



Software Development *The UNIX Philosophy*

Davide Balzarotti

Eurecom – Sophia Antipolis, France

In a Nutshell

This is the UNIX philosophy:

- Write programs that do **one thing** and do **it well**.
- Write programs to **work together**.
- Write programs to handle **text streams**, because that is a universal interface.

-- Doug McIlroy

A Philosophy for Programmers

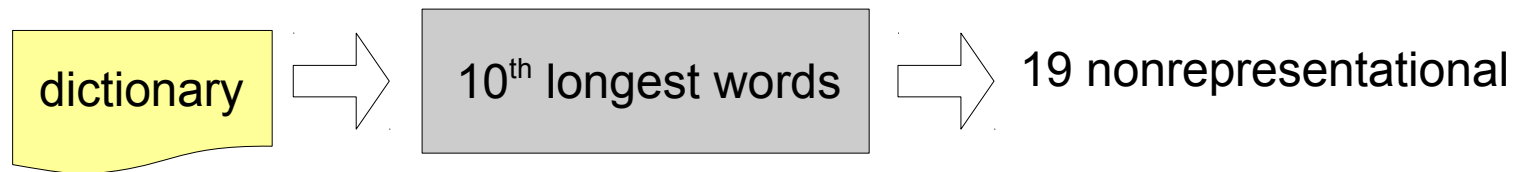
- Keep your system as simple as possible and design **bottom up**
- Analyze problems with a **divide-and-conquer** approach
 - Write simple tools that work together to accomplish complex tasks
(remember: the pipe plays a key role in the UNIX philosophy)
- Write software that relies on the **intelligence of the user**
 - Programs should do what the user asks, quietly, and without getting in the user's way
 - This approach works under the assumption that the user knows what he wants and how to do it (remember that UNIX has been designed by programmers for programmers)

Same Old Good Advices

- **Algorithm Design**
 - Recursively break down a problem into two or more sub-problems until these become simple enough to be solved directly
- **Functional programming:**
 - Write small functions, each focusing on a specific purpose
 - Avoid complicated, nested blocks: count the complexity of the modules and split them into smaller modules whenever the *cyclomatic complexity* of the module exceeded 10
- **Object Oriented programming:**
 - Describing large, complex systems as interacting objects makes them easier to understand
 - Strive for maximum cohesion: “An object should empower one thing, all of that thing and nothing but that thing. A method should do one job, the whole job and nothing but that job”

