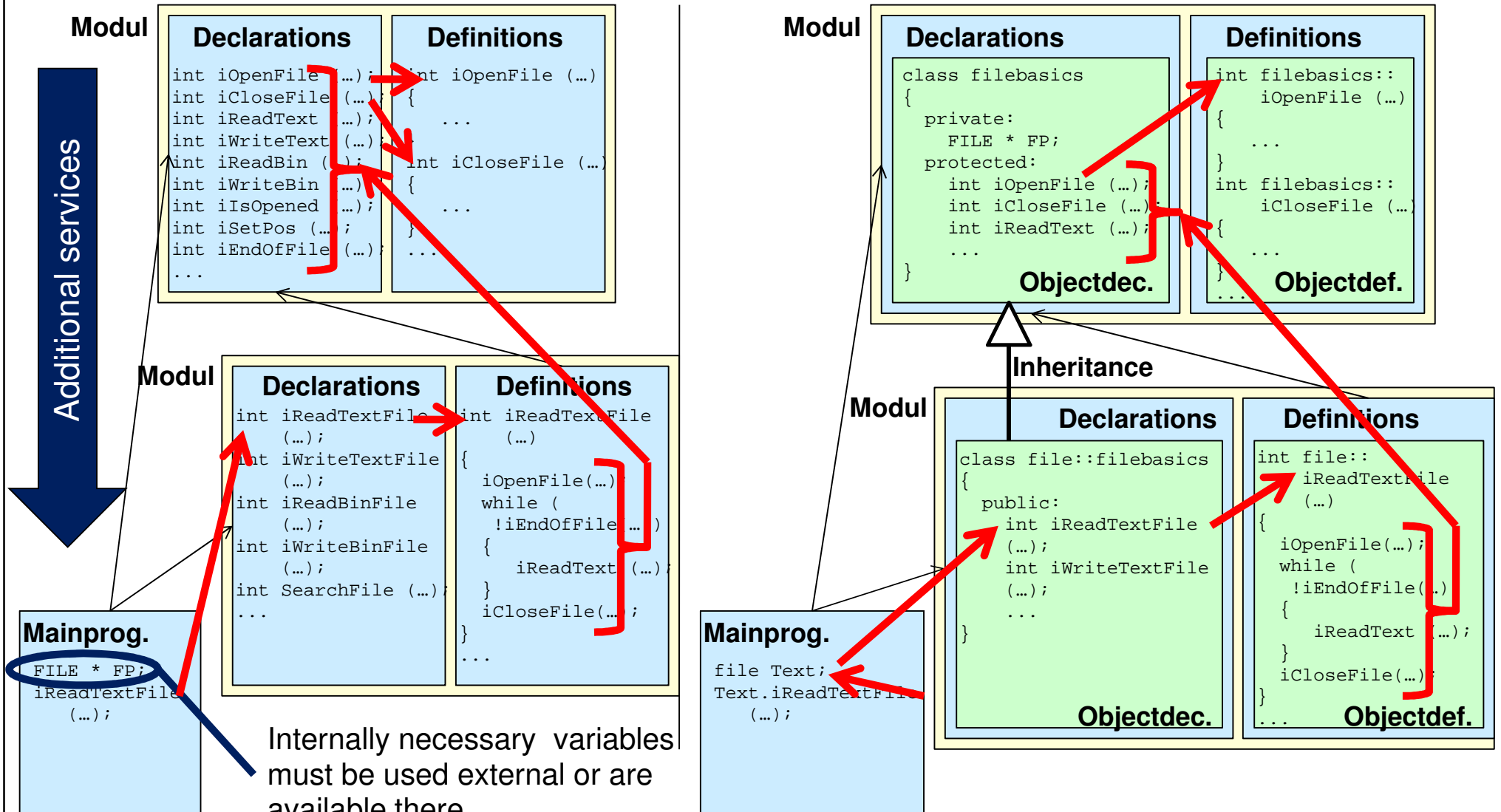
Review: Programming paradigms

Structured programming vs. Object oriented programming



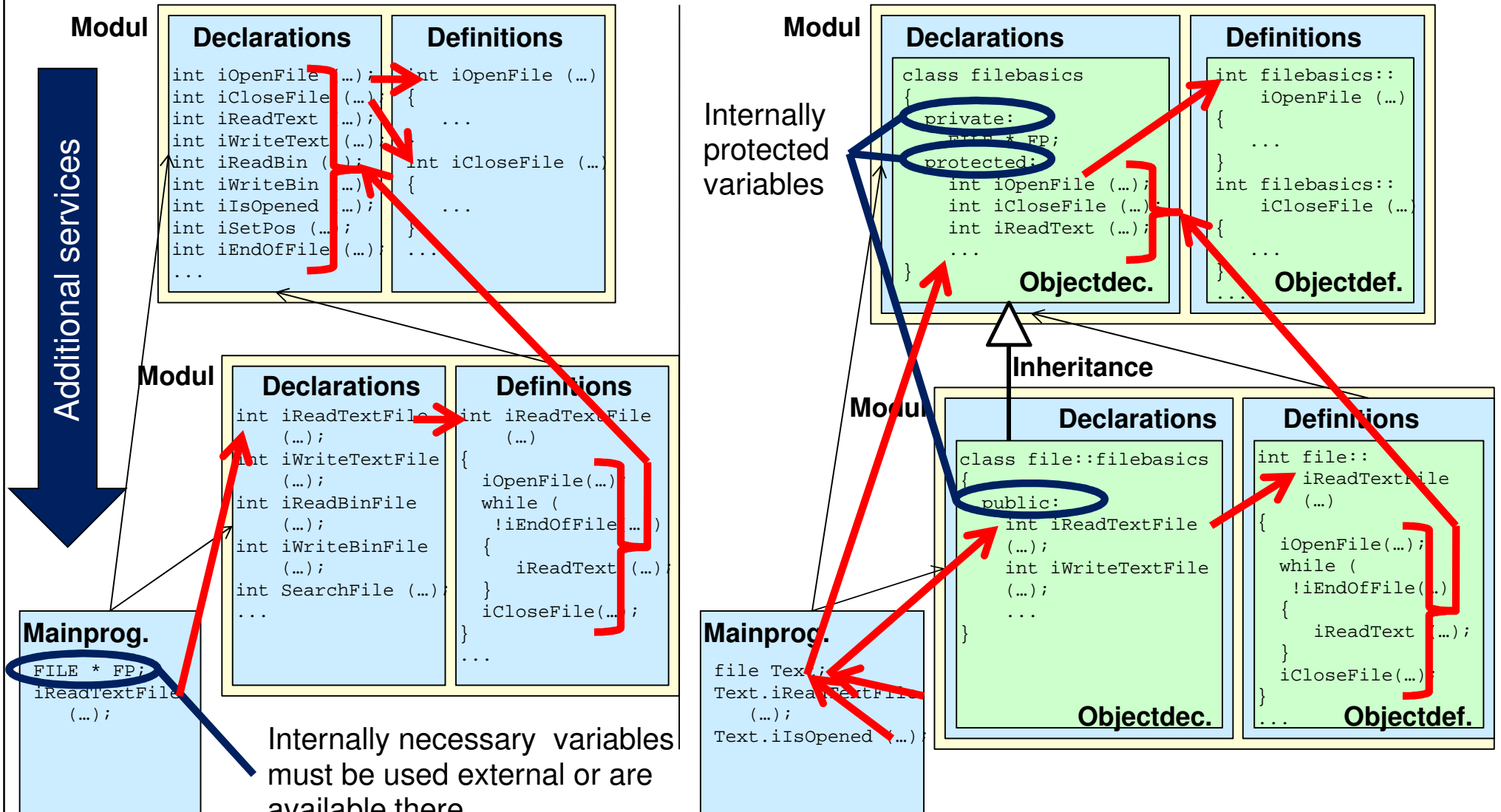
Review: Programming paradigms

Structured programming vs. Object oriented programming



Review: Programming paradigms

Structured programming vs. Object oriented programming

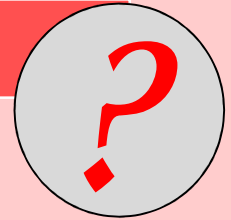


Lecture 3: Software languages and development basics

- ✓ Review: programming paradigms
 - Comparing software languages (C, C++ and Python)
 - Software development method
 - Software development process



Review: programming



Which phases are parts of a compilation workflow (draw the graph)?

What is the difference between compiler and interpreter (give the phases in a short scheme)? What about runtime efficiency of compiler and interpreter code?

Comparing software languages

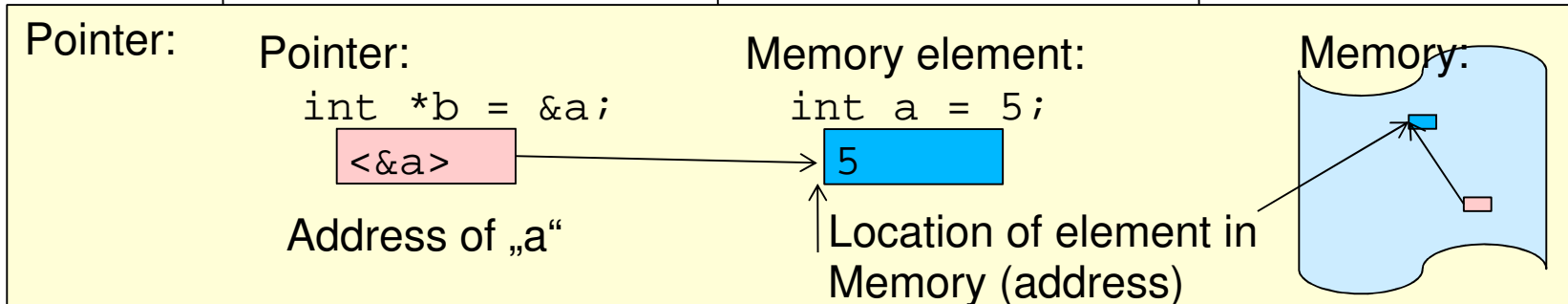
	C	C++	Python**
Translation	Compiler	Compiler	Interpreter, (Compiler)
Programming paradigm	Structured programming	Object oriented programming (Structured prog.)	Object oriented programming (Structured prog.)
History*	B ↓ 1971 C 1978 K&R C 1989 ANSI C	C, Simula67, ... ↓ 1980 C with classes 1983 C++ 1998 ANSI/ISO C++	Modula3, ABC, ANSI C, ↓ ... 1991 Python ... (following versions) 2008 Python 3.0
