Galileo: a legacy for builders of new worlds?

Francesco Bruschi
Politecnico di Milano

Experimenting

- Let's call a system "experimentable" if it is possible (or feasible) to concoct experiences:
  - that are **repeatable**
  - **reproducible**
  - whose effects can be **compared**
- Usually**:**
  - experimenting as a mean of getting to know facts, laws, or explanations about a **natural** world **that is given**!
- Most of the time,
  - informatics practice deals with **building new worlds**, or getting to know worlds built by others

(Brave) New Worlds

- Dijkstra: "And this is what a programmer has to do all the time; he has to introduce new concepts --not occurring in the original problem statement-- in order to be able to find, to describe and to understand his own solution to the problem."
- So programmers either:
  - invent new worlds (example: Design Specific Languages, frameworks, libraries)

  or

  - try to understand worlds invented by others (to stand on their (giants'?) shoulders)

A many many worlds universe

Suppose you want to develop a webapp. Which frameworks, DSL and APIs would you rely on?

- Pyramid, Django, TurboGears, Rails, Bottle, Flask, Sinatra, web.go, happstack, yesod etc etc
- Ever more effort is spent "evaluating" which DSL, or framework, or library is most convenient for the problem at hand, and then learning it.

Read! Evaluate! Print! Loop!

- Ever more often, language builders propose easily accessible REPL environments
    - golang.org, haskell.org, repl.it
- Experimenting is encouraged for evaluation and learning: it has a **cognitive, didactic** value!
- For most of these worlds there is a "*grandissimo libro*" of Nature, but it's unfeasible to read them all...
- But: are all languages equally **experimentable**?

Experimentable languages

- Recent remarkable reinassance of functional languages (Clojure, Haskell, Scala, etc)
- Can calling an API function be an experiment?
- Example:
  - a C function:

```
int f(int *n)
```

Not replicable: you should know the memory state!
Not repeatable: it could access files
Results not comparable: it could have side-effects (IO, etc)

  - a Haskell function:

```
f :: Int -> Int
```

Replicable & Repeatable: result guaranteed to be the same when explicit parameters don't change
Results comparable: the signature guarantees there's no side effect

Galileo's message to Computer Science?

- Can "Experimentability" be considered a goal in the definition of new formalisms? (Can Galileo be prescriptive (at least for SEs)?)
- Is the renewed interest for purely functional languages an invite to "experiment more" in the evaluation, learning and mastering of new formalisms?
- In CS curricula, should we highlight *the possibility* of this approach to programming languages?

From natural to "social" science

Mental model of a working machine

- in the 80s: mostly based on natural phenomena (electronics, mechanics, etc)
- since then: **many** layers of abstraction added between the machine and the user
- nowadays: the reasons of the features and workings of a system are much more related to **linguistics**, **psychology**, **history**

Example: choose a web-development framework

[insert list of frameworks]

- how do you choose?
- making a mental model studying documentation => not feasible (too many options, poor "formal", "complete" documentation)
- **tutorials** are the new learning tool
- learning "by experimenting"?

An engineering perspective

informatics engineering:

- a very wide range of different frameworks for performing a given task
- understanding of the workings of a framework passes throught experimentation (rich tutorials and poor documentation, need to evaluate many different options)
- a renewed interest for functional languages (eg: haskell)

Functional Languages

- In the most active areas of software engineering, renewed interest towards functional languages (the purer, the better)
- remarkable example: haskell (very pure...)
- features:
  - the value of a function only depends on its input:
    - a function will *always* return the same results:
      - *whenever* invoke (repeatability)
      - in whatever environment it is invoked (reproducibility)
  - it is always explicit whether a function has side-effects (comparisons of results)

"non-experimentable" function

```
   FILE f=fopen("data","r");
   scanf("%i",&i); a+=i;
   printf("%i",a);
   return a; }
f(10);
```

execution of f:

- not repeatable (different calls with same input can give rise to different output)
- not reproducible (depens on content of file "data")
- results not comparable (we have side effects!)

"experimentable" function

```
int f(int* a)
f:: int -> int
```

this simple signature guartantees that

1) whenever the function will be called, it will always produce the same result (repeatability)

2) given the same inputs, it produces the same output (reproducibility)

3) it has no side effects (all its "results" are known, and so can be compared)

- Functional languages try to "maximize" the experimentable area of a given framework
- and: they push towards "isolating" all the non-idealities that prevents a system from being "experimentable" (that is, knowable from experiments)
- Galileo here doesn't teach us how to know a given world, but **prescribes us how to build new ones**!!