www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/1.swf

Simple Made Easy

Rich Hickey

www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/2.swf

Simplicity is prerequisite for reliability

Edsger W. Dijkstra

www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/3.swf

Word Origins

- Simple

  sim- plex

  one fold/braid

  vs complex

- Easy

  ease < aise < adjacens

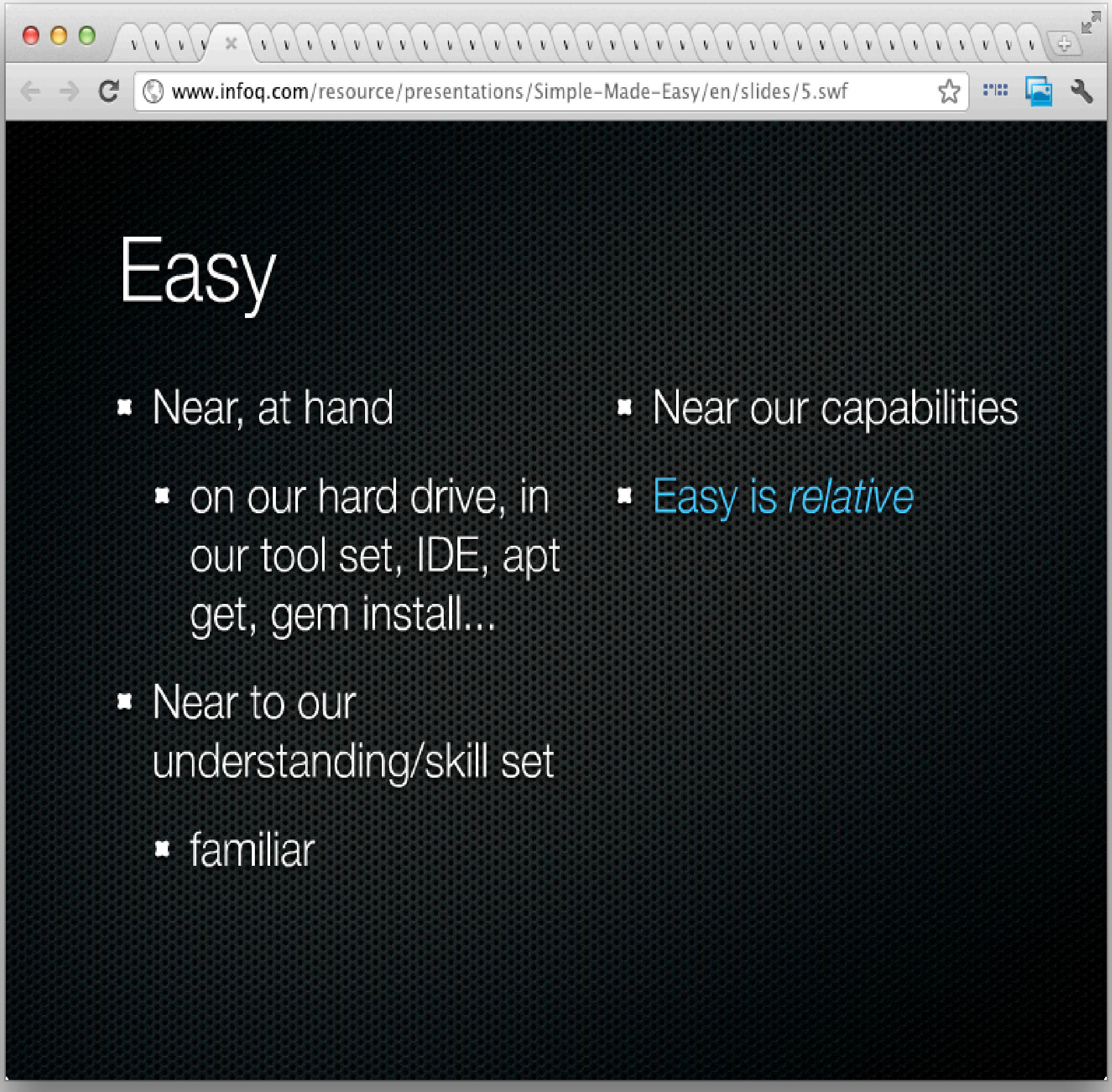
  lie near

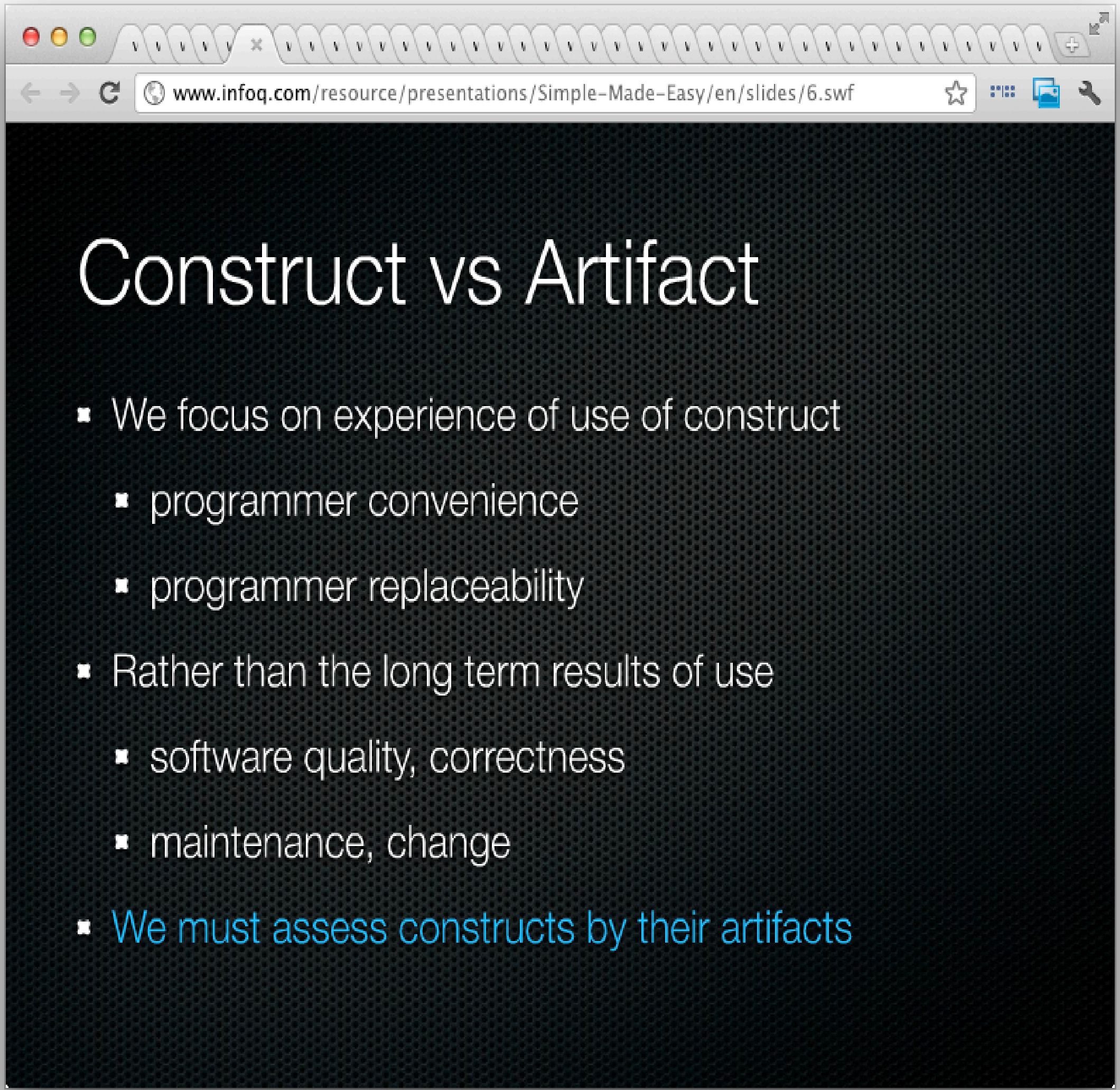
  vs hard

www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/4.swf

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*

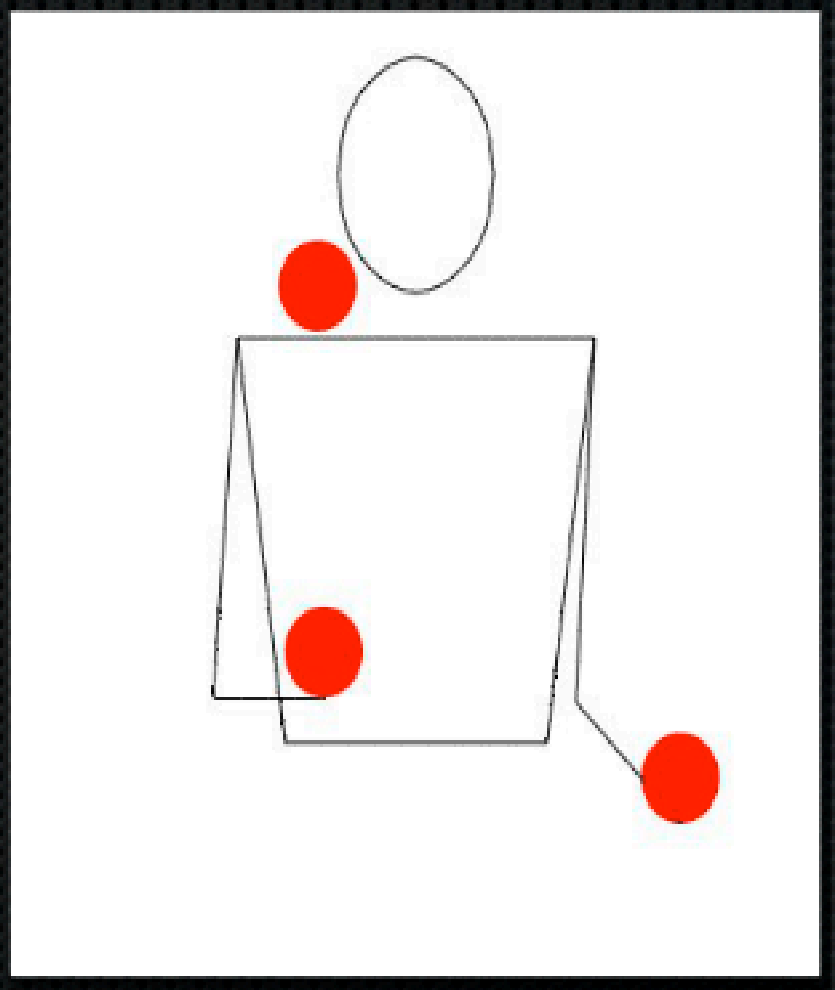

www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/5.swf
Easy
Near, at hand
on our hard drive, in our tool set, IDE, apt get, gem install...
Near to our understanding/skill set
familiar
Near our capabilities
Easy is relative