Basic Structures	Files as modules functions blocks (curly braces)	Files as modules namespaces classes methodes blocks (curly braces)	Files as modules, (packages), classes methodes blocks (indents)

*See: <http://www.levenez.com/lang/>, Download 14.12.2008

**See: <http://wiki.python.org/moin/FrontPage>, Download 14.12.2008

Comparing software languages

	C	C++	Python
Memory Elements	short, int, long, float, double, char, structures, FILE, ... Variables Arrays (e.g. int Ar[5];) Indexes start with 0!!! Pointers	(like C) class, boolean, Standard Template Library: string, list, ... Variables, Objects, Arrays (e.g. int Ar[5];) Indexes starts with 0!!! Pointers	Dynamic typing (integer, float, long integer, octal integer, hexadecimal integer, complex, character string, list, dictionary, tuples, file) Variables, Objects Ar = zeros([2,2], Float) Indexes starts with 0!!!



Comparing software languages

	C	C++	Python
Operators	<u>Standard operator:</u>		
Assign	=,	(like C)	=,
Plus	+,		+,
Minus	-,		-,
Multiplication	*,		*, (power of **)
Division	/,		/,
Modulo-Div.	%,		// (int div.), % (rest),
AND	&& (Bit-AND &)		and,
OR	(Bit-OR)		or,
NOT	!		not, ...
	...		
	<u>Additional operators:</u>		
	Add one, subtract one:		Member of set:
	++,--, (e.g. i++; is i=i+1)		not in, ...,
	and a lot of others:		and a lot of others:
	+=, -=, ..., [,], ->, *, ...		+=, -=, ...

