The Language Stew

Robert C. Martin
Object Mentor Inc.
The Vacuum Catastrophe
The Hardware Catastrophe
Count the orders of magnitude since PDP8

- $1 \times 10^5$ times faster
- $1 \times 10^6$ times more memory
- $1 \times 10^7$ times more storage
- $1 \times 10^4$ less cubic feet.
- $1 \times 10^3$ less power
- $1 \times 10^2$ less expensive

27 orders of magnitude

- And that doesn’t count the internet.
The Software Stagnation
The elements of our trade

- **Sequence**
  - pay = e.calcPay(today);
  - e.deliverPay(pay);

- **Selection**
  - if (e.isPayDay(today)) e.pay(today);

- **Iteration**
  - for (e : employees) e.tryPay(today);
public class DotCounter {
    public static int count(String s) {
        int dots = 0;
        for (int i=0; i<s.length(); i++) {
            if (s.charAt(i) == '.')
                dots++;
        }
        return dots;
    }
}

Sunday, 9 May 2010
public class DotCounter {
    public static int Count(string s) {
        int dots = 0;
        for (int i=0; i<s.Length(); i++)
            if (s[i] == '.')
                dots++;
        return dots;
    }
}

Sunday, 9 May 2010
int count_dots(char* s) {
    int count = 0;
    for (; *s; s++)
        if (*s == '.')
            count++;
    return count;
}
def count_dots(s)
    dots = 0
    s.each_char do |c|
        dots += 1 if c == '.'
    end
    dots
end
def dot_count(s: String): Int = {
    accumulate_dots(List.fromString(s), 0)
}

def accumulate_dots(s: List[Char],
    dots: Int): Int = {
    if (s.length == 0)
        dots
    else
        accumulate_dots(s.tail,
            if (s.head == '.
                dots + 1
            else
                dots)
    }
(defn count-dots [s]
  (if (empty? s)
      0
      (+
       (count-dots (rest s))
       (if (= \. (first s)) 1 0)))))
STR_PTR,  0
DOT,    '. '
DOTS,   0
COUNT_DOTS,  0
DCA STR_PTR
COUNT_NEXT, TAD I STR_PTR
SNZ
JMP DONE
SUB DOT
SNZ
ISZ DOTS
ISZ STR_PTR
JMP COUNT_NEXT
DONE,
CLA
TAD DOTS
JMP I COUNT_DOTS
Many different ways to say THE SAME THING
So is there no benefit to:

- Procedures?
- Objects?
- Functional?
- Information hiding
- Encapsulation
- Inheritance
- Design Patterns
- **Monads**
- etc?
Expression Vs. Technology
Why Doesn’t MDA work?

- Because
  - Sequence
  - Selection
  - and Iteration
- Are not well captured in diagrams.
Procedural Programming

- Easy to add new functions.
- Hard to add new Data Structure.
- State is nearly global
  - Making threading very difficult.

```c
void drawAllShapes(struct Shape* list) {
    for (struct Shape* s = list; s; s = s->next) {
        switch (s->type) {
        case square:
            drawSquare(s);
            break;
        case circle:
            drawCircle(s);
            break;
        }
    }
}
```
Object-Oriented Programming

- Control flow and compile-time dependencies are opposed.
  - Making it easy to add new data structures
  - But hard to add new functions.

- *Some* locality of state

```java
public class Payroll {
  private EmployeeGateway employeeGateway;
  public void payDay(Date payDate) {
    foreach (Employee employee in employeeGateway.findAll())
      employee.calculatePay(payDate);
  }
}
```

---

Control Flow

- `Payroll`
  - `findAll`
- `EmployeeGateway`
  - `findAll`

Compile-time Dependencies

- `ADO Employee Gateway`
Definitions

- Structured Programming:
  - discipline imposed upon direct transfer of control.