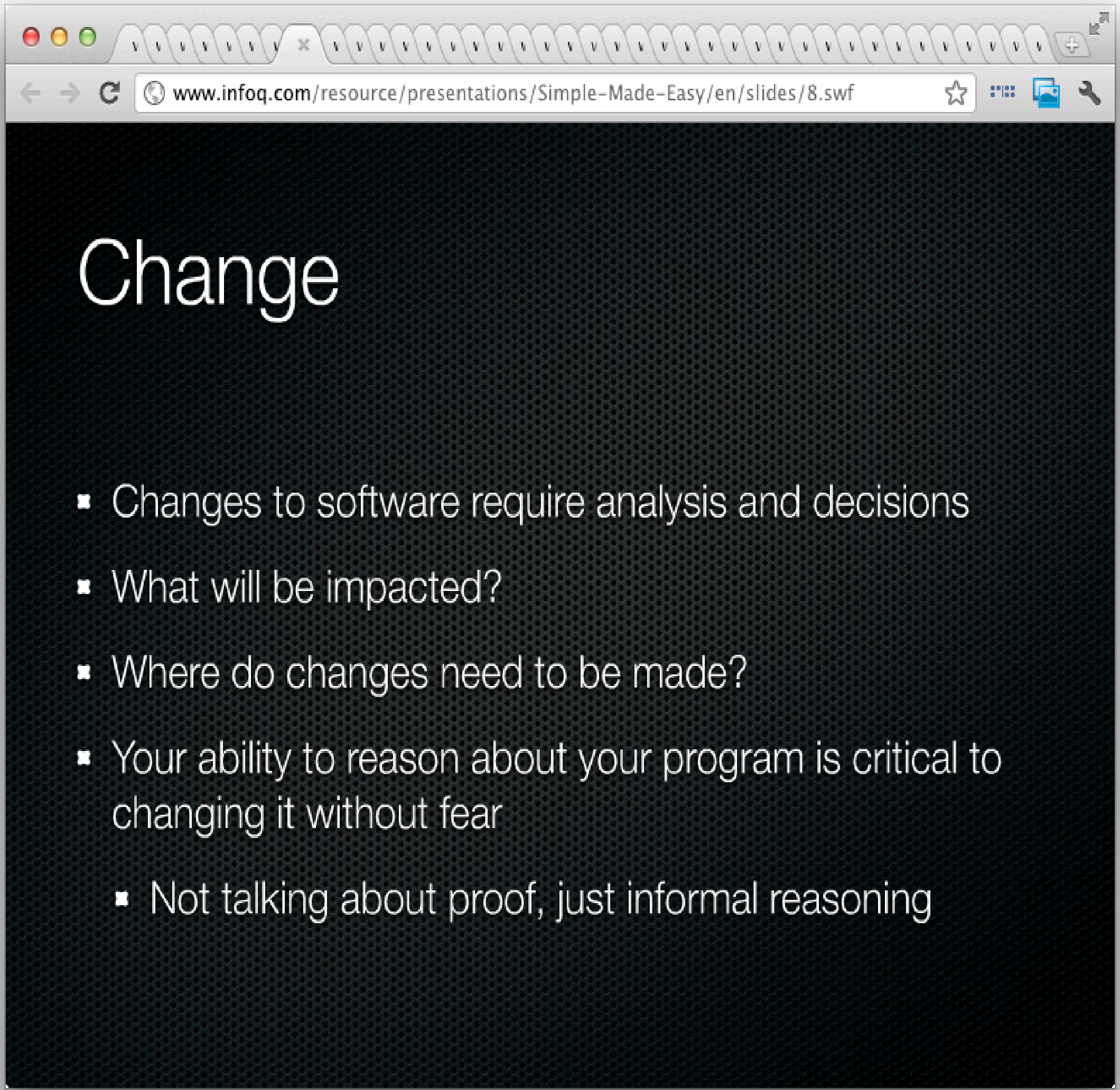

www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/6.swf
Construct vs Artifact
We focus on experience of use of construct
programmer convenience
programmer replaceability
Rather than the long term results of use
software quality, correctness
maintenance, change
We must assess constructs by their artifacts

www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/7.swf

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




www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/8.swf
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

www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/9.swf

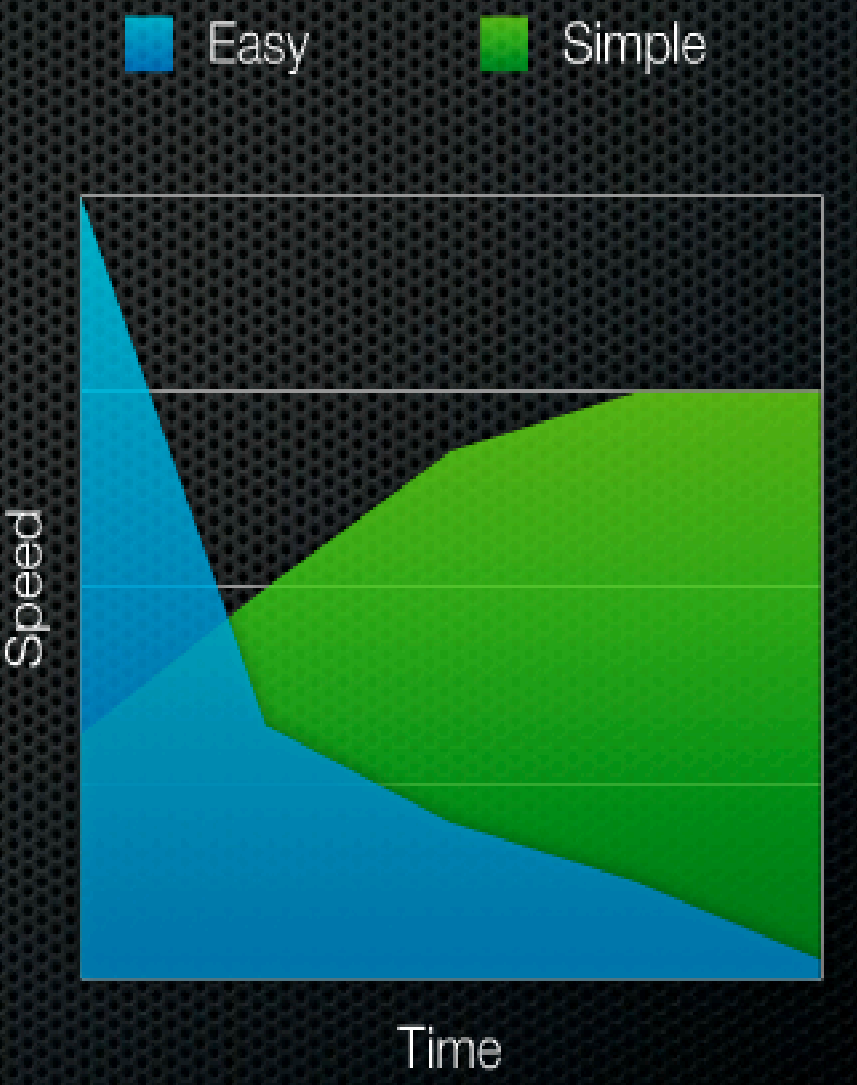
Debugging

- 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

www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/10.swf

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



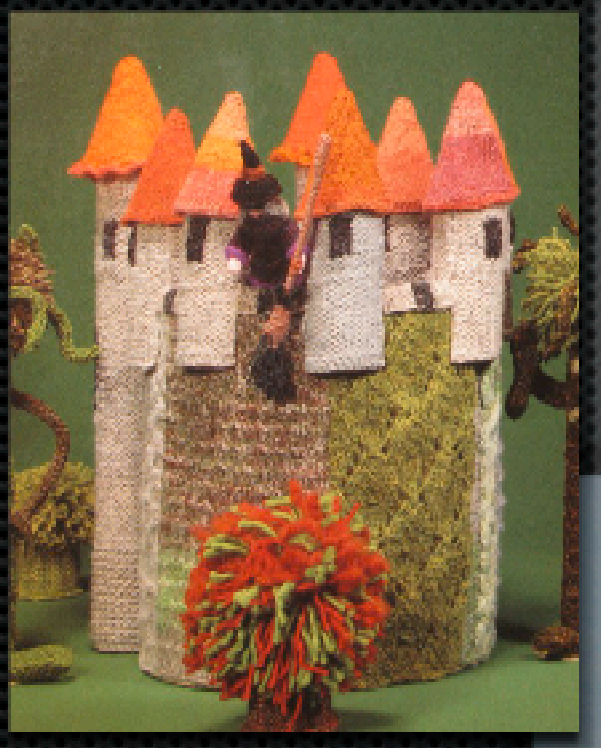
www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/11.swf