Comparing software languages

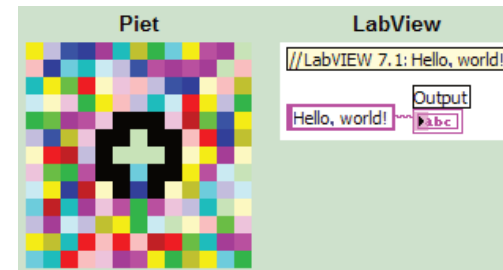
	C	C++	Python
Application workflow	<p>Conditions</p> <pre>if (iIndex < 10) { printf ("A"); } else { printf ("B"); } </pre> <p>(and also „switch/case“)</p> <p>Loops</p> <pre>int i = 0; while (i < 10) { printf ("%d", i); } </pre> <hr/> <pre>int i = 0; do { printf ("%d", i); } while (i < 10) </pre> <hr/> <pre>int i; for (i = 1; i < 10; i++) { printf ("%d", i); } </pre>	<p>Conditions</p> <pre>if (iIndex < 10) { std::cout << "A"; } else { std::cout << "B"; } </pre> <p>(and also „switch/case“)</p> <p>Loops</p> <pre>int i = 0; while (i < 10) { std::cout << i; } </pre> <hr/> <pre>int i = 0; do { std::cout << i; } while (i < 10) </pre> <hr/> <pre>for (int i = 1; i < 10; i++) { std::cout << i; } </pre>	<p>Conditions</p> <pre>if iIndex < 10: print ("A") elif iIndex == 10: print ("B") else: print ("X") </pre> <p>Loops</p> <pre>i = 0 while i < 10: print (i) </pre> <hr/> <pre>for i in range(1,10): print (i) </pre>

Comparing software languages

	C	C++	Python
Flexibility	Very flexible Low-level prog. Easy to learn Fast, optimized run Portable „Closed“ code Compiler platform specific (open and licensed versions)	(like C) Plus better structuring Difficult to learn Sometimes slower than C Easy for project management Compiler platform specific (open and licensed versions)	Very flexible Indents often ends in errors Easy to learn, difficult to use complete possibilities Slow Very portable Open code
Best for	All kinds of programs from hardware, operating systems to applications (design rules are necessary)	(like C) But in most cases for higher level programming (games, office, telecommunication, ...)	University tasks Scripts Prototyping Web applications

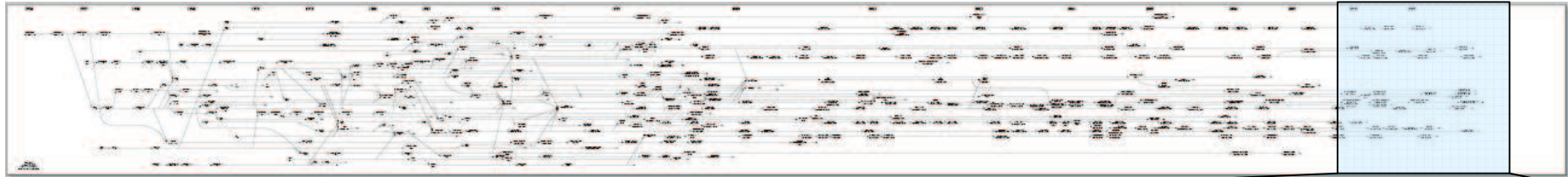
Comparing software languages

The image shows a grid of code snippets for 'Hello, world!' in various programming languages. The languages listed include: ABAP, Ada, ALGOL, AmigaScript, ANL, APL, AppleScript, ASP, Assembler, awk, B, Bash, BASIC, Brainfuck, BS2000, C, C#, C++, COBOL, Common Lisp, dBase, Dylan, ECMAScript, Erlang, Fortran, GIMP, Haskell, Hello, HPLC, HTML, INTERCAL, Java, JavaScript, JSP, LISP, Logo, Machinecode, MARK, Modula-2, MS-DOS Batch, Mumps-M, Oberon, Pascal, Perl, PHP, PL/I, Plinkaku, PostScript, Powershell, Prolog, Python, REXX, Ruby, Scheme, sed, SOF, Simula, Smalltalk, SQL, TeX, Ironpython, Velocity, Whitespace, XML, XUL, and xppsum.



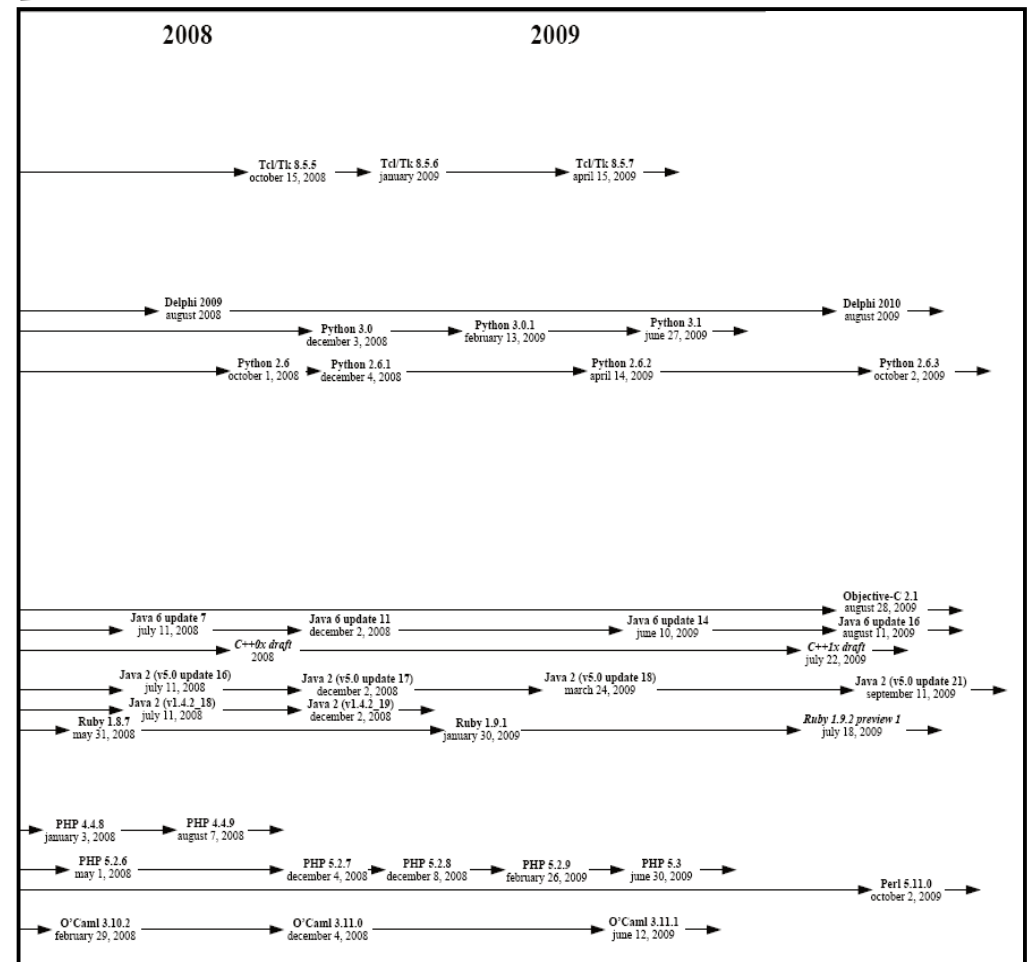
See: <http://www.lehmanns.de/helloworld>,
Download 14.12.2008

Comparing software languages

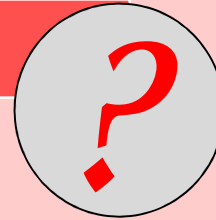


See: Lévénéz, Éric ;
<http://www.levenez.com/lang/>,
 Download 18.10.2009

Or for an overview for other about 2500
 computer languages see:
 Kinnersley, Bill: The language list.
<http://people.ku.edu/~nkinners/LangList/Extras/langlist.htm>,
 Download 18.10.2009



Comparing software languages



*What about
Matlab?*

Translation

Programming paradigm

Structure

Memory elements

Types

Operators

Application workflow

Flexibility/Best for



Comparing software languages



The presentations about different computer languages showed us the different for-loop styles in the different languages. Convert the following C for-loop into a Matlab equivalent one (keep in mind the different indexing!).

