

The image shows a presentation slide titled "Word Origins" with a dark textured background. The slide compares the etymological roots of the words "Simple" and "Easy".

- Simple
 - sim- plex*
 - one fold/braid
 - vs complex
- Easy
 - ease < aise < adjacens*
 - lie near
 - vs hard

The slide is displayed in a web browser window with the URL www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/3.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*

The image shows a screenshot of a web browser window. The title bar indicates the URL is www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/5.swf. The main content area features a large, bold, white sans-serif font word 'Easy' centered on a dark gray textured background. Below the title, there are two columns of bullet points in white text. The left column contains five items, and the right column contains three items. The text is as follows:

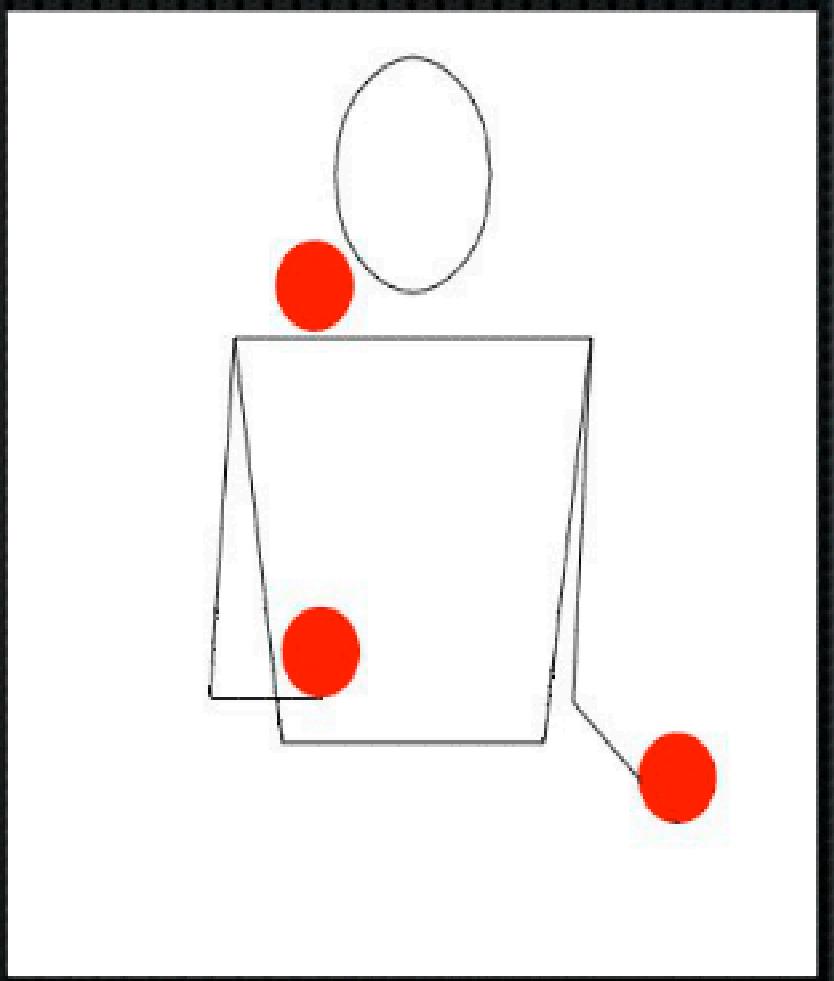
- Near, at hand
 - on our hard drive, in our tool set, IDE, apt get, gem install...
- Near to our understanding/skill set
 - familiar
- Near our capabilities
- *Easy is relative*

The image shows a screenshot of a web browser window. The title bar at the top displays the URL: www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/6.swf. The main content area of the browser contains a presentation slide with a dark textured background. The title of the slide is "Construct vs Artifact". Below the title, there is a bulleted list of points:

- 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



The image shows a screenshot of a web browser window. The title bar at the top displays the URL www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/8.swf. The main content area of the browser features a large, bold, white sans-serif font word "Change" centered on a dark gray background with a subtle grid texture. Below the title, there is a bulleted list of six items, each preceded by a black square bullet point:

- 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

Speed

Time

Easy Simple

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*



www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/12.swf

Benefits of Simplicity

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

The slide features two images side-by-side. The left image shows a colorful, hand-made clay castle with multiple towers and orange roofs, surrounded by greenery and a red flower bush. The right image shows a simple, light-colored LEGO castle with a single tower and a flag, featuring a knight on horseback in the foreground.