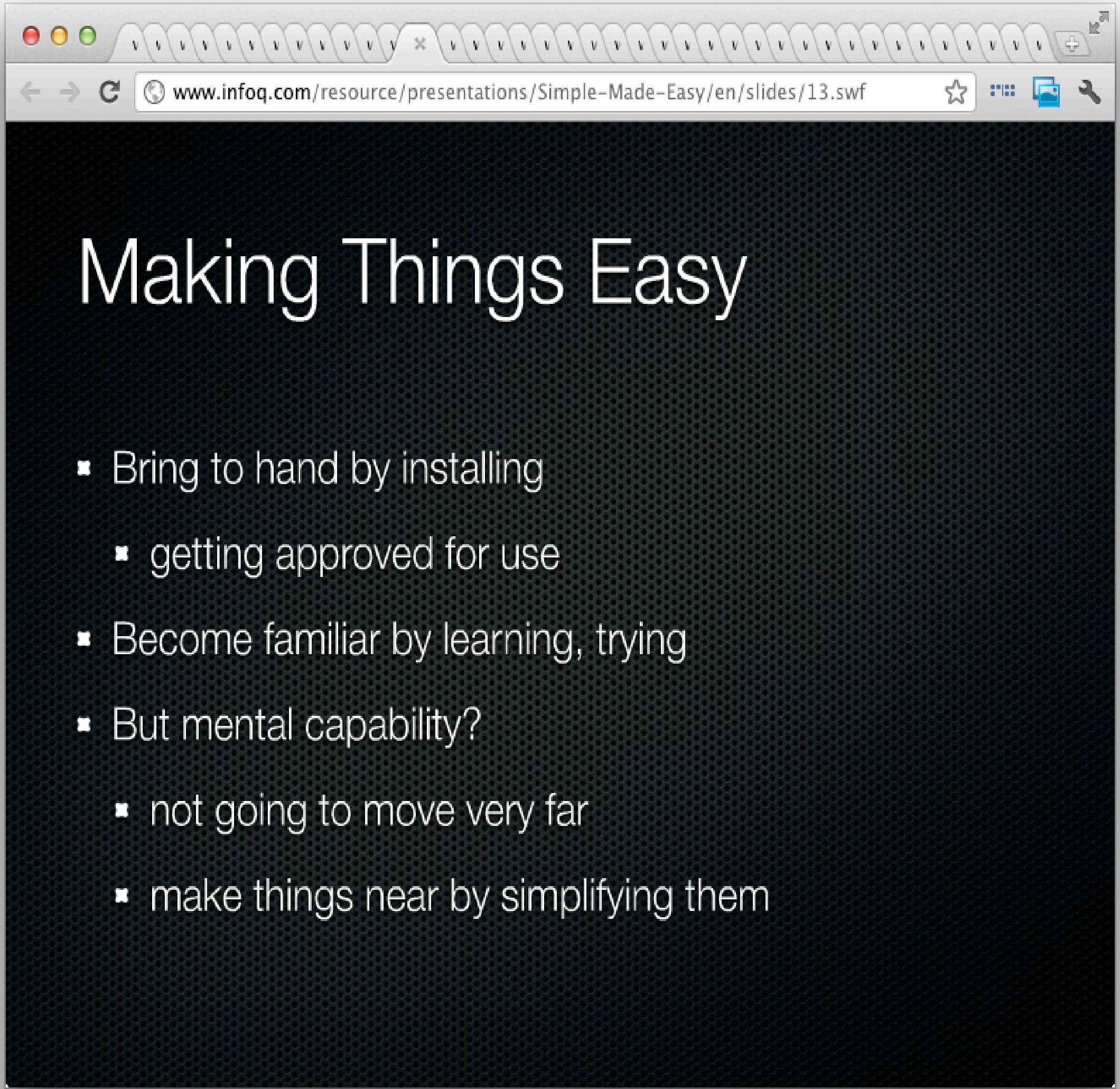
Easy Yet Complex?

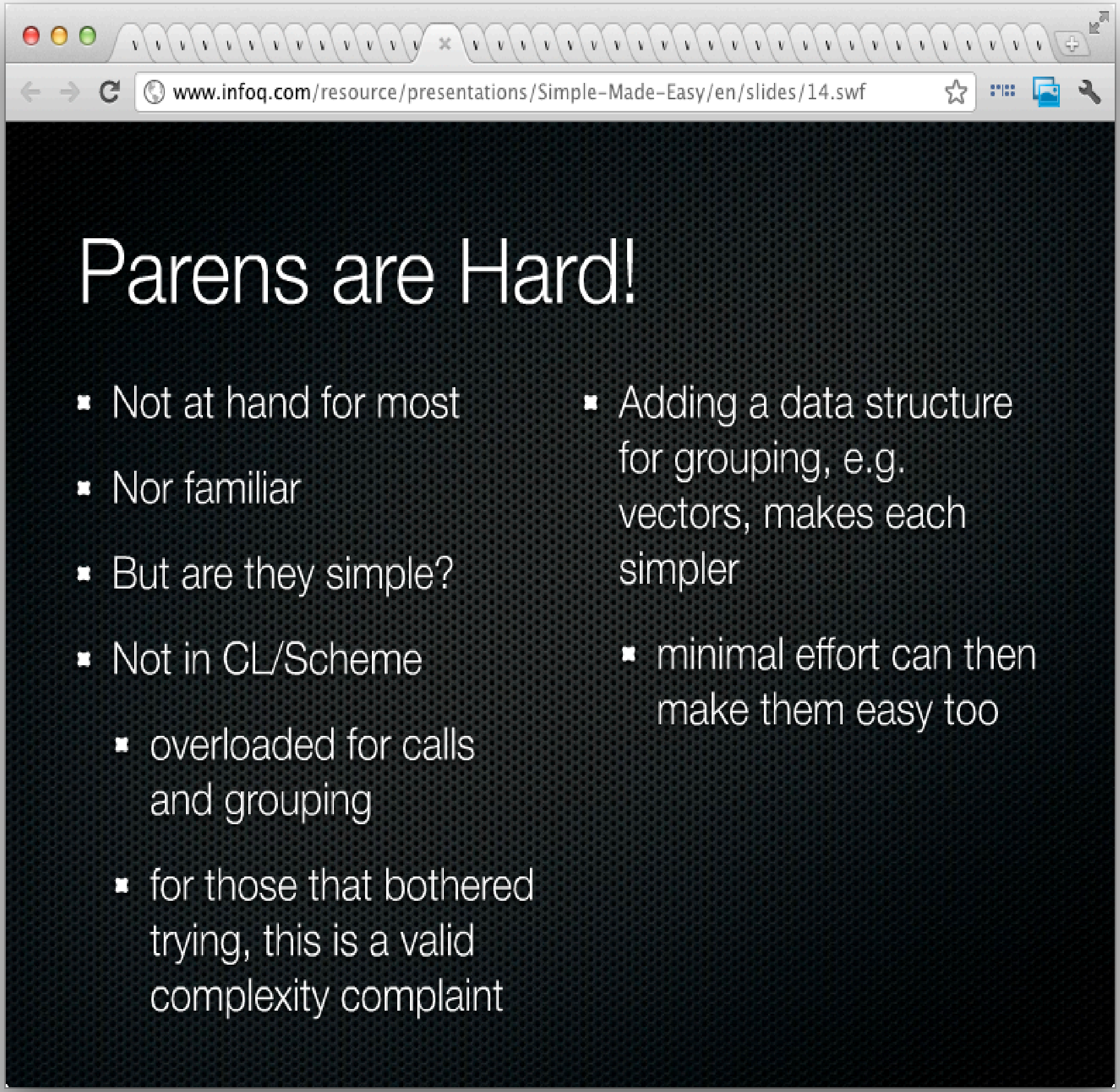
- Many complicating constructs are
  - Succinctly described
  - Familiar
  - Available
  - Easy to use
- What matters is the complexity they *yield*
  - Any such complexity is *incidental*

Benefits of Simplicity

- Ease understanding
- Ease of change
- Easier debugging
- Flexibility
  - policy
  - location etc



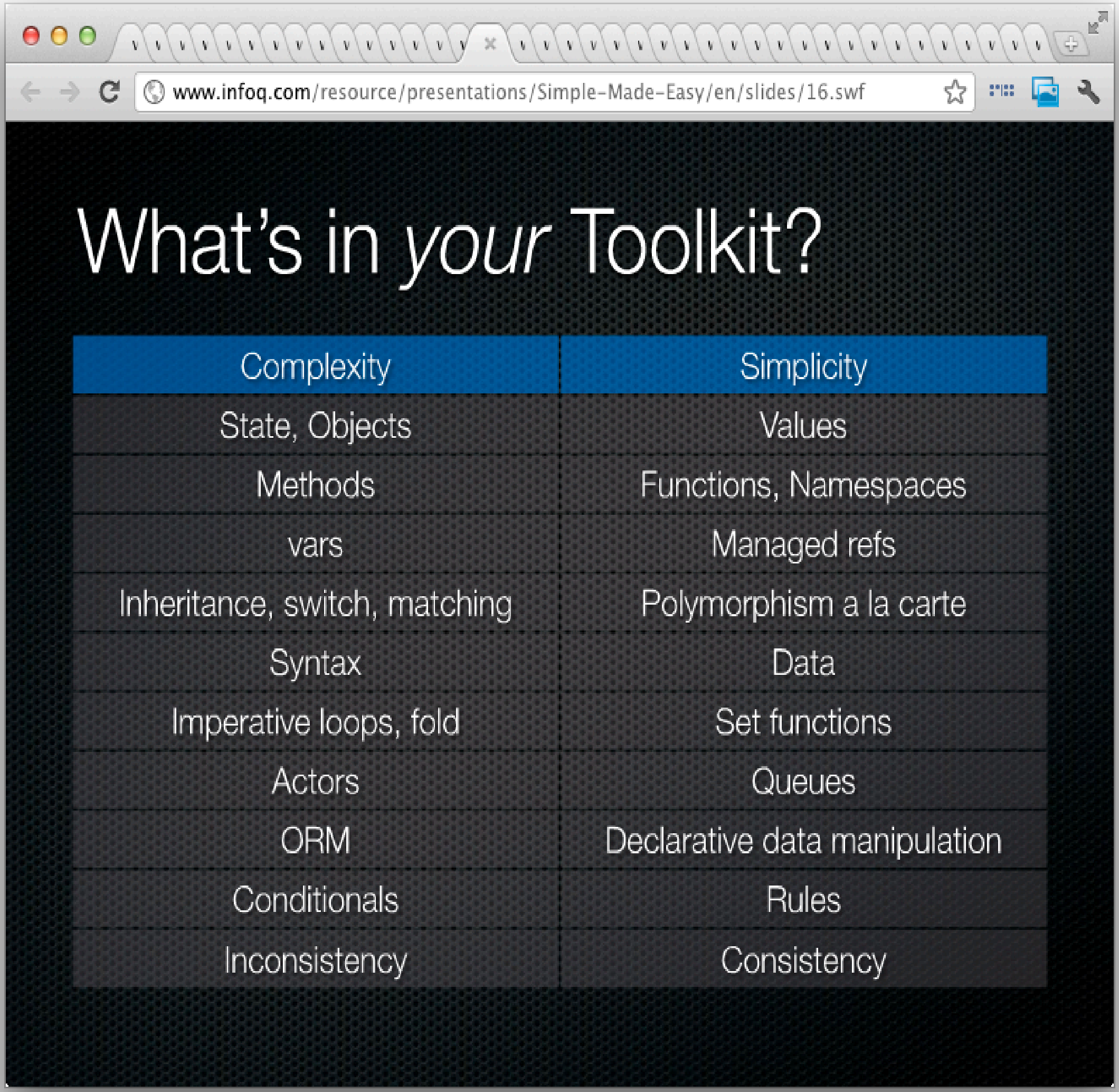

www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/13.swf
Making Things Easy
Bring to hand by installing
getting approved for use
Become familiar by learning, trying
But mental capability?
not going to move very far
make things near by simplifying them