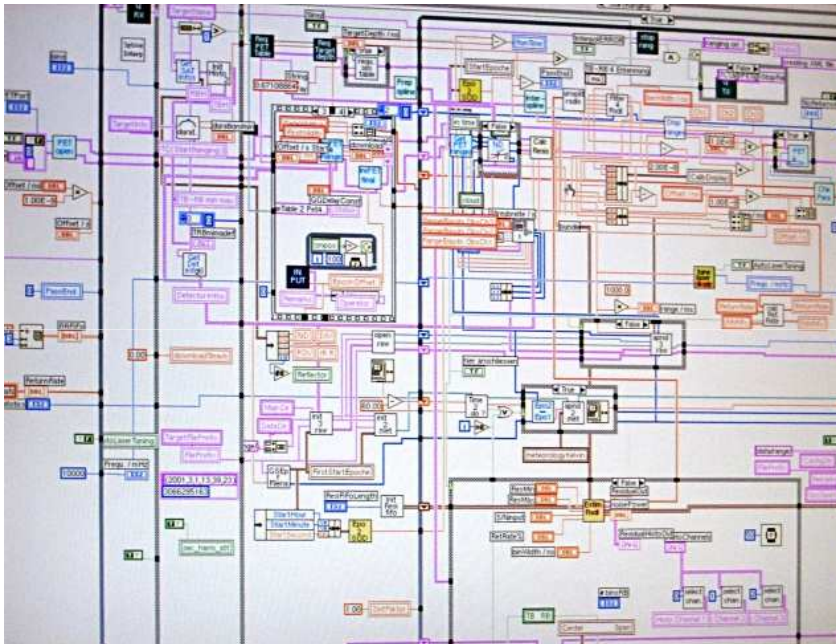
```
int i,j;
int A[10][10];
for (i=0; i<10; i++)
{
    for (j=0; j<10; j++)
    {
        A[i][j] = i*j;
    }
}
```

Lecture 3: Software languages and development basics

- ✓ Review: programming paradigms
- ✓ Comparing software languages (C, C++ and Python)
 - Software development method
 - Software development process

Software methodology: Why ... ?

Often software looks like this ...



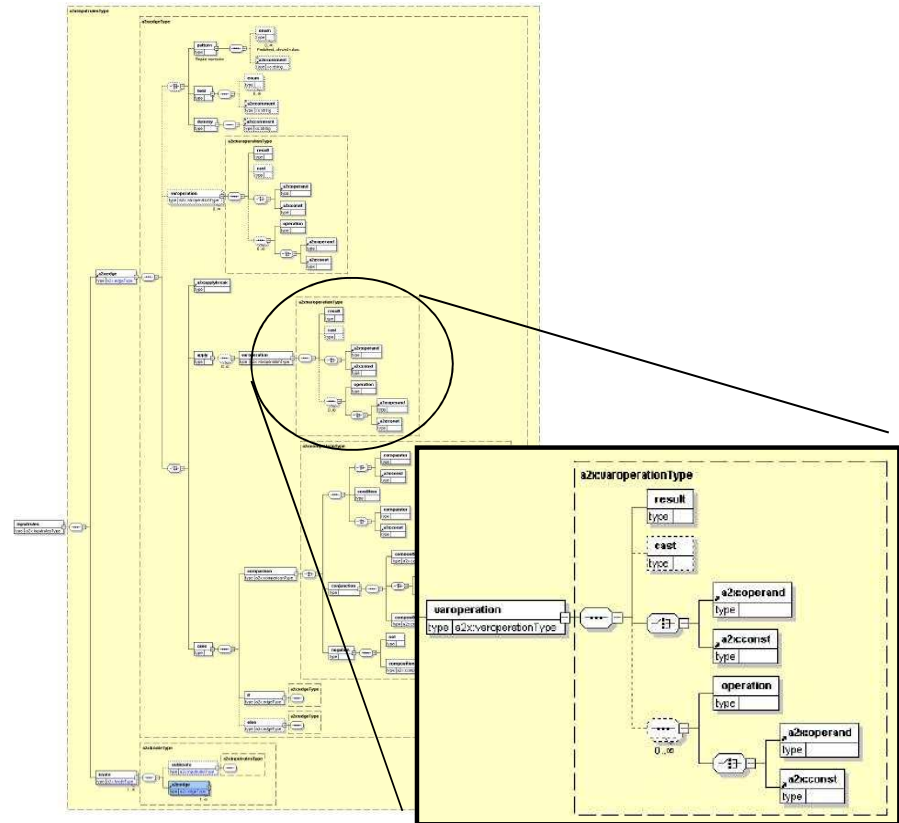
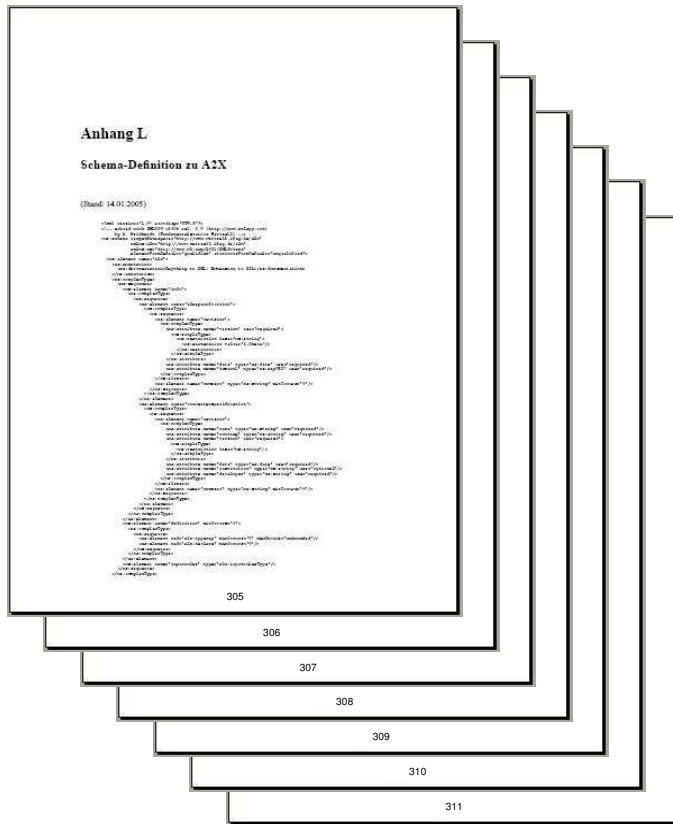
... and this look like that!




