Jeff Bezos (Amazon's CEO) Mandate

From: Stevey's Google Platforms Rant

https://plus.google.com/112678702228711889851/posts/eVeouesvaVX

Jeff Bezos Mandate

1. All teams will henceforth expose their data and functionality through service interfaces.

2. Teams must communicate with each other through these interfaces.

3. There will be no other form of interprocess communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.

4. It doesn't matter what technology they use. HTTP, Corba, Pubsub, custom protocols -- doesn't matter. Bezos doesn't care.

Jeff Bezos Mandate

5. All service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.

6. Anyone who doesn't do this will be fired.

7. Thank you; have a nice day!

Stevey's Google Platforms Rant

- Google+ is a prime example of our complete failure to understand platforms
- The Golden Rule of Platforms, "Eat Your Own Dogfood", can be rephrased as "Start with a Platform, and Then Use it for Everything."
- Certainly not easily at any rate -- ask anyone who worked on platformizing MS Office.
- If you delay it, it'll be ten times as much work as just doing it correctly up front.

Stevey's Google Platforms Rant

- You can't cheat.
  - You can't have secret back doors for internal apps to get special priority access, not for ANY reason.
  - You need to solve the hard problems up front.

- I'm not saying it's too late for us, but the longer we wait, the closer we get to being Too Late.

On the Criteria to Be Used in Decomposing Systems into Modules

David L. Parnas

Communications of the ACM,

December 1972

Introduction

- Usually nothing is said about the criteria to be used in dividing the system into modules.

- This paper suggest some criteria which can be used in decomposing a system into modules.

- What is modularization?
  - “module" = responsibility assignment
  - (rather than a subprogram)

Benefits of Modular Programming

- **Managerial**: development time should be shortened because separate groups would work on each module with little need for communication

- **Product Flexibility:** it should be possible to make drastic changes to one module without a need to change others;

- **Comprehensibility:** it should be possible to study the system one module at a time.

Example: KWIC System

- Input: an ordered set of lines
  - Each line is an ordered set of words
  - Each word is an ordered set of characters.

- Any line may be "circularly shifted" by repeatedly removing the first word and appending it at the end of the line.

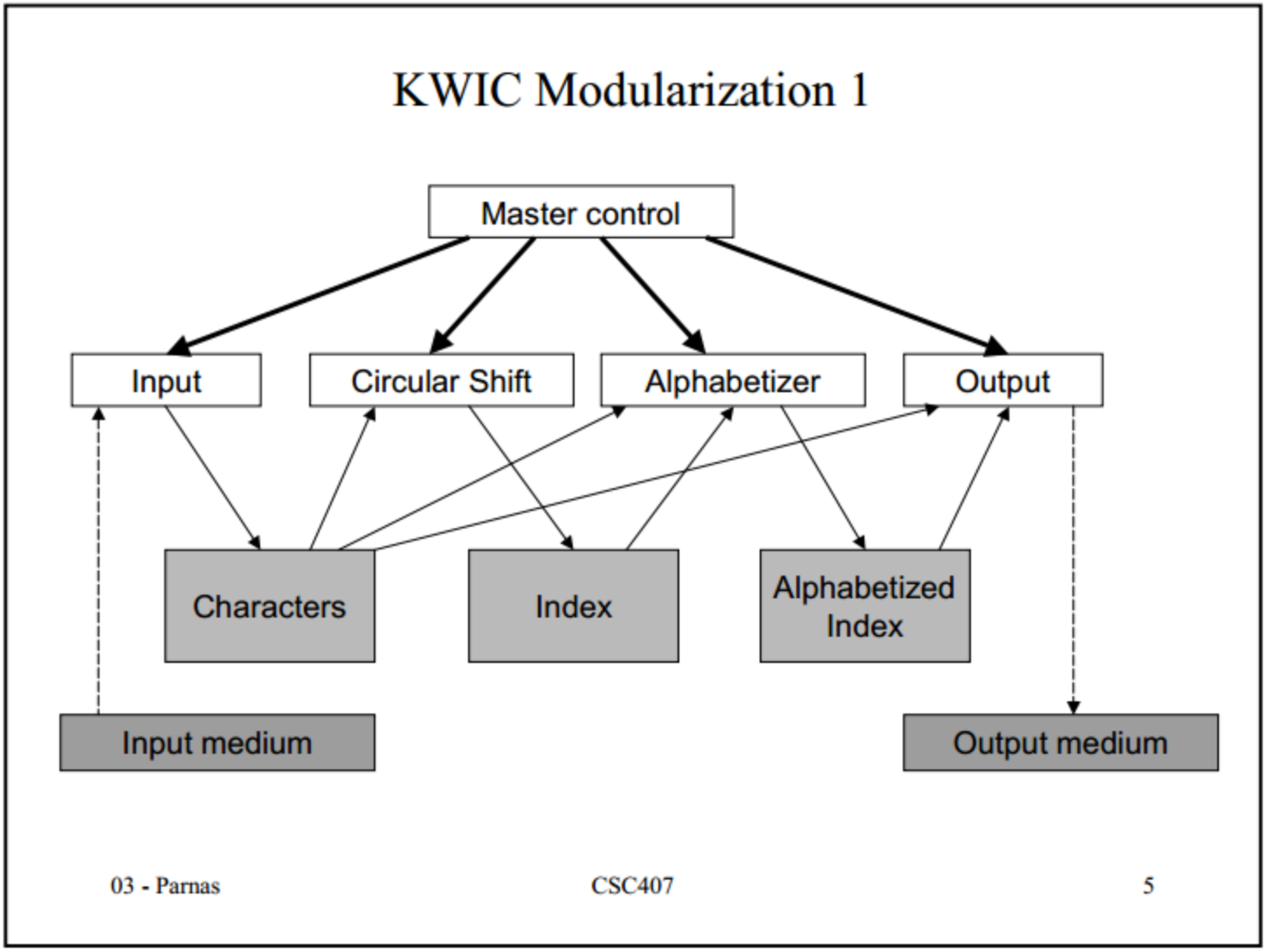
- Output:
  - A listing of all circular shifts of all lines in alphabetical order