www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/14.swf
Parens 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

www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/15.swf

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

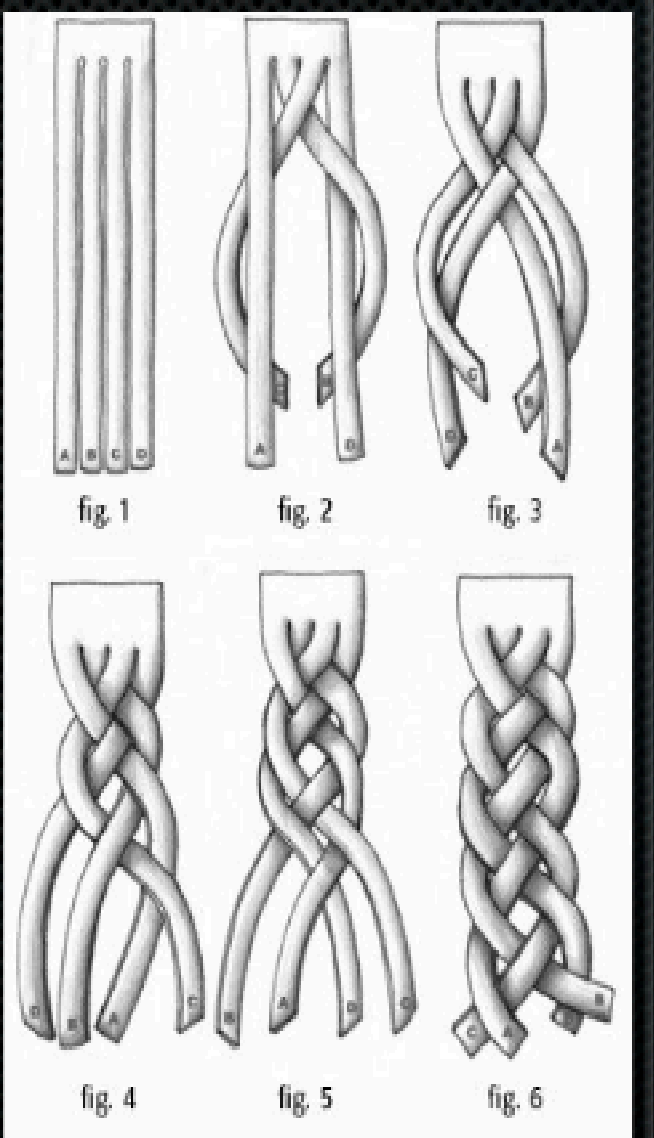
Alan Perlis


www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/16.swf
What's in *your* Toolkit?
Complexity
Simplicity
State, Objects
Values
Methods
Functions, Namespaces
vars
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

www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/17.swf

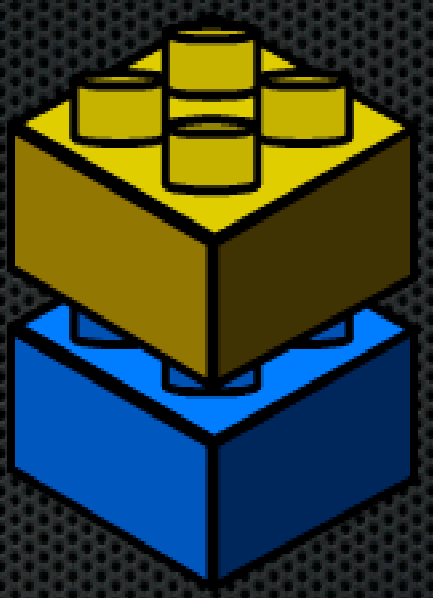
Complect

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

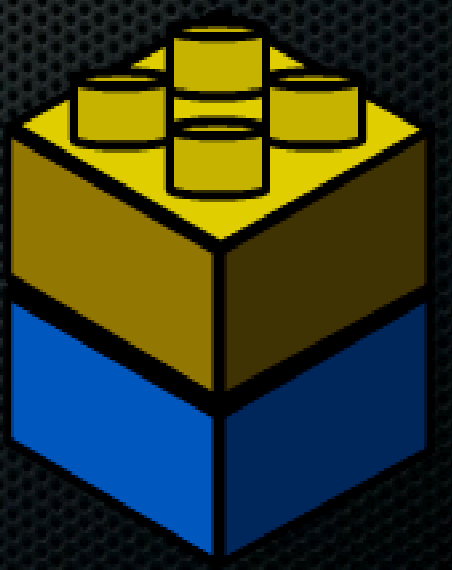


www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/18.swf

Compose

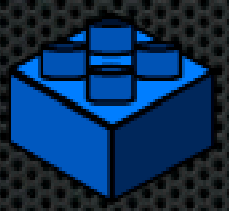


- To place together
- Composing simple components is the key to robust software



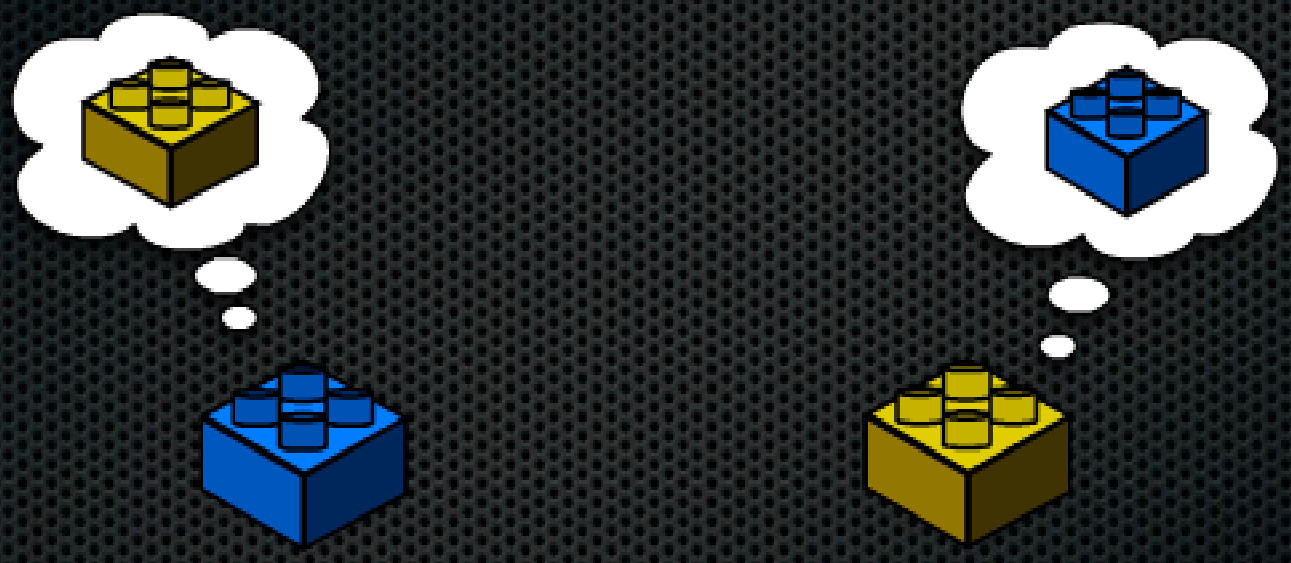
www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/19.swf

Modularity and Simplicity



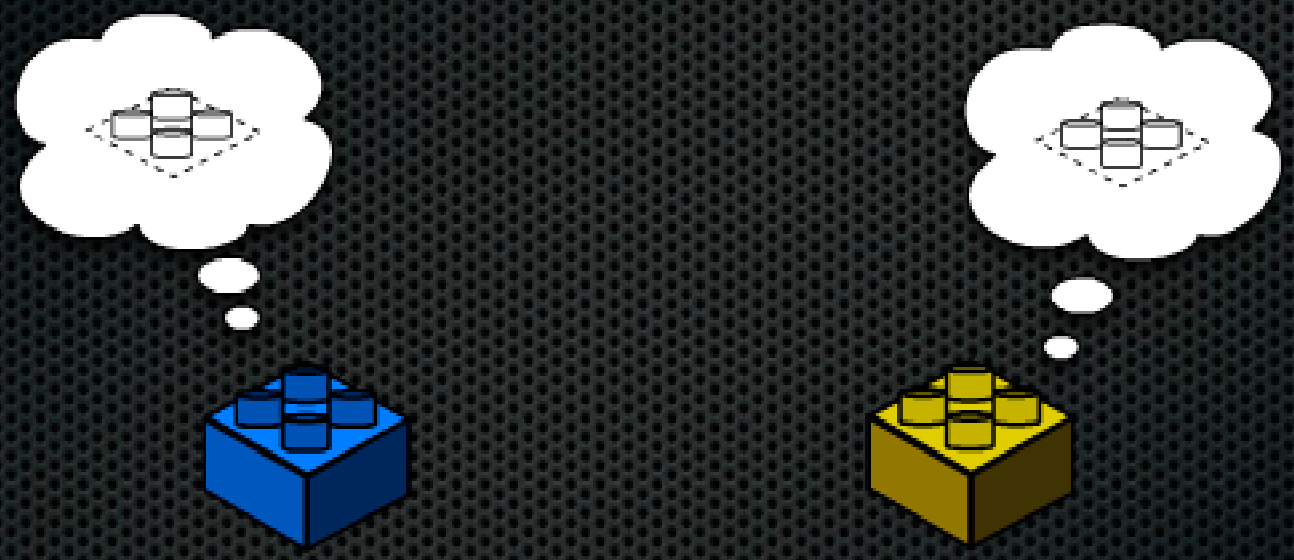
www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/20.swf