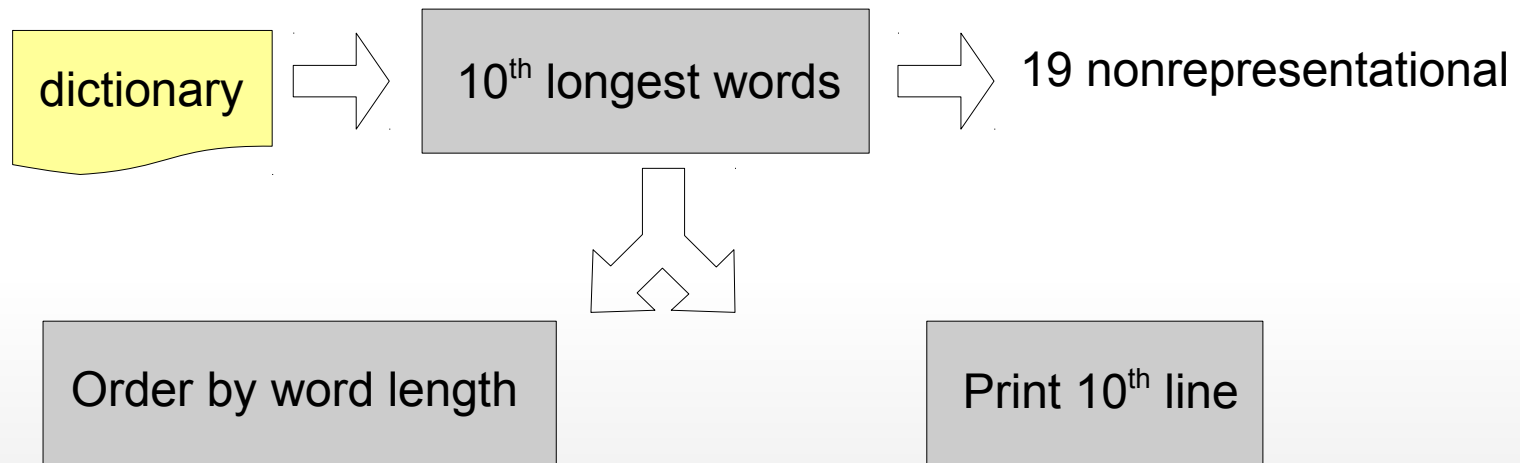
Example

“Print the 10th longest word in the dictionary”



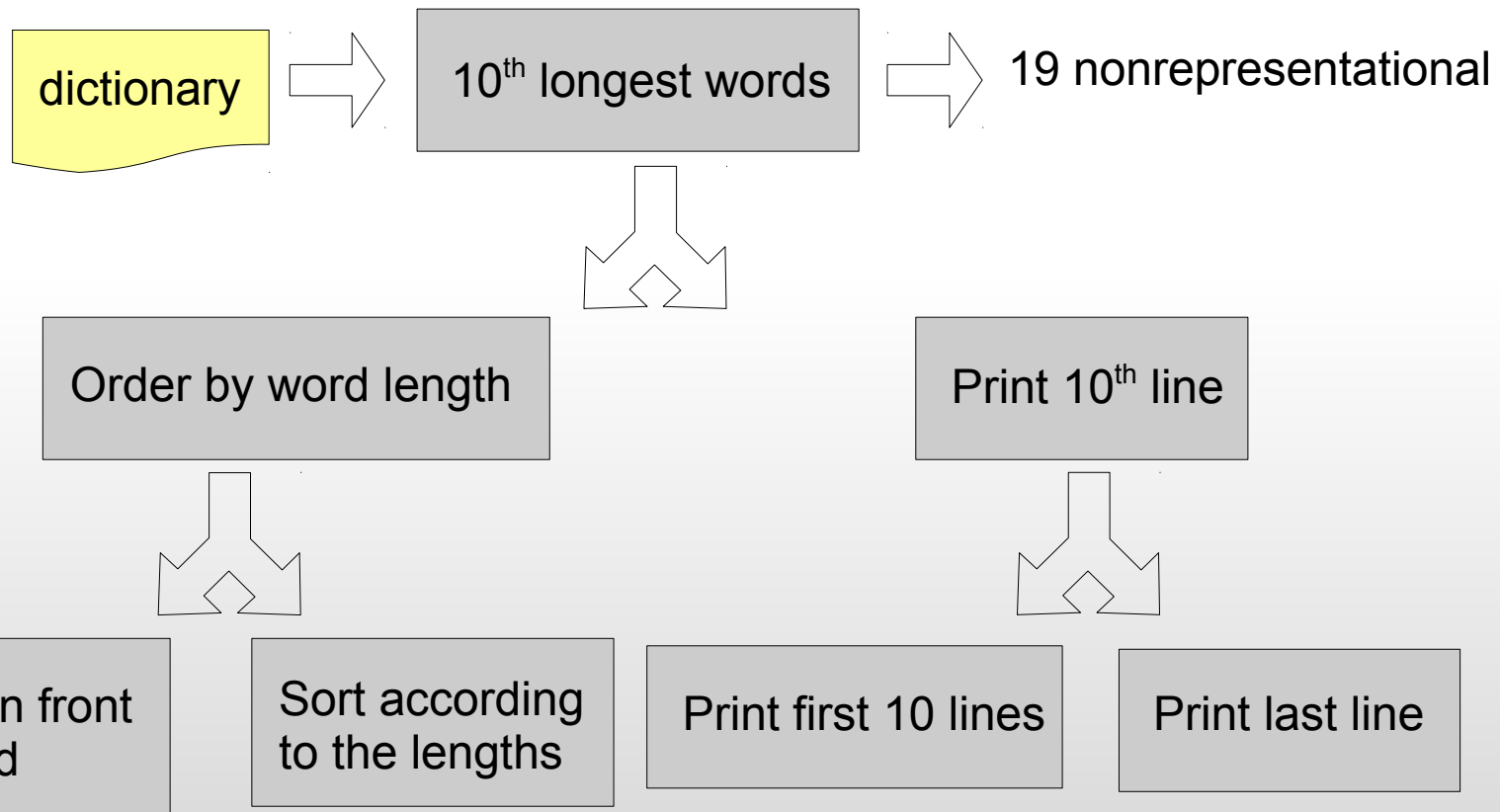
Example

“Print the 10th longest word in the dictionary”