KWIC Example

- Input:
  - Pattern-Oriented Software Architecture
  - Software Architecture
  - Introducing Design Patterns

- Output
  - Architecture Software
  - Architecture Pattern-Oriented Software
  - Design Patterns Introducing
  - Introducing Design Patterns
  - Patterns Introducing Design
  - Pattern-Oriented Software Architecture
  - Software Architecture
  - Software Architecture Pattern-Oriented

Modularization #1



Modularization #1

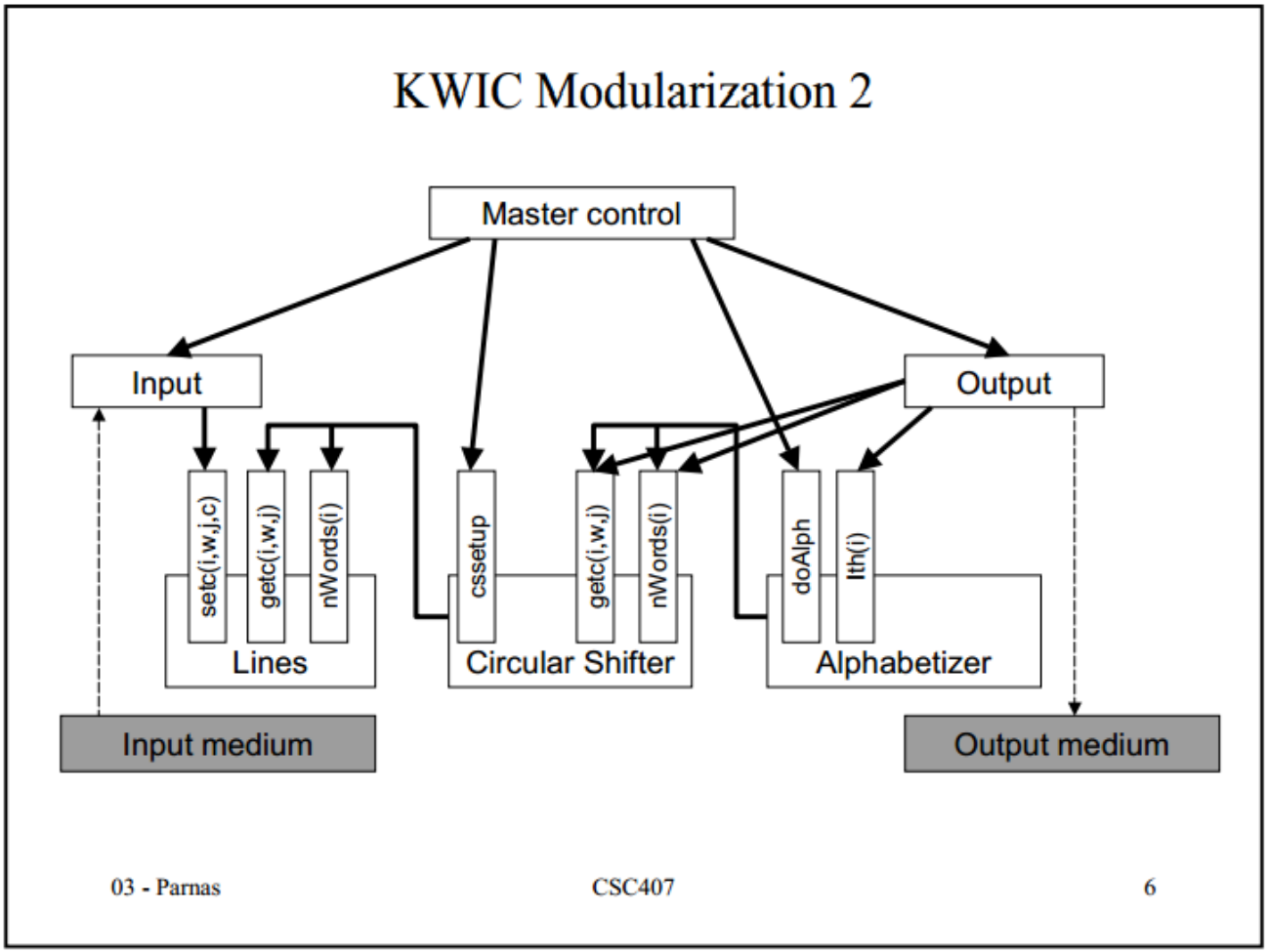
- **Input.** This module reads the data lines from the input medium and stores them in core for processing by the remaining modules