Complex structures have a tendency to become intransparent and unstructured

Software methodology: Why ... ?

It's not so easy to understand this ...



... but it's easier to understand that!

 **The human brain handles graphical content impressions better than listed details**

Software methodology: Why ... ?

Chaos Study:

- Only 16 % of all software projects are successful -> this means 84% doesn't match with the expectations
- 53% of all software projects last too long and cost much more than planned, because the planning phase was too inefficient
- 31% of all software projects never ends, because of a deficiency at the planning, in aims and team work

Chaos-Study of the Standish Group

(Reference: <http://gondi-online.de/index.php/Projekt>, Download 08.12.2006)



Software development is complex and it is not easy to guarantee software quality

Software methodology: quality factors

And the quality factors are:

- Reliability/Robustness
- Portability/Scalability
- Usability/Functionality/Correctness
- Maintainability
- Compactness
- Re-usability/Modularity
- Comprehensibility/Understandability
- Schedulability/Efficiency/Flexibility
- Testability
- Security ...

Software methodology

There are a lot of techniques which can help to improve the development of software

Software methodology :=

software development
process

+

process attending modelling
techniques

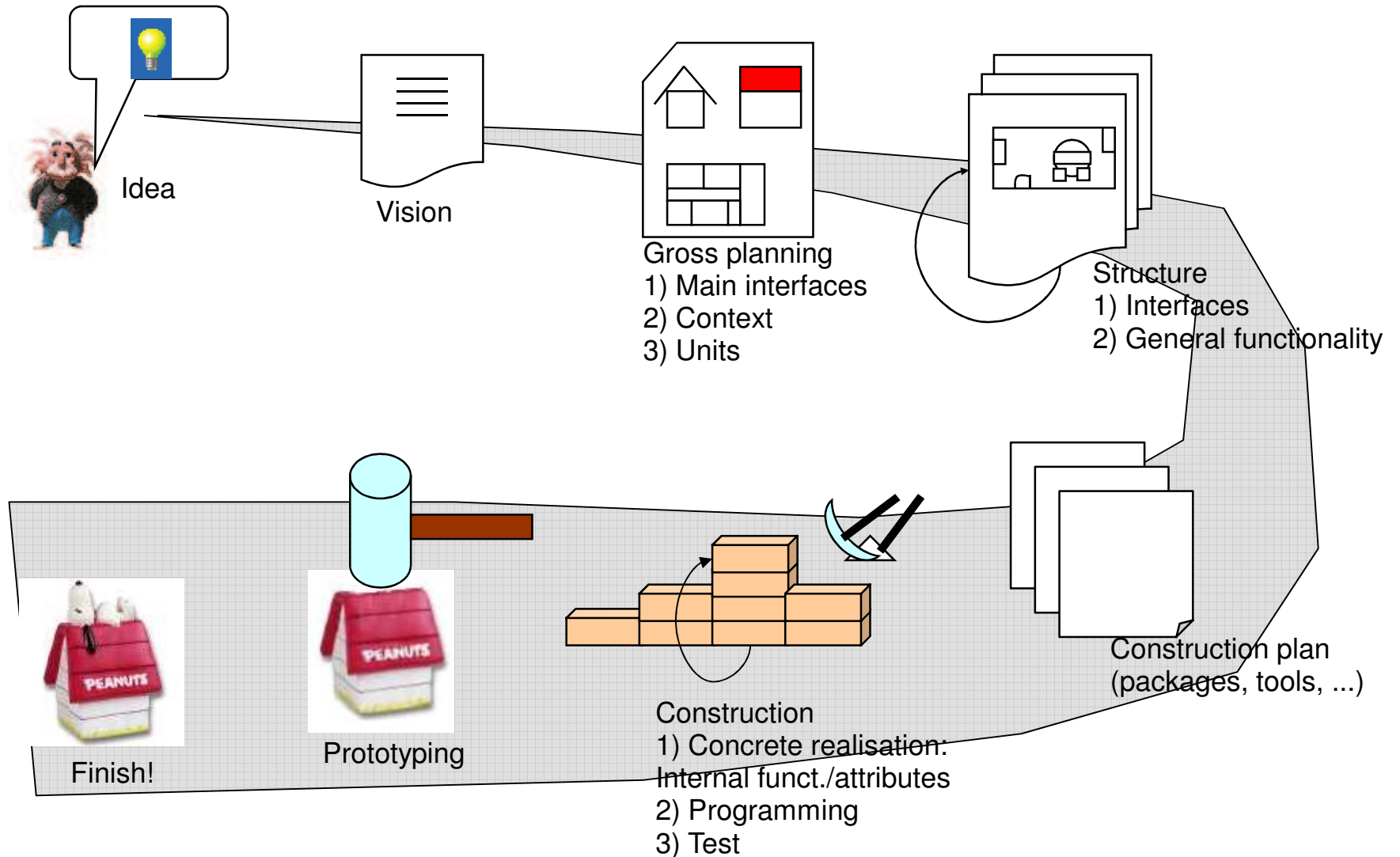
+

(process attending design/
programming rules)

