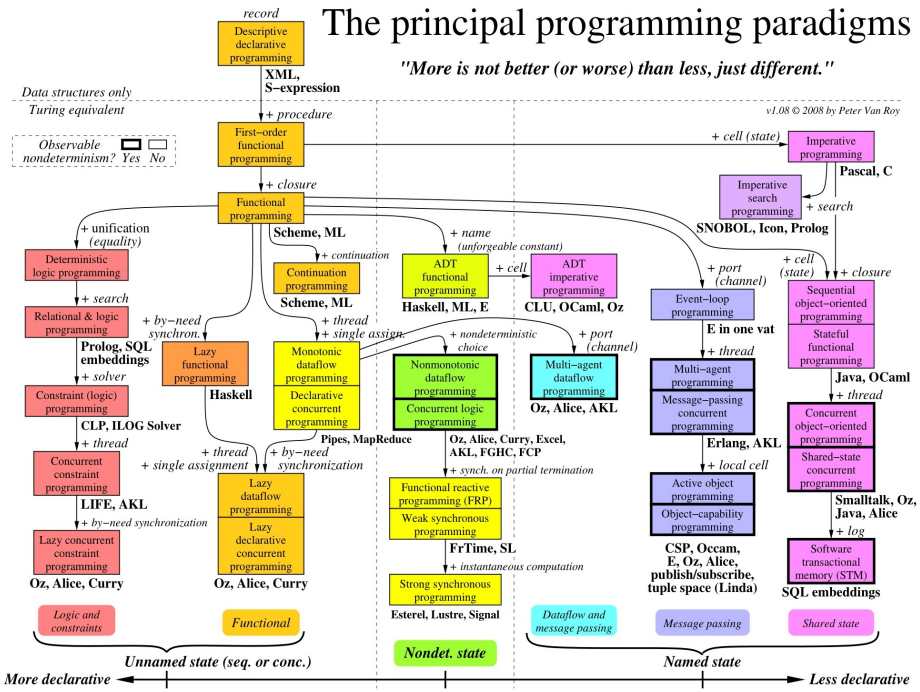
TDDA69 Data and Program Structure Summary

Cyrille Berger

LINKÖPING UNIVERSITY



Lectures

Lecture content

- Summary
  - Choosing an appropriate Programming Language Paradigm
    - The different types of interpreter
- GUI Programming

Summary

Do we need new programming languages?

- New Concepts
  - in the early days, object orientation
- New problems and new infrastructure
  - Multi-threading
  - Distributed computing
  - ...
- Develop a better syntax

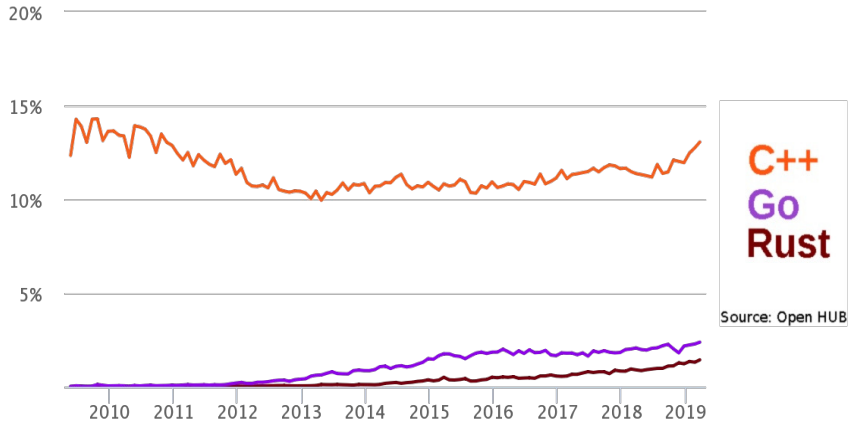
Motivation for creating Go

Rob Pike, Go creator: *"A couple of years ago, several of us at Google became a little frustrated with the software development process, and particularly using C++ to write large server software. We found that the **binaries tended to be much too big**. They took **too long to compile**. And the language itself, which is pretty much the main system software language in the world right now, is **a very old language**. A lot of the **ideas and changes in hardware** that have come about in the last couple of decades haven't had **a chance to influence C++**."*

Motivation for creating Rust

Graydon Hoare, Rust creator: *"A lot of **obvious good ideas**, known and loved in other languages, **haven't made it** into widely-used systems languages, or are deployed in languages that have very poor (unsafe, concurrency-hostile) memory models. There were a lot of good competitors in the late 70s and early 80s in that space, and I wanted to **revive some of their ideas** and give them another go, on the theory that circumstances have changed: the internet is highly concurrent and highly security-conscious, **so the design-tradeoffs that always favor C and C++ (for example) have been shifting**."*

Usage of C++ vs Go vs Rust



Is it easier to change and fix existing languages?

- Backward-compatible changes
- Backward-incompatible changes
  - Some changes are too difficult
  - Introducing Unicode in Python and PHP
  - Garbage collector in C++
- Those changes introduce long development time and long acceptance time
  - Python 3.0 was introduced in 2008
  - PHP 6 was started in 2006 and never released, PHP 7 released in December 2015

PHP Criticism

- PHP was not designed, but developed
  - Inconsistent naming of functions and order of their parameters
    Some function names were chosen to improve the distribution of hash values
  - Rather than aborting with an error, PHP will try to guess the developer intent
    Problems with weak typing
  - PHP compilation options, server configurations, applications configurations and global states can affect function behaviour
  - Incoherent mix between functional and object-oriented programming
  - ...
- You need a vision and a design when developing a programming language!