Modularity and Simplicity



www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/21.swf

Modularity and Simplicity

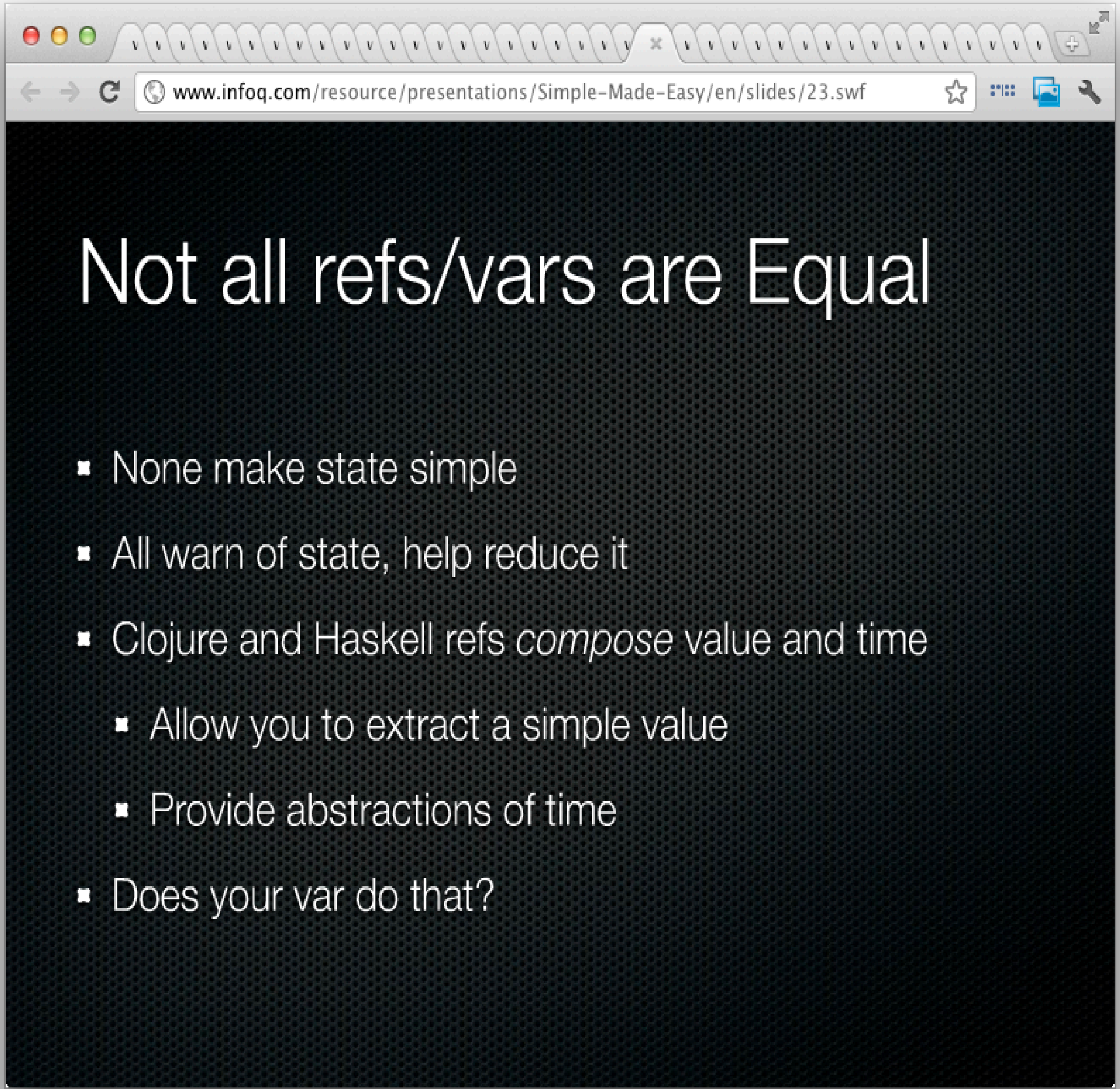


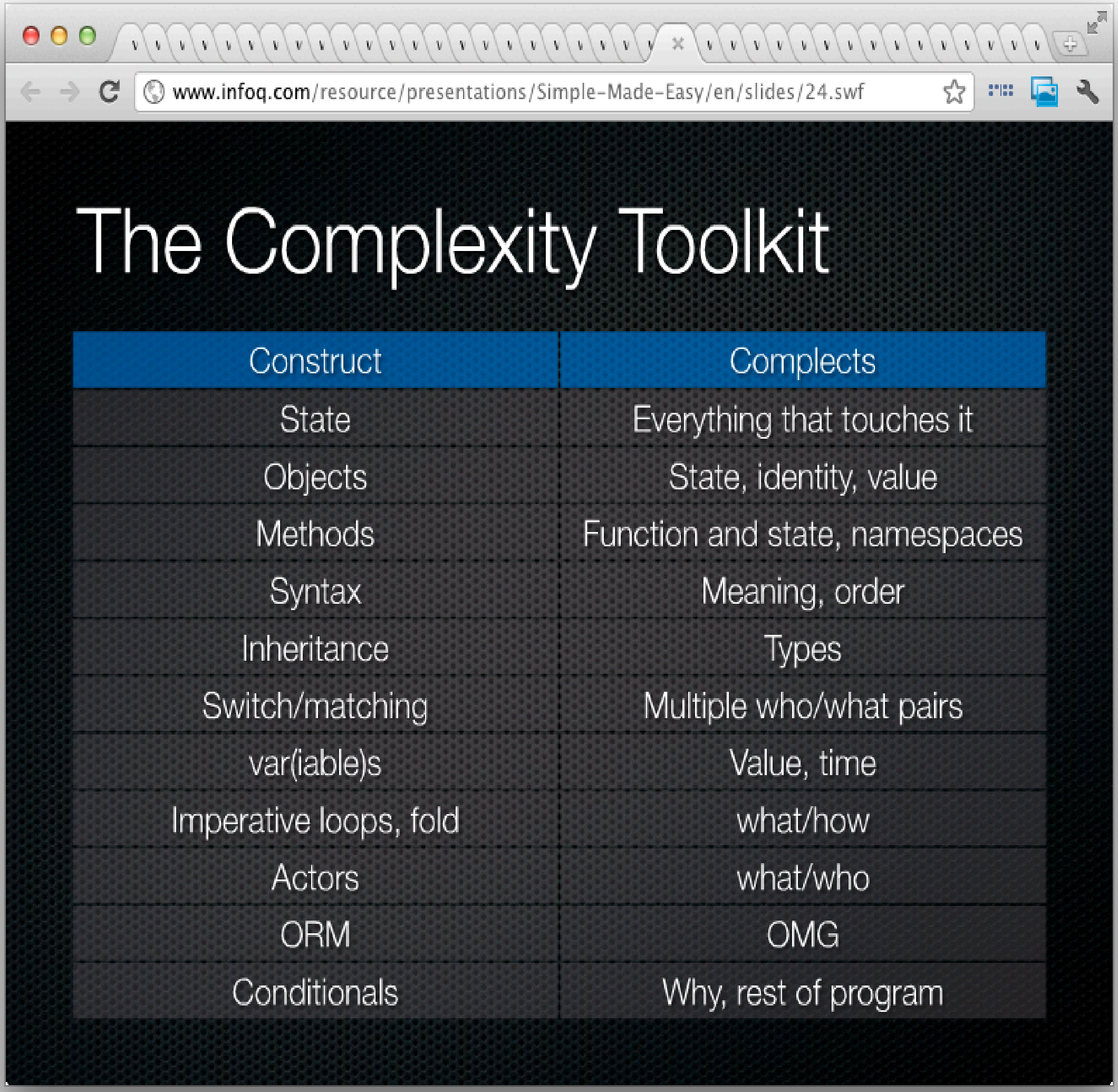
- Partitioning and stratification don't imply simplicity
  - but *are* enabled by it
- Don’t be fooled by code organization

www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/22.swf

State is Never Simple

- Complects 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


www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/23.swf
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?