- **Circular Shift.** This module prepares an index which gives the address of the first character of each circular shiff

- **Alphabetizing.** This module produces an array in the same format as that produced by module 2. In this case, however, the circular shifts are listed in another order (alphabetically).

- **Output.** Using the arrays produced by module 3 and module 1, this module produces a formatted output listing all of the circular shifts

Modularization #2



Modularization #2

- **Lines Storage**:
  - *CHAR(r,w,c):* an integer representing the c-th character in the r-th line, w-th word
  - *SETCHAR(rpv,c,d):* causes the c-th character in the w-th word of the r-th line to be the character represented by d
  - *WORDS(r)* returns as value the number of words in line *r*.

- **Circular Shifter**:
  - The module creates the impression that we have a line holder containing all the circular shifts of the lines.
  - *CSCHAR(I,w,c)* provides the value representing the c-th character in the w-th word of the I-th circular shift

Comparison

- Modularization #1:
  - Each major step in the processing was a module

- Modularization #2: **information hiding** / abstract data types
  - Each module has one or more "secrets”
  - Each module is characterized by its knowledge of design decisions which it hides from all others.

Changeability

- Design decisions likely to change under many circumstances.
  1. Input format
  2. The decision to have all lines stored in core
  3. The decision to pack the characters four to a word
  4. The decision to make an index for the circular shifts rather that actually store them as such

- Differences between the two modularizations:
  - Change #1: confined to one module in both decompositions.
  - Change #2: for mod #1, changes in every module!
  - Change #3: for mod #1, changes in every module!
  - Change #4: confined to the circular shift module in the 2nd decomposition, but in the 1st decomposition the alphabetizer and the output routines will also change.

Independent Development

- In the first modularization the interfaces between the modules are the fairly complex formats and table organizations.

- In the second modularization the interfaces are more abstract.
  - They consist primarily in the function names and the numbers and types of the parameters.
  - These are relatively simple decisions and the independent development of modules should begin much earlier.

Comprehensibility

- To understand the output module in the first modularization, it will be necessary to understand something of the alphabetizer, the circular shifter, and the input module.
- The system will only be comprehensible as a whole.
- It is my subjective judgment that this is not true in the second modularization.

Conclusion

- We have tried to demonstrate by these examples that it is almost always incorrect to begin the decomposition into modules on the basis of a flowchart.

- We propose instead that one begins with a list of difficult design decisions or design decisions which are likely to change.

- Each module is then designed to hide such a decision from the others.

Later Comments

- To a man with a hammer, everything looks like a nail.
- To a Computer Scientist, everything looks like a language design problem.
- Languages and compilers are, in their opinion, the only way to drive an idea into practice.
- My early work clearly treated modularisation as a design issue, not a language issue. A module was a work assignment, not a subroutine or other language element.
- Although some tools could make the job easier, no special tools were needed to use the principal, just discipline and skill.