Lecture 3: Software languages and development basics

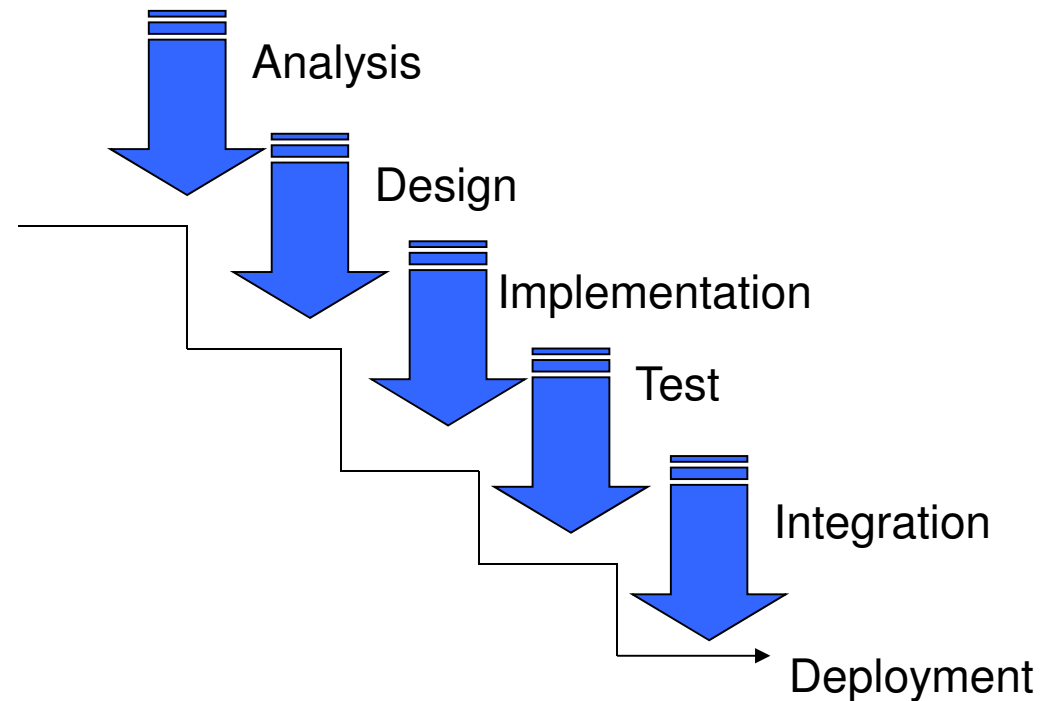
- ✓ Review: programming paradigms
- ✓ Comparing software languages (C, C++ and Python)
- ✓ Software development method
 - Software development process

Software development process



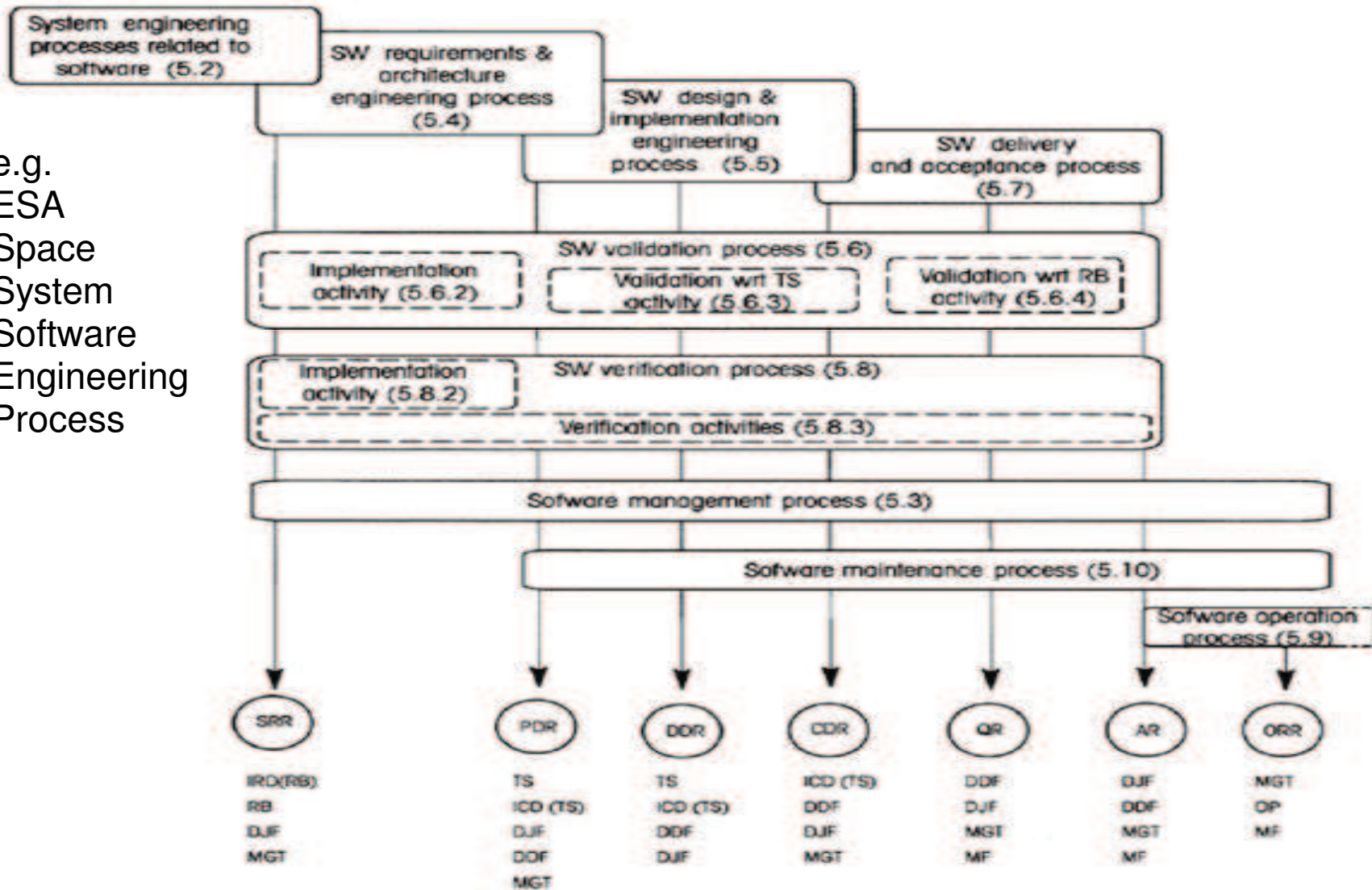
Software development process: sequential

