Simple Made Easy

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
Construct vs Artifact

- We focus on experience of use of construct
  - programmer convenience
  - programmer replaceability
- Rather than the long term results of use
  - software quality, correctness
  - maintenance, change
- We must assess constructs by their artifacts
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

- Changes to software require analysis and decisions
- What will be impacted?
- Where do changes need to be made?
- Your ability to reason about your program is critical to changing it without fear
  - Not talking about proof, just informal reasoning
Debugging

- What’s true of every bug in the field?
- It has passed the type checker
- and all the tests
- Your ability to reason about your program is critical to debugging
Development Speed

- Emphasizing ease gives early speed
- Ignoring complexity will slow you down over the long haul
- On throwaway or trivial projects, nothing much matters
Easy Yet Complex?

- Many complicating constructs are
  - Succinctly described
  - Familiar
  - Available
  - Easy to use
- What matters is the complexity they yield
- Any such complexity is incidental
Benefits of Simplicity

- Ease understanding
- Ease of change
- Easier debugging
- Flexibility
  - policy
  - location etc
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 things near by simplifying them
Parentheses are Hard!

- Not at hand for most
- Nor familiar
- But are they simple?
- Not in CL/Scheme
  - overloaded for calls and grouping
  - for those that bothered trying, this is a valid complexity complaint
- Adding a data structure for grouping, e.g. vectors, makes each simpler
- Minimal effort can then make them easy too
LISP programmers know the value of everything and the cost of nothing.

Alan Perlis
What’s in your Toolkit?

<table>
<thead>
<tr>
<th>Complexity</th>
<th>Simplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>State, Objects</td>
<td>Values</td>
</tr>
<tr>
<td>Methods</td>
<td>Functions, Namespaces</td>
</tr>
<tr>
<td>vars</td>
<td>Managed refs</td>
</tr>
<tr>
<td>Inheritance, switch, matching</td>
<td>Polymorphism a la carte</td>
</tr>
<tr>
<td>Syntax</td>
<td>Data</td>
</tr>
<tr>
<td>Imperative loops, fold</td>
<td>Set functions</td>
</tr>
<tr>
<td>Actors</td>
<td>Queues</td>
</tr>
<tr>
<td>ORM</td>
<td>Declarative data manipulation</td>
</tr>
<tr>
<td>Conditionals</td>
<td>Rules</td>
</tr>
<tr>
<td>Inconsistency</td>
<td>Consistency</td>
</tr>
</tbody>
</table>
Compliekt

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

- To place together
- Composing simple components is the key to robust software
Modularity and Simplicity
Modularity and Simplicity
Modularity and Simplicity

- Partitioning and stratification don't imply simplicity
  - but are enabled by it
- Don't be fooled by code organization
State is Never Simple

- Completes value and time
- It *is* easy, in the at-hand and familiar senses
- Interweaves everything that touches it, directly or indirectly
  - Not mitigated by modules, encapsulation
- Note - this has nothing to do with asynchrony
Not all refs/vars are Equal

- None make state simple
- All warn of state, help reduce it
- Clojure and Haskell refs `compose` value and time
  - Allow you to extract a simple value
  - Provide abstractions of time
- Does your var do that?
## The Complexity Toolkit

<table>
<thead>
<tr>
<th>Construct</th>
<th>Complects</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>Everything that touches it</td>
</tr>
<tr>
<td>Objects</td>
<td>State, identity, value</td>
</tr>
<tr>
<td>Methods</td>
<td>Function and state, namespaces</td>
</tr>
<tr>
<td>Syntax</td>
<td>Meaning, order</td>
</tr>
<tr>
<td>Inheritance</td>
<td>Types</td>
</tr>
<tr>
<td>Switch/matching</td>
<td>Multiple who/what pairs</td>
</tr>
<tr>
<td>var(iable)s</td>
<td>Value, time</td>
</tr>
<tr>
<td>Imperative loops,</td>
<td>what/how</td>
</tr>
<tr>
<td>fold</td>
<td>what/who</td>
</tr>
<tr>
<td>Actors</td>
<td>OMG</td>
</tr>
<tr>
<td>ORM</td>
<td>Why, rest of program</td>
</tr>
<tr>
<td>Conditionals</td>
<td></td>
</tr>
</tbody>
</table>
# The Simplicity Toolkit

<table>
<thead>
<tr>
<th>Construct</th>
<th>Get it via…</th>
</tr>
</thead>
<tbody>
<tr>
<td>Values</td>
<td>final, persistent collections</td>
</tr>
<tr>
<td>Functions</td>
<td>a.k.a. stateless methods</td>
</tr>
<tr>
<td>Namespaces</td>
<td>language support</td>
</tr>
<tr>
<td>Data</td>
<td>Maps, arrays, sets, XML, JSON etc</td>
</tr>
<tr>
<td>Polymorphism a la carte</td>
<td>Protocols, type classes</td>
</tr>
<tr>
<td>Managed refs</td>
<td>Clojure/Haskell refs</td>
</tr>
<tr>
<td>Set functions</td>
<td>Libraries</td>
</tr>
<tr>
<td>Queues</td>
<td>Libraries</td>
</tr>
<tr>
<td>Declarative data manipulation</td>
<td>SQL/LINQ/Datalog</td>
</tr>
<tr>
<td>Rules</td>
<td>Libraries, Prolog</td>
</tr>
<tr>
<td>Consistency</td>
<td>Transactions, values</td>
</tr>
</tbody>
</table>
Environmental Complexity

- Resources, e.g. memory, CPU
- *Inherent* complexity in implementation space
  - All components contend for them
- Segmentation
  - waste
- Individual policies don’t compose
  - just make things more complex
Programming, when stripped of all its circumstantial irrelevancies, boils down to no more and no less than very effective thinking so as to avoid unmastered complexity, to very vigorous separation of your many different concerns.

Edsger W. Dijkstra
Abstraction for Simplicity

- Abstract
  - drawn away
- vs Abstraction as complexity hiding
- I don’t know, I don’t want to know
What

- Operations
- Form abstractions from related sets of functions
  - Small sets
- Represent with polymorphism constructs
- Specify inputs, outputs, semantics
  - Use only values and other abstractions
- Don’t complect with:
  - How
Who

- Entities implementing abstractions
- Build from subcomponents direct-injection style
  - Pursue many subcomponents
    - e.g. policy
  - Don’t complect with:
    - component details
    - other entities
How

- Implementing logic
- Connect to abstractions and entities via polymorphism constructs
- Prefer abstractions that don’t dictate how
  - Declarative tools
- Don’t complect with:
  - anything
When, Where

- Strenuously avoid complecting these with anything in the design
- Can seep in via directly connected objects
  - Use queues
Why

- The policy and rules of the application
- Often strewn everywhere
  - in conditionals
  - completed with control flow etc
- Explore rules and declarative logic systems
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
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
Simplifying

- Identifying individual threads/roles/dimensions
- Following through the user story/code
- Disentangling
Simplicity is a Choice

- Requires vigilance, sensibilities and care
- Your sensibilities equating simplicity with ease and familiarity are wrong
  - Develop sensibilities around entanglement
- Your 'reliability' tools (testing, refactoring, type systems) don't care
  - and are quite peripheral to producing good software
Simplicity Made Easy

- Choose simple constructs over complexity-generating constructs
  - It’s the artifacts, not the authoring
- Create abstractions with simplicity as a basis
- Simplify the problem space before you start
- Simplicity often means making more things, not fewer
- Reap the benefits!
Simplicity is the ultimate sophistication.

Leonardo da Vinci