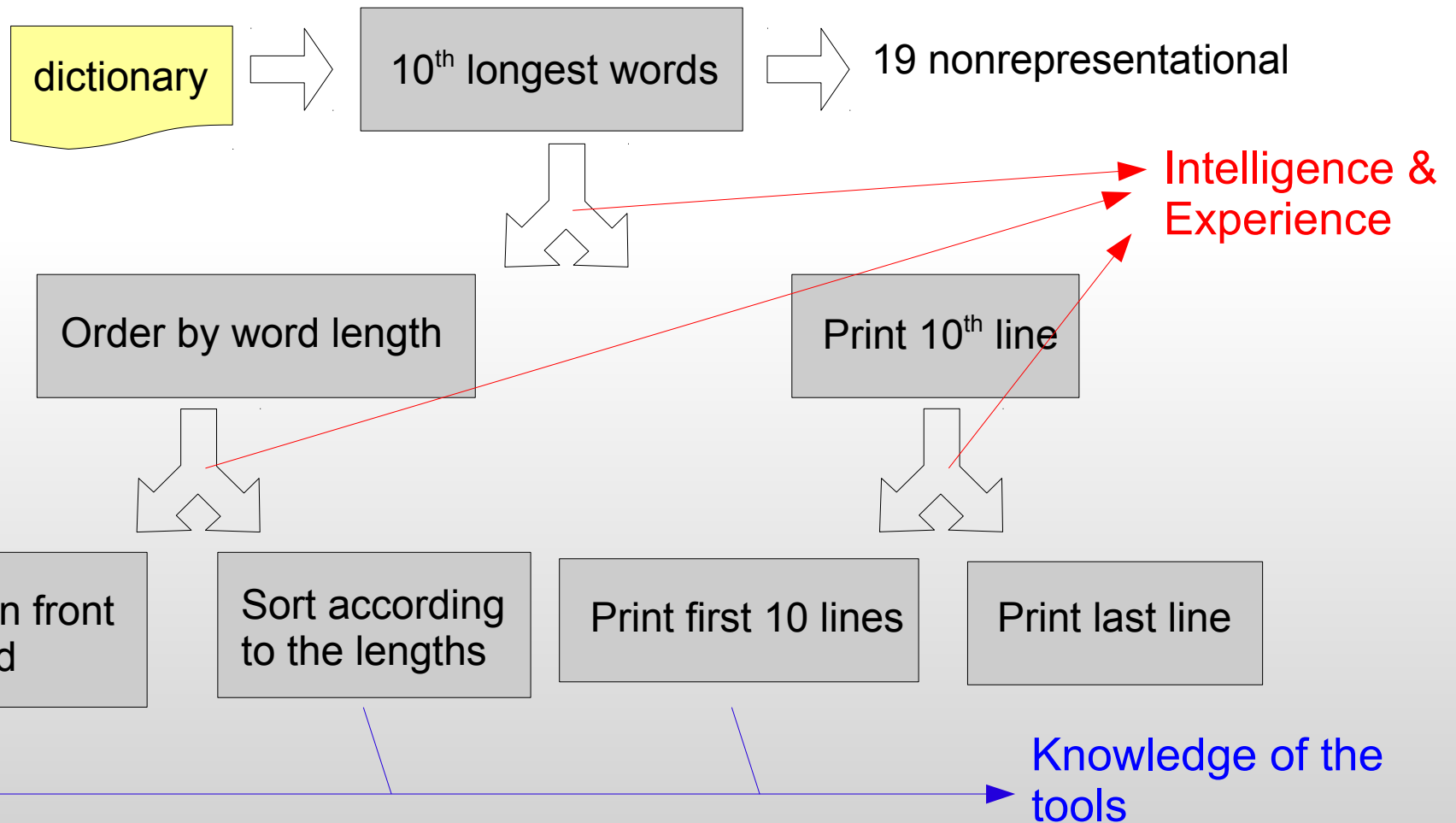
Example

“Print the 10th longest word in the dictionary”