Sequential process: the top-down or water fall model



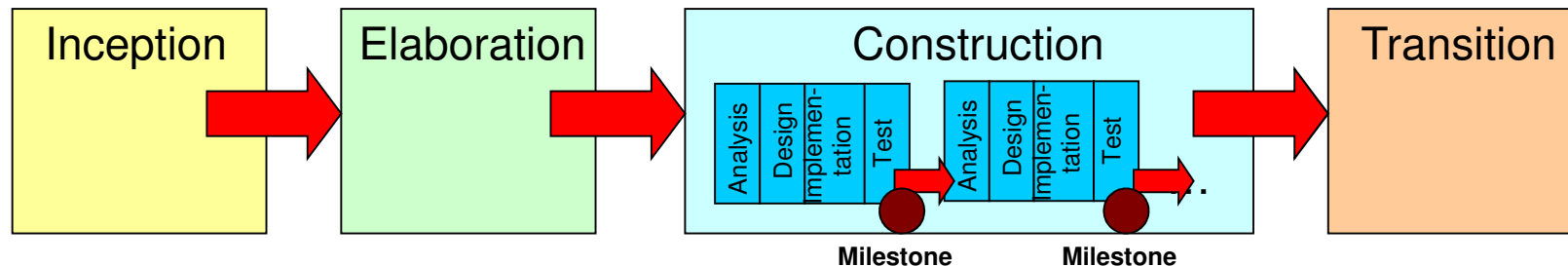
Software development process

e.g.
ESA
Space
System
Software
Engineering
Process



Software development process: iterative

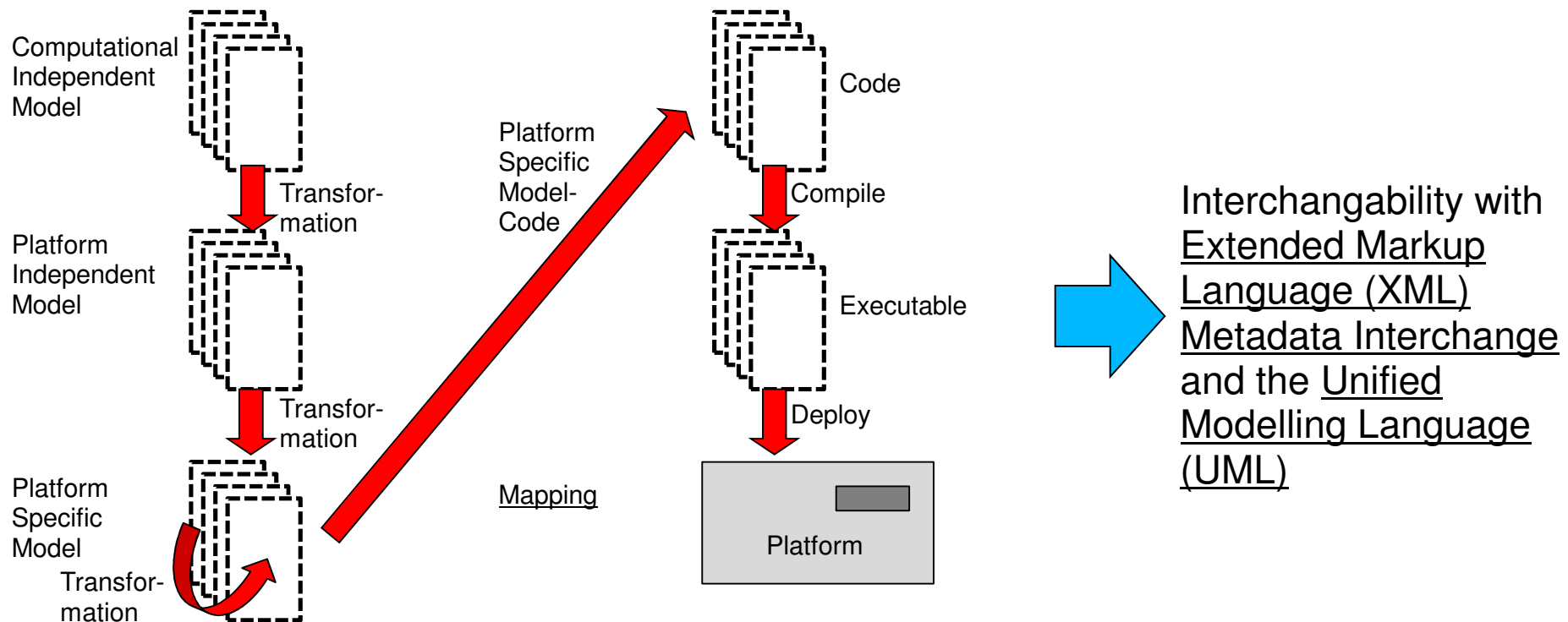
An iterative and incremental way to solve problems ...



- An iterative and incremental way to solve problems ...
- **Inception** defines the framework for the project (Feasibility study or „small talk“, analysing the complexity)
- **Elaboration** defines the detailed requirements, an upper layer design, the architecture and a construction plan (risk analysis, use cases to plan user interactions, time plan and milestones, domain model with terminology/workflows/requirements)
- The software is then developed as partially during the **construction phases** and not as one complete block (analyse small moduls, design, implement and test them as reduced managable „mini“-projects)
- At the **transition phase** finalizing activities guide into future software usage
- The complexity is given by formal ceremonials

Software development process: iterative

The Model Driven Architecture (MDA): the avant-garde
 Definition of so-called development patterns, they describe how to transform a general meta-model in a fitting model
 Goal: portability, interoperability, reusability
 => Development process which is platform-independent, design for a specific platform, allow a transformation with dedicated rules from the independent model into the real architecture (ideal case: with a generative tool)

