Language and Library API Design for Usability of Ruby

Akira Tanaka
akr@fsij.org
National Institute of Advanced Industrial Science and Technology (AIST)

Goal

Design good programming language and library API

- "good" means usability for programmers
- programmers should be able to create a program more easily

Outline

- Examples of conflicts with usability and the other good properties of programming languages and library APIs
- Design Patterns for explaining the policy

Background

- I feel Ruby is comfortable
  - I can program my idea frankly
- But there are inconsistencies
- Proposal to fix them tend to be rejected

- Consistency is not the most important policy
- What's the design policy of Ruby?
- I'd like to know the design policy for comfortable language and library API

Question

- When Ruby ignore consistency for usability?
- How people can study the policy?

Question, Generalized

- When usability should be preferred over other good language/library properties: consistency, simplicity, etc?
- When the good properties should be preferred over usability?
- How we can distinguish them?
- How we can explain this policy?

Inconsistency Example
bang (!) methods

- Method name can end with bang (!) in Ruby
- Bang is used for dangerous methods
  Programmers should be careful to use it
  Destructive methods in most cases
- This usage of bang is similar to Scheme
- But it is not used consistently in Ruby
  Bang is used for some of destructive methods
  (not all)

Destructive Methods in Array

- clear
- collect!
- compact!
- concat
- delete
- delete_at
- delete_if
- fill
- flatten!
- insert
- map!
- pop
- push
- reject!
- replace
- reverse!
- shift
- slice!
- sort!
- uniq!
- unshift

Bang (!) Methods Inconsistency

- Method name ends with bang is destructive
- Method name ends without bang is sometimes destructive

Several people try to fix it

- Proposal for adding bang for all destructive methods

- Rejected
- Reason:
  - Too many destructive methods in Ruby
  - Destructive methods are common in imperative style
  - Bang gets attention but programmers cannot pay attention for too many bangs
  - Consistent bang is less useful in Ruby

Consistency v.s. Usability

- If all destructive methods ends with bang,
  - [good] easy to remember the method names
  - [bad] too many bang is less useful for attention
- Consistency and Usability conflicts here

Complex Design Example block and lambda

- Ruby has lambda as Scheme
  lambda {|x| x + 1 } # (lambda (x) (+ x 1))
- But block is used much more frequently
  obj.method(args) { ... }
- Ruby's method call can take a block
  array.map {|x| x + 1 }
- Similar to higher order function
  (map (lambda (x) (+ x 1)) array)
- {|x| x + 1 } is not an expression but a builtin syntax for method call

Block violates simplicity

- lambda can be used instead of block
- block is not usable if two or more functions are passed
- Simpler design: no block. lambda only

Why block?

- There are many usages for method call with single block
- block is succinct than lambda
  - a.map {|x| x + 1 }
  - (map (lambda (x) (+ x 1)) a)
- less nestings
- Succinct programs are easy to read and write (if not too succinct)

Succinctness

- Succinct program is easy to write
  - less number of types (or keystrokes)
- Succinct program is easy to read
  - less number of program elements

Simplicity v.s. Usability

- If Ruby don't have block,
  - [good] syntax and semantics are simplified
  - [bad] make programs less succinct
- Simplicity and usability conflicts here

Bad Inconsistency

There are bad inconsistencies in Ruby

- Arguments passing semantics different between block and method
  - Almost fixed in Ruby 1.9
- "utc" and "local" method in Time class is destructive
  - hard to fix because incompatibility
- etc.

How to cope with the conflicts

- Resolve conflicts if possible
- Prefer one which is more important
- Decide it objectively if possible
- Decide it subjectively, or
- Don't decide until possible

Hard to Formalize the Decision

- No absolute axiom/theory
- Good programmers do it implicitly
- Somewhat subjective

How to explain the decision method

- Various techniques are used for usability
- We should compare advantage and disadvantage of the technique
- This is difficult to be quantitative
- Design pattens (pattern language) would be a good way to describe them

Possible Pattern
Optimize for Common Usage

- bang-methods and block concentrates common usage
- How to apply:
  - Guess common usage
    - imperative style
    - higher order function which takes single function
  - Optimize for that
    - bang methods
    - block

Possible Pattern Incremental Design

- We may not certain about common usage
  - Imperative style is really common?
  - Single block is really useful in most case?
- Find common usage in experience
  - idiom
  - code search
  - etc.
- Avoid future incompatibilities
  - Method name should explain the behavior to avoid future method renaming

Other possible patterns

- Fewer class/arguments
- Feature-rich (many methods) class than compact class
- DRY (Don't Repeat Yourself)
- Delay decisions
- Respect programmer's knowledge
- Concentrate to base level programming over meta programming
- DSL (Domain Specific Languages)
- White spaces for structures

Summary

- Usability can conflict with the other good properties
- There are various techniques for usability
  - optimize for common usages
  - incremental design
  - etc.
- Design patterns would be good to describe the techniques