Design Considerations for a Programming Language

A programming language must be:

- **predictable**
  - Source code is read more often than written, a human must be able to understand what he read
- **consistent**
  - Knowing part of a language should help learn other parts
- **concise**, **simple** and **general**
- **reliable**
  - Programming language are here to solve problem, not to introduce new one
- **debuggable**
  - Developers will inevability write *bugs*, they need all the help they can get to find them
- **implementable**
  - This reduce the number of bugs in the language implementation

What is the purpose of the new language?

- First question is, a new language, what for?
  - Querying knowledge?
  - Distributed numerical computation?
  - Writting drivers for an Operating System?
  - Writting web applications?
  - Answering the Ultimate Question of Life, the Universe, and Everything
  - A programming language for teaching about interpreters and programming models/paradigms
  - ...

Design choices for a Programming Language

- Programming Paradigm
- Dynamic vs Static (Typing...)
- Low-level vs High-level
- Direct interpretation, Virtual Machine, JIT, Native Compilation...

Choosing an appropriate Programming Language Paradigm

Declarative

- Expresses logic of computation without control flow:
  - What should be computed and not how it should be computed.
- Examples: XML/HTML, antlr4/yacc, make/ants...

Functional

- Computation are treated as mathematical function
  - without changing any internal state
- Examples: Lisp, Scheme, Haskell...

Logic Programming

- Based on Formal logic: expressing facts and rules
- Examples: Prolog

Imperative

- Express how computation are executed
  - Describes computation in term of statements that change the internal state
- Examples: C/C++, Pascal, Java, Python, JavaScript...

Object-Oriented

- Based on the concept of *objects*, which are *data structures* containing *fields* and *methods*
  - Programs are designed by making objects interact with each others
- Examples: C++, Java, C#, Python, Ruby, JavaScript...

How to choose a programming paradigm?

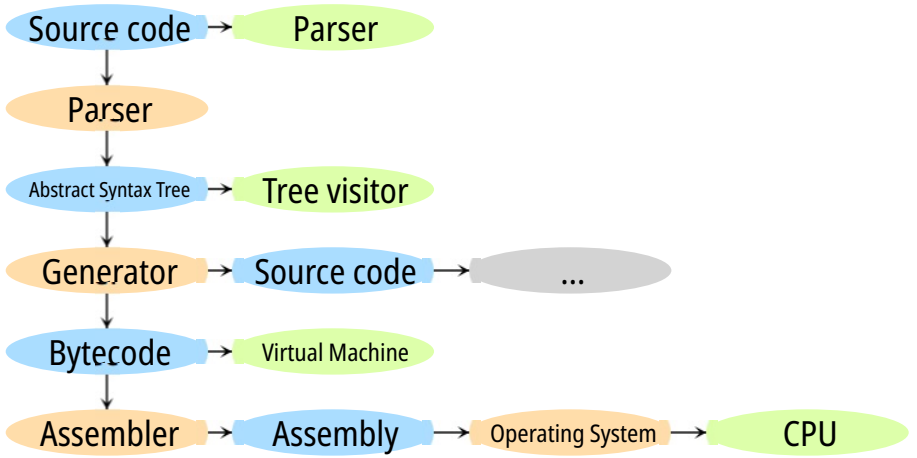
- The choice depends on the goal
  - Querying knowledge?
    - ⇒ Declarative or logic (reasoning)
  - Distributed numerical computation?
    - ⇒ Functional
  - Writting drivers for an Operating System?
    - ⇒ Imperative
  - Writting web applications?
    - ⇒ Object-Oriented
- The choice can be considered controversial!

Conscequences of the choice

- Conscequence for the users
  - Expressivity, simplicity, readability
- Conscequence on the implementation
  - Different types of interpreters

The different types of interpreter

How is a program interpreted?



Interpreting Declarative and Logic

- Using a query executor (for SQL)
- Using a reasoning engine and unification (for Prolog)

Interpreting Functional

- Functions are evaluated in order
- Applicative vs Normal
- In normal order, the tree needs to be expanded, virtual machines are more difficult (might be impossible)

Interpreting Imperative / Object-oriented

- Directly from the abstract syntax tree
- Using virtual machines
- Running on the CPU
- The most versatile, the most common, the most studied

GUI Programming

GUI Programming

- What model?
- Purely procedural
  - Sequence of graphical command
  - Example: AWT, SWING...

```
drawRectangle(0, 5, 10, 20)
```

- Purely declarative
  - Choosen from a set of possibilities
  - Example: HTML...

```
<rectangle geometry='0,5,10,20' />
```


Procedural approach

- Set of primitive operations combined in a program
- Unlimited expressiveness
- Harder to do formal manipulation (converting data to user interface)

Declarative approach

- Set of possible shapes for different attributes
- Limited expressiveness
- Easy match between data and user interface

Combining declarative/procedural approach

- Examples: QML, React,
- Takes the best of both
- Declarativeness is used
  - Describing static structure of a window
  - The type of widgets
  - The initial state of the widgets
  - The resize behaviour
- Procedure is used
  - Procedures executed when events occurs
  - Handlers

Example

```
Window
{
  width: 150
  height: 50
  Rectangle
  {
    anchors.fill: parent
    color: "red"
    MouseArea
    {
      anchors.fill: parent
      onClicked: Qt.quit()
    }
  }
  Repeater
  {
    model: [a list]
    delegate: Text { text: modelData }
  }
}
```


Conclusion

- Many problems, many solutions, many developers, many paradigms.
- A large program is likely to be a combination of paradigms
- Programming language tend to combine from different paradigms