www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/24.swf
The Complexity Toolkit
Construct
Complects
State
Everything that touches it
Objects
State, identity, value
Methods
Function and state, namespaces
Syntax
Meaning, order
Inheritance
Types
Switch/matching
Multiple who/what pairs
var(iable)s
Value, time
Imperative loops, fold
what/how
Actors
what/who
ORM
OMG
Conditionals
Why, rest of program

www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/25.swf

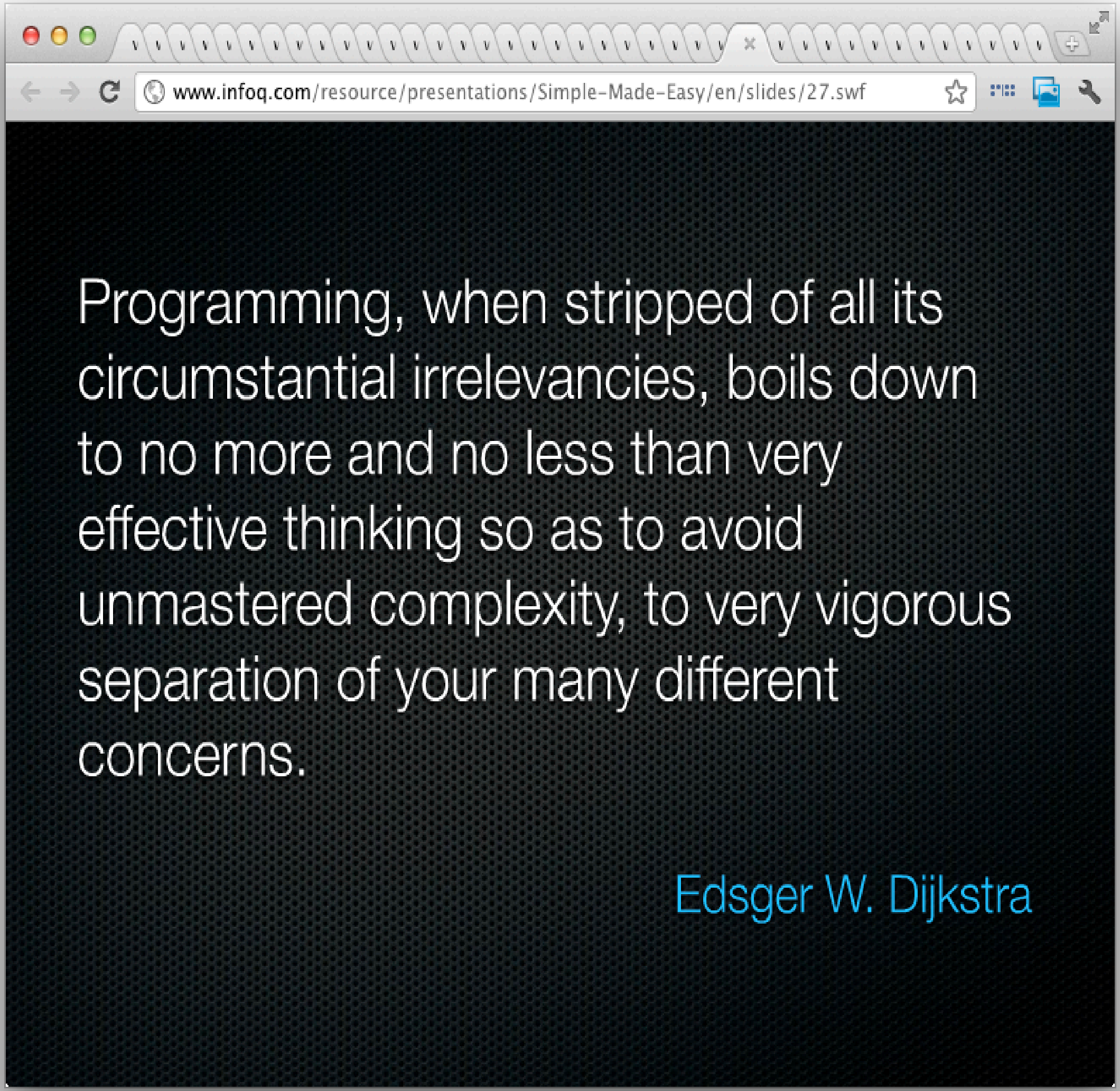
The Simplicity Toolkit

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

www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/26.swf

Environmental Complexity

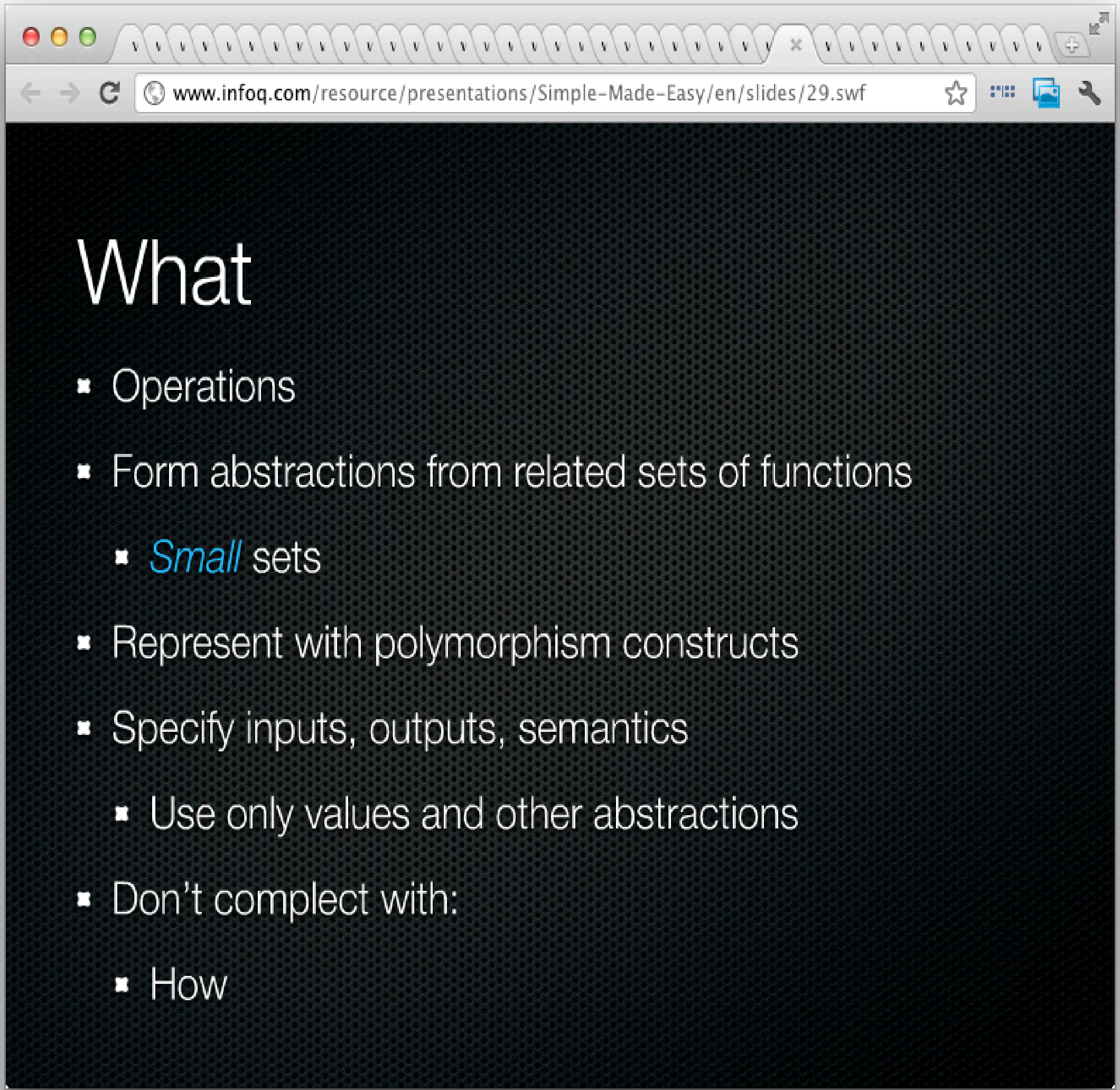
- Resources, e.g. memory, CPU
- *Inherent* complexity in implementation space
  - All components contend for them
- Segmentation
  - waste
- Individual policies don't compose
  - just make things more complex

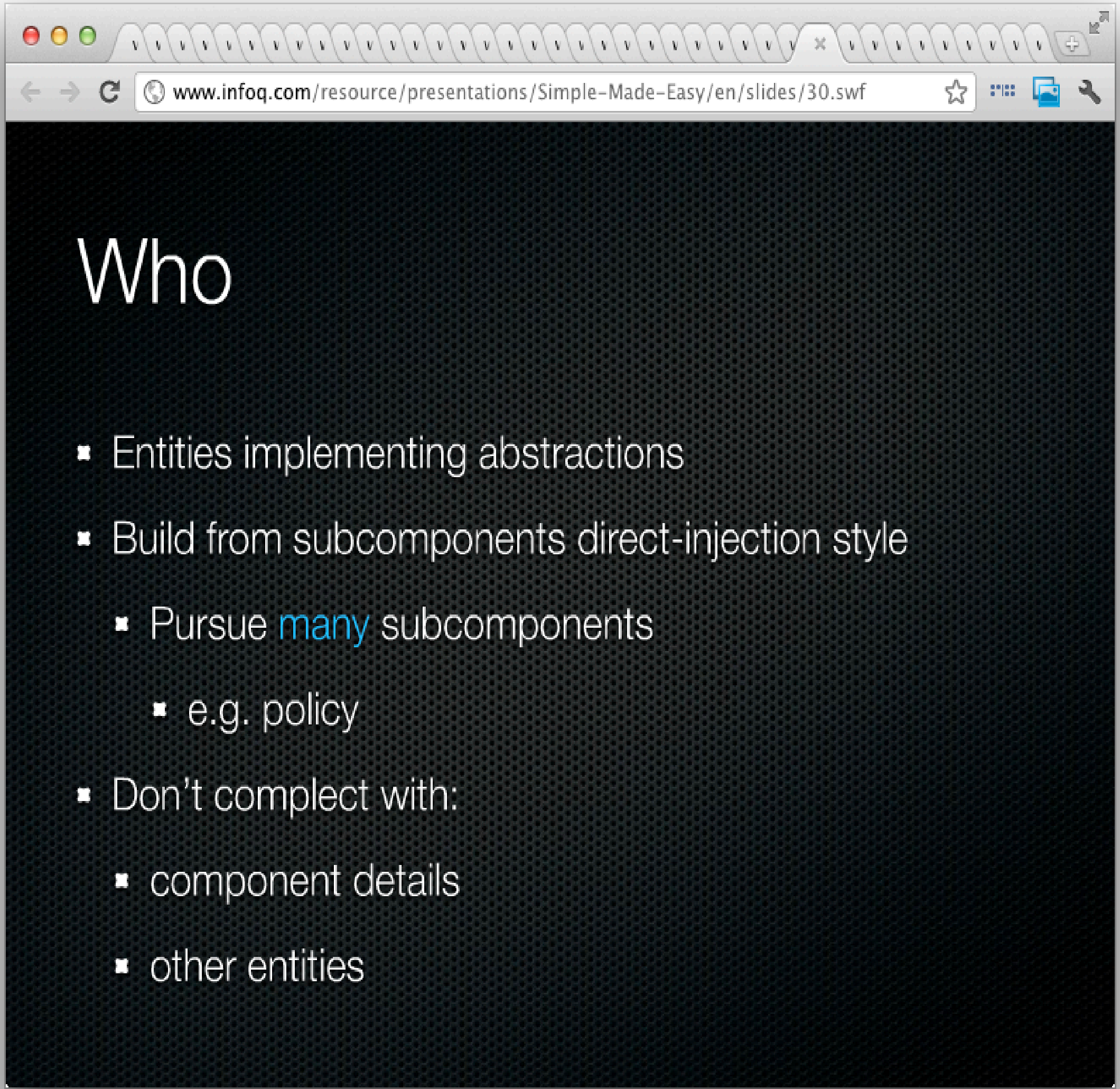

www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/27.swf
Programming, when stripped of all its circumstantial irrelevancies, boils down to no more and no less than very effective thinking so as to avoid unmastered complexity, to very vigorous separation of your many different concerns.
Edsger W. Dijkstra