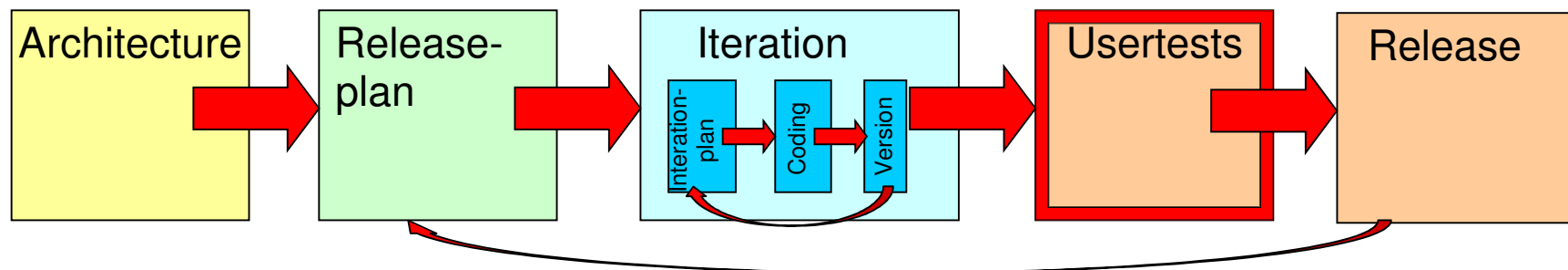


Software development process: agile

Lightweight process, eXtreme Programming

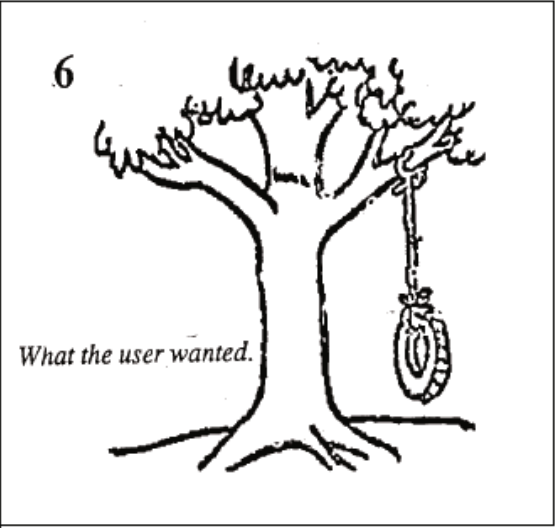
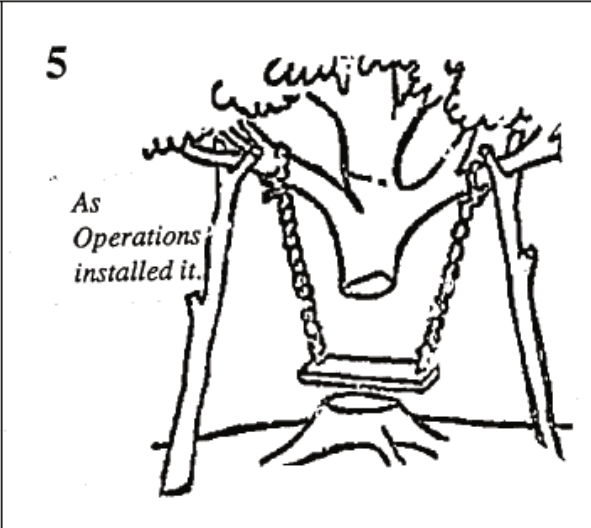
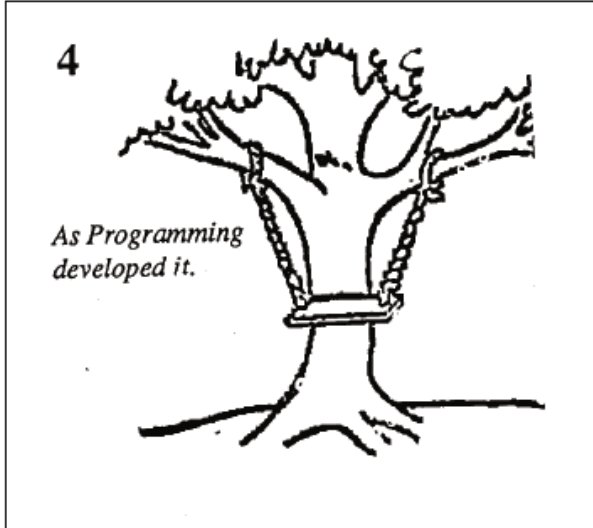
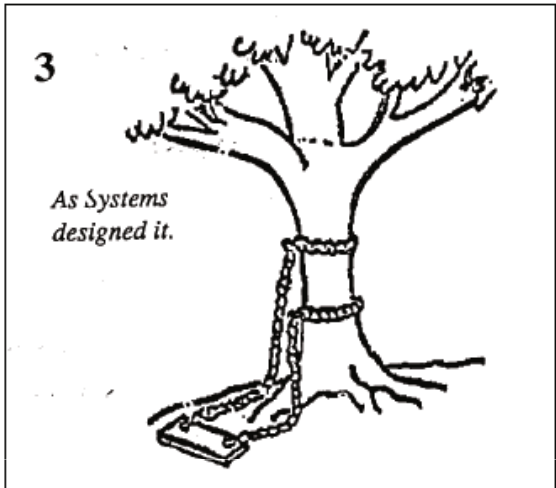
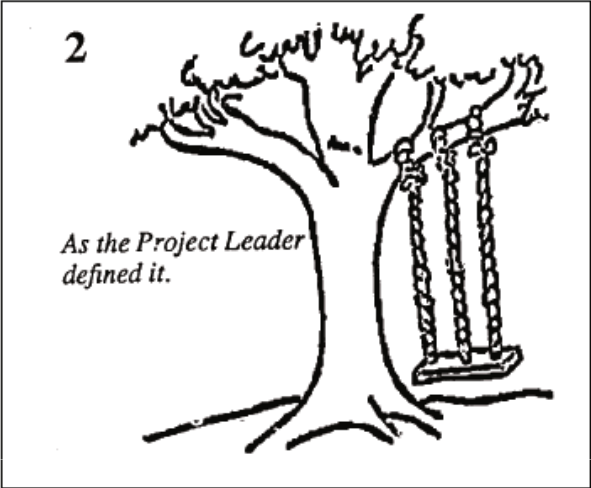
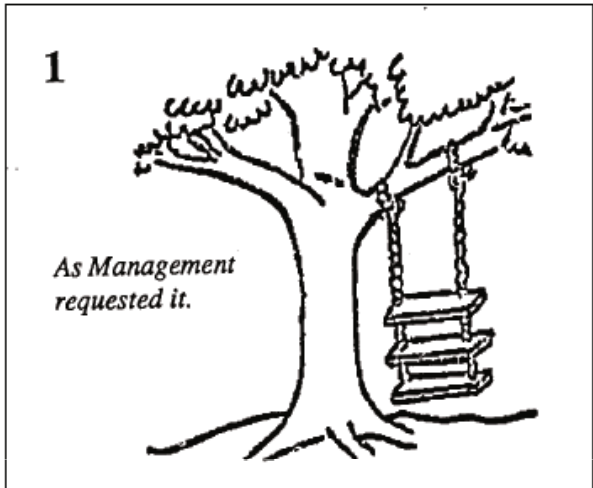
Ideas:

- The future can't be planned
- No classic, structured analyses with a document based process model
- Reduce believing in tools and strengthen the individual ability of each software developer, who is responsible for his own part, to solve complex problems
- Reduce outsourcing
- Selfregulated, selforganizing systems
- Management as moderating instance and not for controlling
- Success is based on people and their communication between each other and not on processes and technique (see development of Linux)
- Most developments specifications are not complete



=> Communication, Testing, Feedback, Simplicity

Software development process: is not easy ...



Thank you