Example

“Print the 10th longest word in the dictionary”



Philosophy Revised

Gancarz's 9 paramount precepts (The UNIX Philosophy)

1. Small is beautiful.
2. Make each program do one thing well.
3. Build a prototype as soon as possible.
4. Choose portability over efficiency.
5. Store data in flat text files.
6. Use software leverage to your advantage.
7. Use shell scripts to increase leverage and portability.
8. Avoid captive user interfaces.
9. Make every program a filter.

Philosophy Revised

Conner's Paramount presents

Eric Raymond's 17 Design Rules (The Art of Unix Programming)

1. Rule of Modularity: Write simple parts connected by clean interfaces
2. Rule of Clarity: Clarity is better than cleverness
3. Rule of Composition: Design programs to be connected to other programs
4. Rule of Separation: Separate policy from mechanism; separate interfaces from engines
5. Rule of Simplicity: Design for simplicity; add complexity only where you must
6. Rule of Parsimony: Write a big program only when it is clear by demonstration that nothing else will do
7. Rule of Transparency: Design for visibility to make inspection and debugging easier
8. Rule of Robustness: Robustness is the child of transparency and simplicity
9. Rule of Representation: Fold knowledge into data so program logic can be stupid and robust
10. Rule of Least Surprise: In interface design, always do the least surprising thing.
11. Rule of Silence: When a program has nothing surprising to say, it should say nothing
12. Rule of Repair: When you must fail, fail noisily and as soon as possible
13. Rule of Economy: Programmer time is expensive; conserve it in preference to machine time
14. Rule of Generation: Avoid hand-hacking; write programs to write programs when you can
15. Rule of Optimization: Prototype before polishing. Get it working before you optimize it
16. Rule of Diversity: Distrust all claims for "one true way".
17. Rule of Extensibility: Design for the future, because it will be here sooner than you think

Different Prophets, the Same Message

- Software should be small, simple, clear, and portable

“The only way to write complex software that won't fall on its face is to hold its global complexity down — to build it out of simple parts connected by well-defined interfaces, so that most problems are local and you can have some hope of upgrading a part without breaking the whole”

- Silence is gold

- When a program has nothing surprising to say, it should say nothing
- Only report (noisily) if something is wrong

- The importance of prototyping

- *“Make it run, then make it right, then make it fast ”*

- Keep the (graphic) interface separated from the engine

Consequences for the Users

- Unix trusts the user
 - Doesn't bother him with questions like “*Are you really sure? (Y/N)*”
- Unix gives the user a lot of options and a lot of power
 - But remember Spiderman:
“With great power comes great responsibility”
- The command line is very powerful, but it takes a very long time to master it

“Unix gives you enough rope to hang yourself...
and then a couple of feet more just to be sure.”

Microsoft Windows

- Unix was designed for scientists and programmers
Windows was designed for everybody else
 - For users that don't want to know how to program a computer (and they don't want to read manuals)
 - For users that don't know “*how*” and have only a vague idea of “*what*” they want to do
- The different assumption required a different approach
 - Programs must protect users against themselves (newbies tend to do stupid things that cause a lot of damage)
 - Programs often have to “*guess*” what to do, because users instructions are usually vague and incomplete
 - Programs must do a bit of everything, because users cannot combine different tools

Two Opposite Philosophies

- Two opposite mindsets
 - Unix relies in the intelligence of the users
 - Windows hard-codes the intelligence in the OS
- The goal of Windows is to make the system simple enough that it can be used without any understanding of how it works
 - PRO: flatten the learning curve
 - CONS: advocate users' ignorance
 - CONS: limit the user's flexibility
- The goal of UNIX is not to make things easy, but to make things possible
 - CONS: Steep learning curve. It doesn't shield the user from complexity