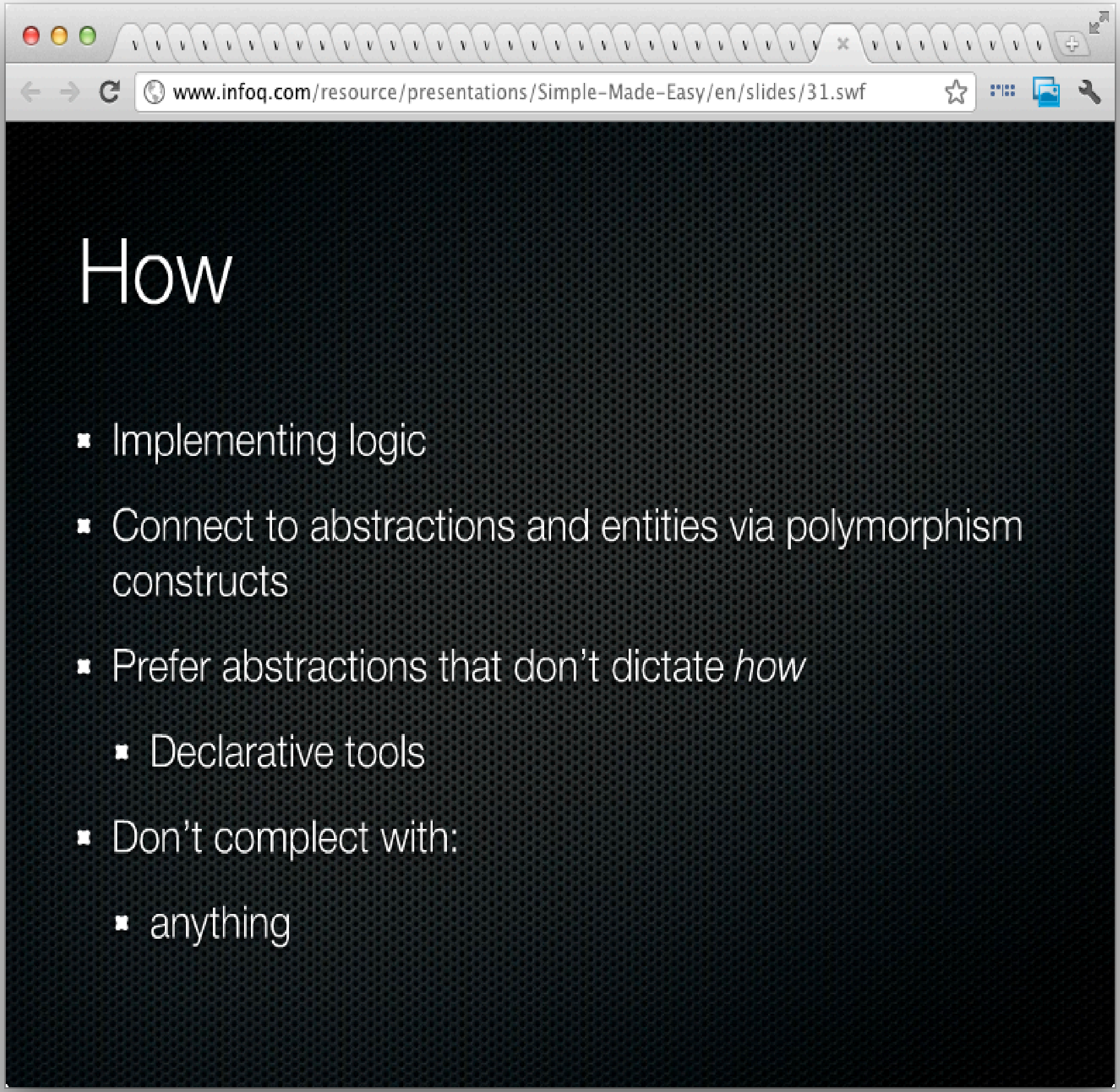
www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/28.swf

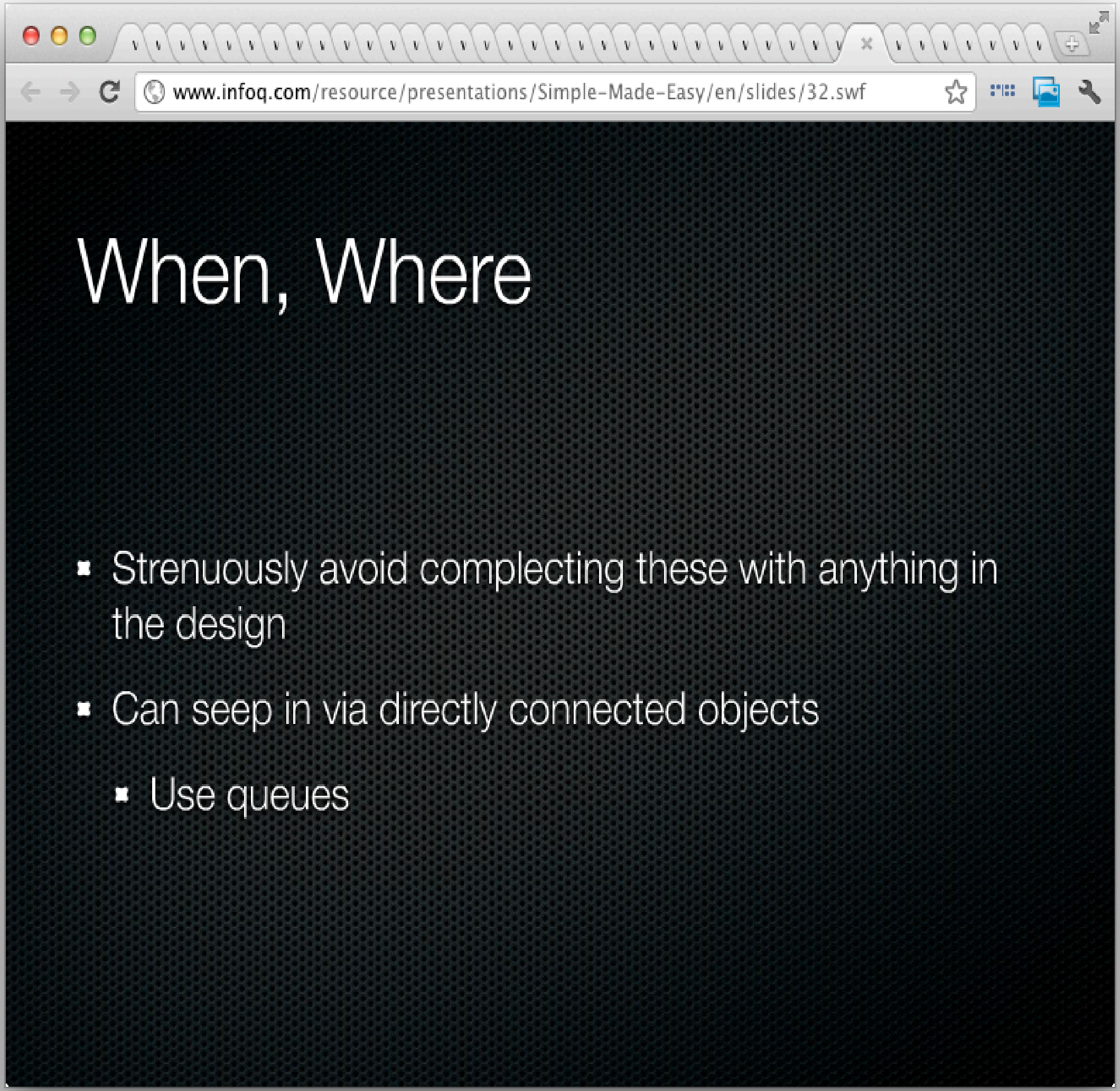
Abstraction for Simplicity

- Abstract
  - drawn away
- vs Abstraction as complexity *hiding*
- Who, What, When, Where, Why and How
- I don’t know, I don’t want to know


www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/29.swf
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


www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/30.swf
Who
Entities implementing abstractions
Build from subcomponents direct-injection style
Pursue many subcomponents
e.g. policy
Don't complect with:
component details
other entities


www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/31.swf
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


www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/32.swf
When, Where
Strenuously avoid complecting these with anything in the design
Can seep in via directly connected objects
Use queues