- Object-Oriented Programming:
  - discipline imposed upon indirect transfer of control.
Functional Programming

- State is local
- Better protection from Threads.
- Source code dependencies align with Control Flow
  - Making it easy to add new functions
  - But hard to add new datatypes.
- Functions are first class elements.
  - Source Code Dependencies oppose Control Flow
  - Not best of both worlds, but workable.
- State is extremely local
  - Making multiprocessing easier.
Definition

- Functional Programming:
  - Discipline imposed upon mutable state.
  - Nahhhhh.
Decades of war has left a ruined landscape.

- C vs. Pascal
- C++ to the Rescue.
- C++ vs Smalltalk
- Smalltalk’s untimely demise.
- The ascendancy of static typing.
- TDD!
  - and the revolution begins.
- Python, Ruby, Rails!
Java, C#, C++

- Urgh.
- OO-ish.
- Statically typed.
- Source code dependencies can oppose flow of control.
- Componentizable.
- Wordy and constrained.

```java
public boolean equals(Object obj) {
    if (obj == null)
        return false;
    if (!(obj.getClass() == getClass()))
        return false;
    MediaCopy that = (MediaCopy) obj;
    return this.id.equals(that.id) &&
           this.media.equals(that.media);
}
```
Ruby

- OO
- Dynamically Typed.
- Compoundable
- Monkey-patch-able!
  - “I reject your reality and substitute my own.”
- Elegant, and yet...
Ruby

- OO
- Dynamically Typed.
- Componentizable
- Monkey-patch-able!
  - “I reject your reality and substitute my own.”
- Elegant, and yet...

```ruby
def discountedPrice(bag)
  minPrice = nil
  minAllocation = nil
  bag.forEachDiscountAllocation do |allocation|
    price = allocation.calculateDiscount(self)
    if (minPrice == nil || price < minPrice)
      minPrice = price
      minAllocation = allocation.dup
    end
  end

  undiscoutedBooks = (bag.dup)
  undiscoutedBooks.remove(minAllocation.books)
  minPrice + grossPriceOf(undiscoutedBooks.books)
end
```
Scala / F#

- Hybrid languages.
  - Can be procedural.
  - Can be OO.
  - Can be functional.
  - Statically typed.
  - Warty like C++

```scala
trait Ord {
  def <(that: Any): Boolean
  def <=(that: Any): Boolean = (this < that) || (this == that)
  def >(that: Any): Boolean = !(this <= that)
  def >=(that: Any): Boolean = !(this < that)
}
```
Clojure

- Functional
- Java Stack
- Very disciplined State/Identity/Value model. STM!
  - Atoms
  - Agents
  - Refs in transactions.
- Lots of Insidious Sequential Parenthesis.
STM

- **Refs**

  ```clojure
  (def current-track (ref "Venus, the Bringer of Peace"))
  (def current-composer (ref "Holst"))
  (dosync (ref-set current-track "Credo")
    (ref-set current-composer "Byrd"))
  ```

- **Atoms**

  ```clojure
  (def current-track (atom {:title "Credo" :composer "Byrd"}))
  (reset! current-track {:title "Spem in Alium" :composer "Tallis"})
  (swap! current-track assoc :title "Sancte Deus")
  ```

- **Agents**

  ```clojure
  (def counter (agent 0))
  (send counter inc)
  ```

- **Dereference**

  ```clojure
  @counter
  ```
Moore's Law is Dead.
Well, at least for processor speed.

- Individual cycle times aren’t going to get faster.
- Multiple cores are the clear solution.
- And that means:
Well, at least for processor speed.

- Individual cycle times aren’t going to get faster.
- Multiple cores are the clear solution.
- And that means:

Concurrency
Concurrency Antidote

- Extreme Localized Scope
- Disciplined model of State/Value/Identity
- Functional Language
- Java Stack

- I vote for Clojure.
But can mere mortals...
fin

- unclebob @ objectmentor.com
- fitnesse.org
- cleancodeproject.com