Recap & Trivia

COMP 524: Programming Language Concepts
Björn B. Brandenburg

The University of North Carolina at Chapel Hill

What are the two main language design paradigms?

Declarative and Imperative.

List the three major sub-categories of declarative languages.

Name an language in each category.

Functional programming. E.g., Haskell.
Logic/constrained-based programming. E.g., Prolog.
Dataflow. E.g., Id, Val.

What are the key characteristics of procedural languages?

Sequential computation.
Direct data manipulation.
Problem decomposition via subroutines.

What is the design metaphor of object-orientation?

Inspired by human organization.

Communicating experts.
Problem decomposition by delegation to “domain specialists”.
Objects send and respond to messages.

Problem is solved by “team” of “collaborating” “workers.”

"XXX is a language so far ahead of its time, that it was not only an improvement on its predecessors, but also on nearly all its successors."

– C. A. R. Hoare

Which language is XXX?

Algol 60.

Name three contributing factors to the success of object orientation.

Easy-to-understand, human-friendly model.
Simplifies code reuse: great libraries.
With careful design, object-oriented languages can be compiled into very efficient code.

How do (pure) functional languages differ from imperative languages?

No concept of memory and object identity; only values and equality.
No side effects: functions only map input values to output values.
No global state; esp. no time.
No explicit loops.

Why is there an increasing interest in declarative languages?

Concurrency. Declarative languages lend themselves (in theory) to auto-parallelization (= the compiler figures out how to do things in parallel).
Imperative languages tend to over-specify the amount of sequentiality required.

Name a traditional application of scripting languages.

Administration work.
- Job control. (mainframes)
- Shell. (Unix)

What are the three primary use cases of high-level programming languages?

Being read by a human. (Maintenance!)
Being used to specify algorithms.
Being used to write new / extend existing applications.

What's the purpose of the `invariant`, `require`, and `ensure` keywords in Eiffel?

`invariant`: class invariant; some properties of the local state that all objects must satisfy at all times.
`require`: method pre-condition; constraints that must hold for a method succeed
`ensure`: method post-condition; constraints that must hold after a method returned.

"It is practically impossible to teach good programming to students that have had a prior exposure to XXX: as potential programmers they are mentally mutilated beyond hope of regeneration."

Said who about what?

Dijkstra on Basic.

What greatly contributed to the success of Pascal?

Niklaus Wirth developed an easy-to-port, high-quality implementation and gave it away for free.

What's the problem with languages that have only a single vendor?

If the vendor goes out of business, then your programs may become obsolete for lack of a modern compiler.

What's the DRY principle?
Why is it propagated?

Don't repeat yourself.
Duplicating information causes (boring) work and creates the danger of being of of sync.

Why don't the Linux developers simply enable all warnings that gcc has to offer?

False positives. They would get flooded with meaningless warnings.

What's the benefit of "linking" processes in Erlang?

Avoids half-dead systems; enables clean restart.

Name a typical security issue caused by the lack of array bounds checking in C.

Name a typical security issue caused by the lack of integer overflow checking in C.

array bound checking: buffer overflow + stack smashing.
Integer overflow: heap-based buffer overflow (allocation of too-small buffers), followed by tricky ways to take over control flow.

"There is another argument which is all too prevalent among enthusiastic language designers, that efficiency of object code is no longer important; that the speed and-capacity of computers is increasing and their price is coming down, and the programming language designer might as well take advantage of this. This is an argument that would be quite acceptable if used to justify an efficiency loss of ten or twenty percent,or even thirty and forty percent. But all too frequently it is used to justify an efficiency loss of a factor of two, or ten, or even more; and worse, the overhead is not only in time taken but in space occupied by the running program."

From which year is this criticism? Do you know who voiced it?

1973! And still relevant… C.A.R.Hoare