www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/33.swf

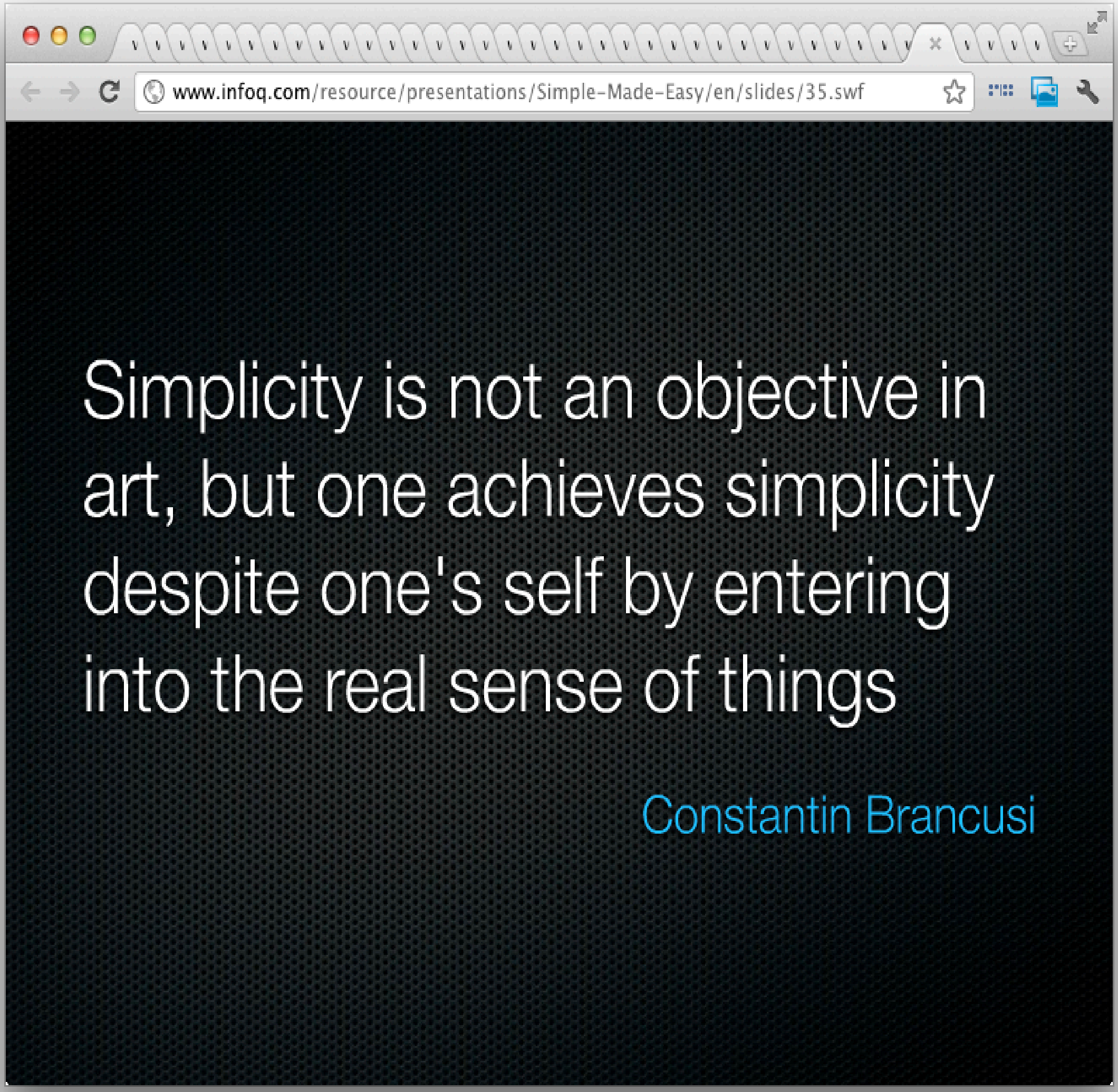
Why

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

www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/34.swf

Information *is* Simple

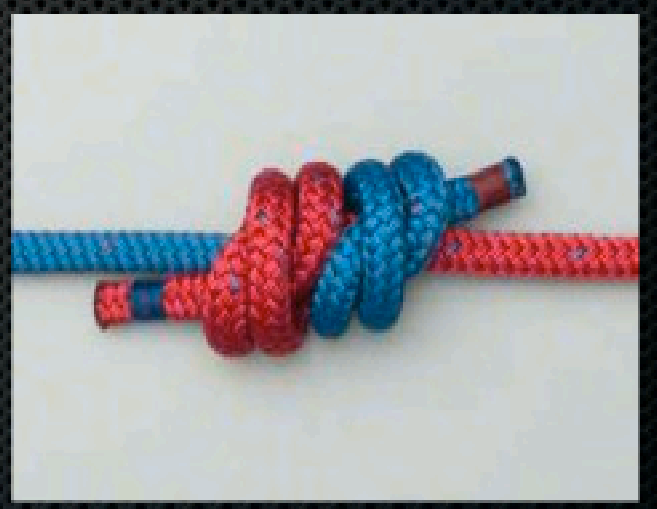
- Don't ruin it
- By hiding it behind a micro-language
  - i.e. a class with information-specific methods
  - thwarts generic data composition
  - ties logic to representation du jour
- Represent data as data

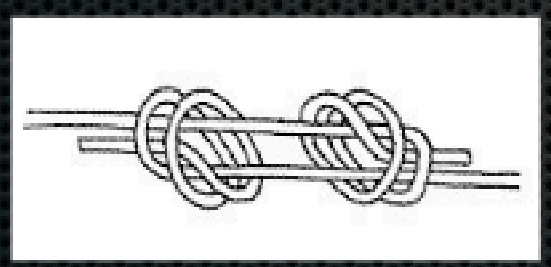

www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/35.swf
Simplicity is not an objective in art, but one achieves simplicity despite one's self by entering into the real sense of things
Constantin Brancusi

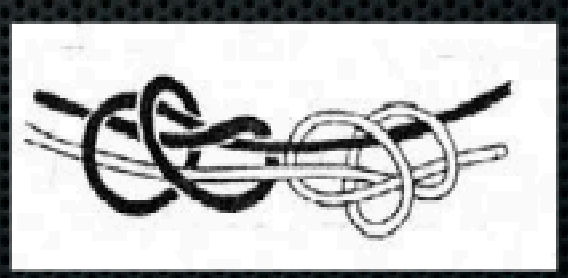
www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/36.swf

Simplifying

- Identifying individual threads/roles/dimensions
- Following through the user story/code
- Disentangling







www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/37.swf

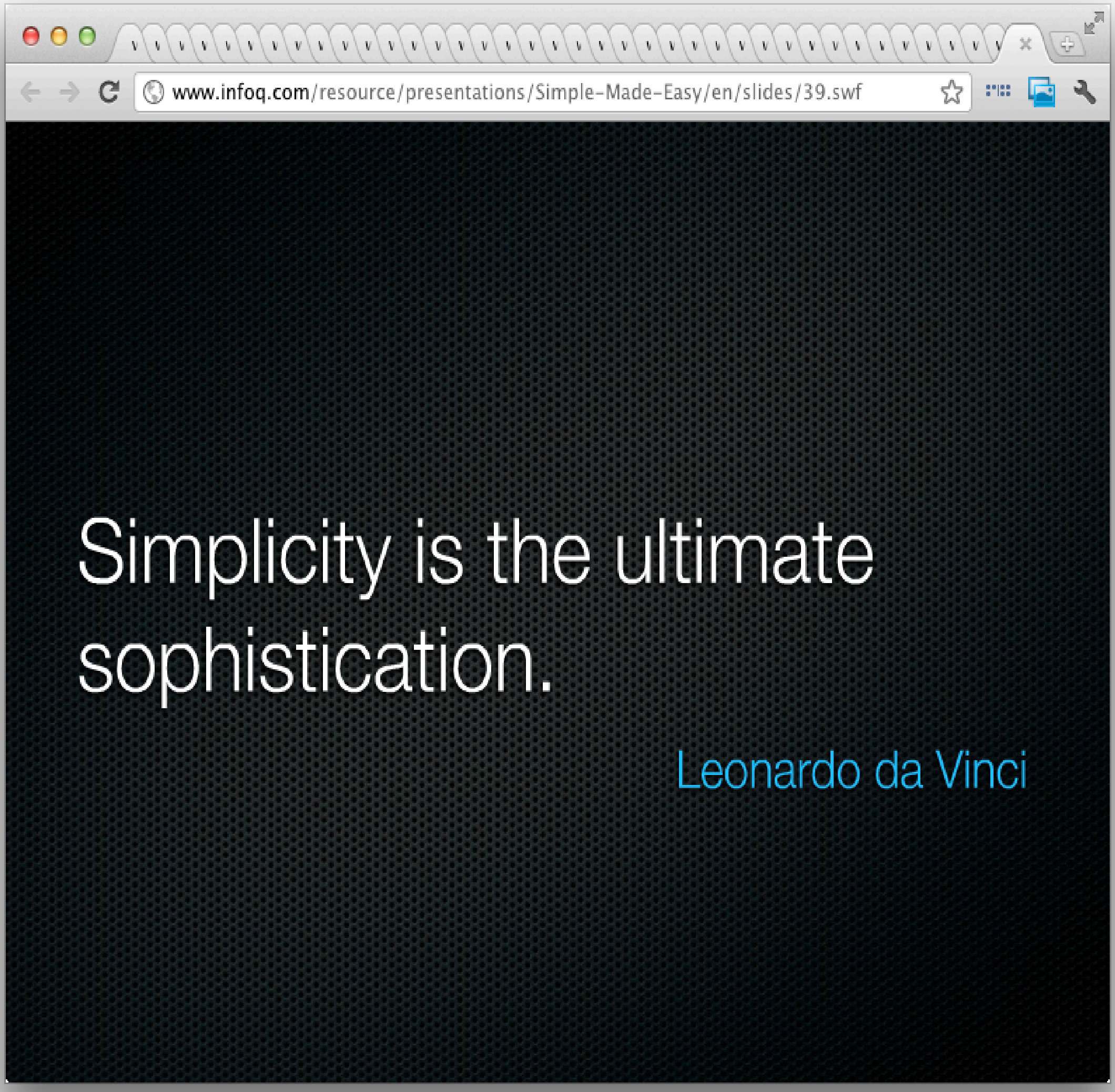
Simplicity is a Choice

- 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

www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/38.swf

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!


www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/39.swf
Simplicity is the ultimate sophistication.
Leonardo da Vinci