# Simplicity Matters

Rich Hickey

Simplicity is prerequisite for reliability

Edsger W. Dijkstra

# Word Origins

- Simple

  *sim- plex*

  one fold/braid

  vs complex

- Easy

  *ease < aise < adjacens*

  lie near

  vs hard

# Simple

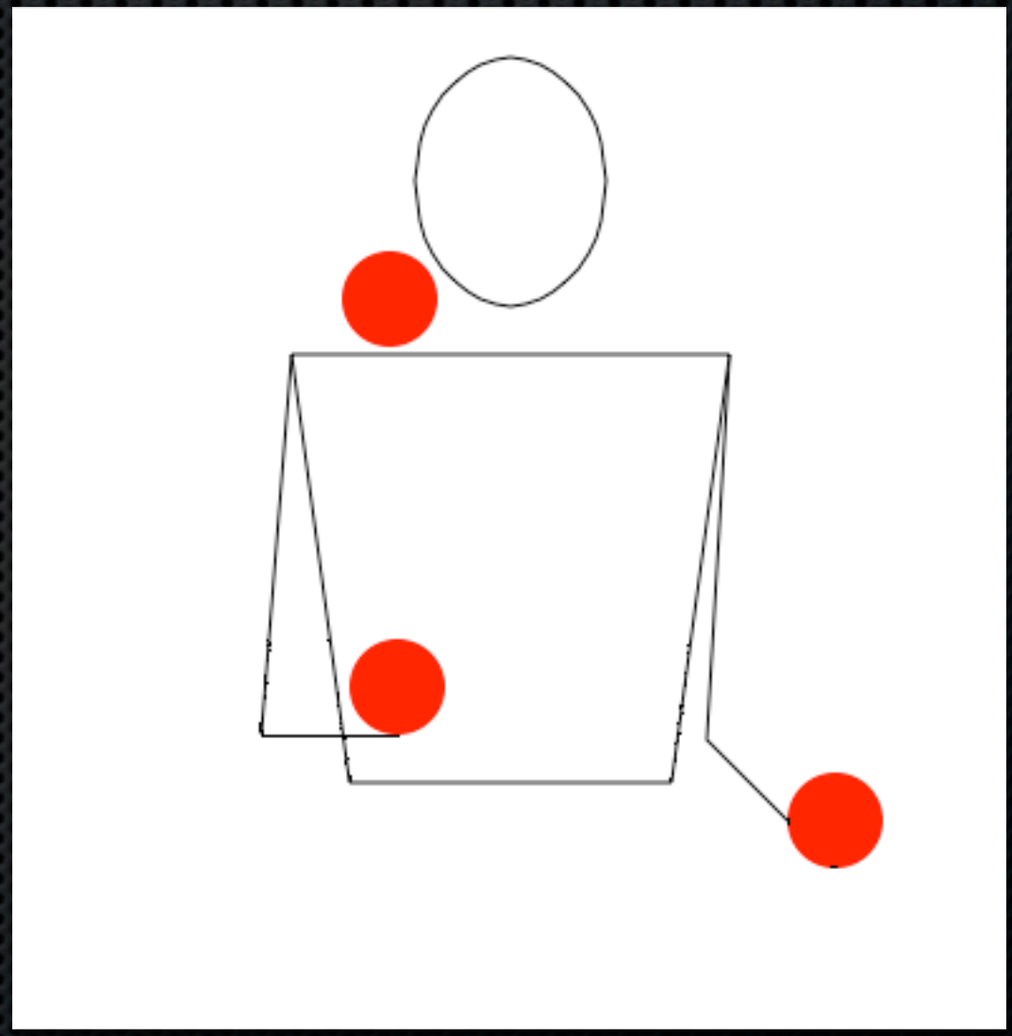
- One fold/braid
  - One role
    - One task
    - One concept
    - One dimension
- But not
  - One instance
    - One operation
    - About lack of interleaving, not cardinality
    - *Objective*

# Easy

- Near, at hand
  - on our hard drive, in our tool set, IDE, apt get, gem install...
- Near to our understanding/skill set
  - familiar
- Near our capabilities
- Easy is *relative*

# Limits

- We can only hope to make reliable those things we can understand
- We can only consider a few things at a time
- Intertwined things must be considered together
- Complexity undermines understanding



# Change

- Do more, Do it differently, Do it better
- Changes to software require analysis and decisions
- Your ability to reason about your program is critical
  - More than tests, types, tools, process

# Simplicity = Opportunity

- Architectural Agility wins
  - else - push the elephant
- Design is about pulling things apart
- Repurpose, substitute, move, combine, extend

LISP programmers know the value of everything and the cost of nothing.

Alan Perlis

Programmers know the benefits of everything and the tradeoffs of nothing.


AbbeyRoad

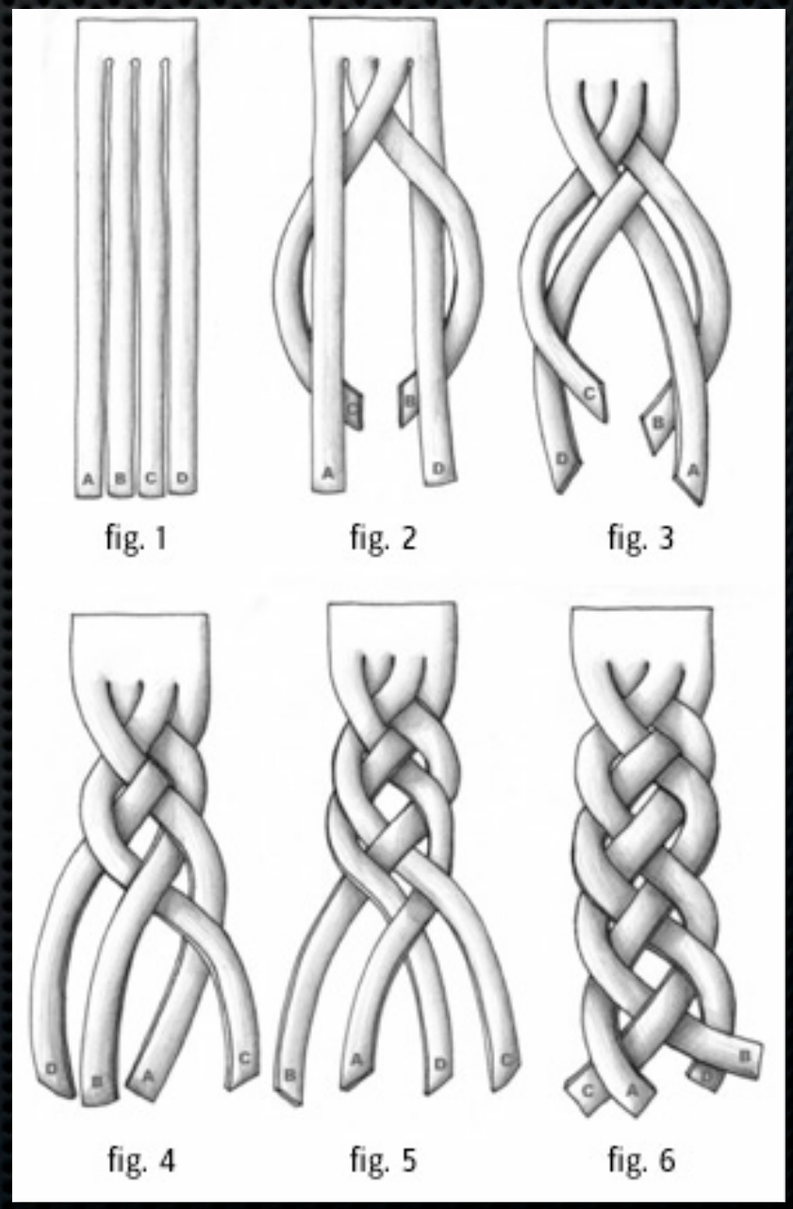

AbbeyRoad

# Programmers vs Programs

- We focus on ourselves
  - programmer convenience
  - programmer replaceability
- Rather than the programs
  - software quality, correctness
  - maintenance, change
- gem install hairball

# Complect

- To interleave, entwine, braid
- Don’t do it!
  - Complecting things is the source of complexity
- Best to avoid in the first place



# Making Things Easy

- Bring to hand by installing
  - getting approved for use
- Become familiar by learning, trying
- But mental capability?
  - not going to move very far
- Make challenges easy by simplifying them

We can be creating the exact same programs out of significantly simpler components

# What's in *your* Toolkit?

| Complexity | Simplicity |
|---|---|
| State, Objects | Values |
| Methods | Functions, Namespaces |
| variables | Managed refs |
| Inheritance, switch, matching | Polymorphism a la carte |
| Syntax | Data |
| Imperative loops, fold | Set functions |
| Actors | Queues |
| ORM | Declarative data manipulation |
| Conditionals | Rules |
| Inconsistency | Consistency |

Simplicity--the art of maximizing the amount of work not done--is essential.

http://agilemanifesto.org/principles.html

Simplicity is not an objective in art, but one achieves simplicity despite one's self by entering into the real sense of things

Constantin Brancusi

# Lists and Order

- A sequence of things
- Does order matter?
  - `[first-thing second-thing third-thing ...]`
  - `[depth width height]`
- `set[x y z]`
  - order clearly doesn’t matter

# Why Care about Order?

- Complects each thing with the next
- Infects usage points
- Inhibits change
  - `[name email] -> [name phone email]`
- “We don’t do that”

# Order in the Wild

| Complex | Simple |
| --- | --- |
| Positional arguments | Named arguments or map |
| Syntax | Data |
| Product types | Associative records |
| Imperative programs | Declarative programs |
| Prolog | Datalog |
| Call chains | Queues |
| XML | JSON, Clojure literals |
| ... | |

# Maps (aka hashes), Dammit!

- First class associative data structures
- Idiomatic support
  - literals, accessors, symbolic keys...
- Generic manipulation
- Use ‘em

# Information *is* Simple

- Don’t ruin it
- By hiding it behind a micro-language
  - i.e. a class with information-specific methods
  - thwarts generic data composition
  - ties logic to representation du jour
- Represent data as data

# Encapsulation

- Is for implementation details
- Information doesn't have implementation
  - Unless *you* added it - why?
- Information *will* have representation
  - have to pick one

# Wrapping Information

- The information class:
  - `IPersonInfo{`

    `getName();`

    `... verbs and other awfulness ...}`
- A service based upon it:
  - `IService{`

    `doSomethingUseful(IPersonInfo); ...}`

# Can You Move It?

- Litmus test - can you move your subsystems?
  - out of proc, different language, different thread?
- Without changing much
  - Not seeking transparency here

# Subsystems Must Have

- Well-defined boundaries
- Abstracted operational interface (verbs)
- General error handling
- Take/return data
  - `IPersonInfo` - oops!
- Again, maps (hashes)

# Simplicity is a Choice

- Requires vigilance, sensibilities and care
- Equating simplicity with ease and familiarity is wrong
  - Develop sensibilities around entanglement
- Your 'reliability' tools (testing, refactoring, type systems) don't care if program is simple or not
- Choose simple constructs

# Simplicity Matters

- Complexity inhibits understanding
  - and therefor robustness
- Simplicity enables change
  - It is the primary source of true agility
- Simplicity = Opportunity
- Go make (simple) things

Simplicity is the ultimate sophistication.

Leonardo da Vinci