www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/13.swf

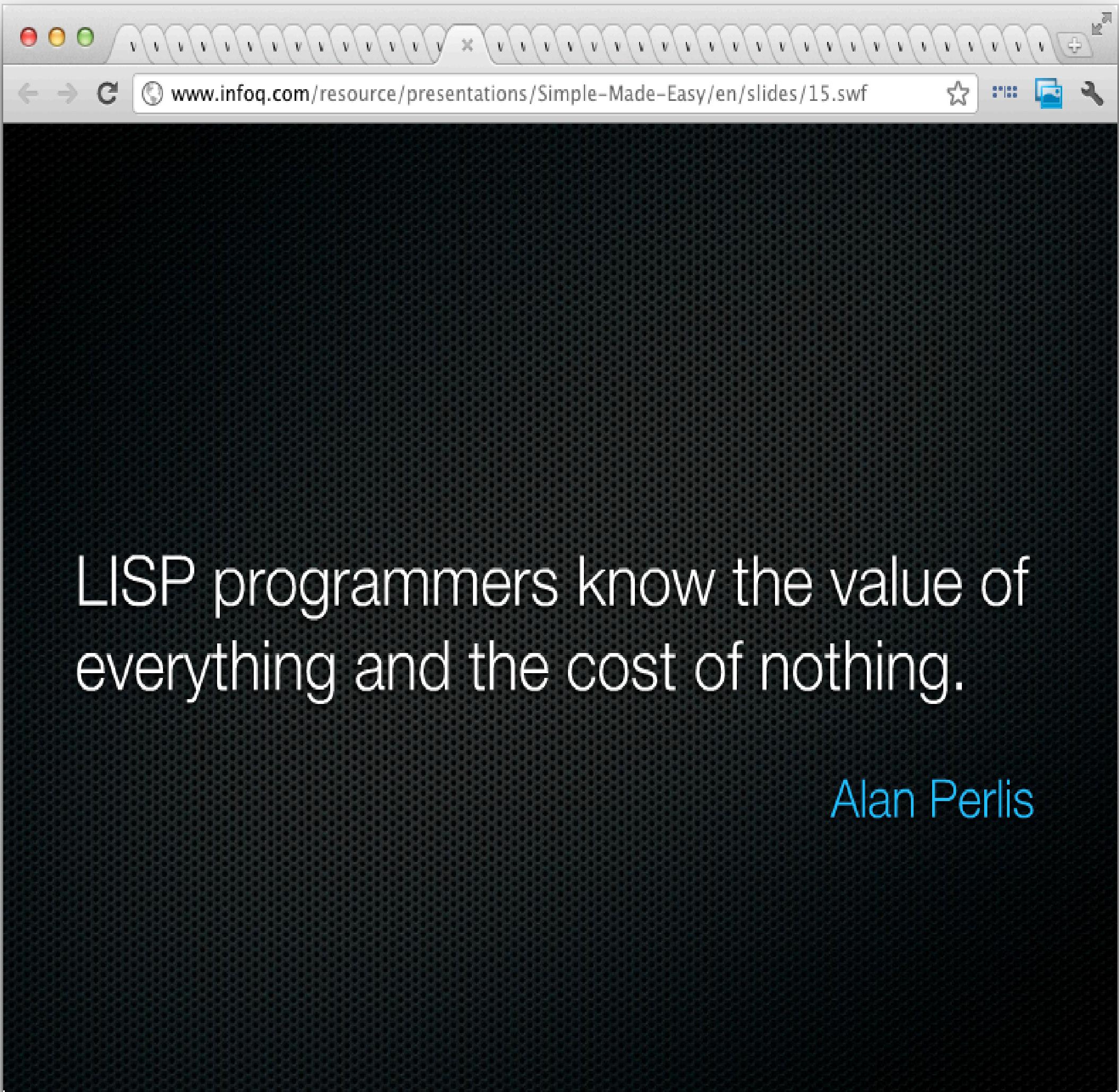
Making Things Easy

- 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

The image shows a screenshot of a web browser window. The title bar at the top displays the URL: www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/14.swf. The main content area of the browser contains a presentation slide with a dark textured background. The slide has a large, bold title "Parentheses are Hard!" in white. Below the title is a bulleted list of points, also in white, discussing the challenges and solutions related to parentheses.

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/16.swf

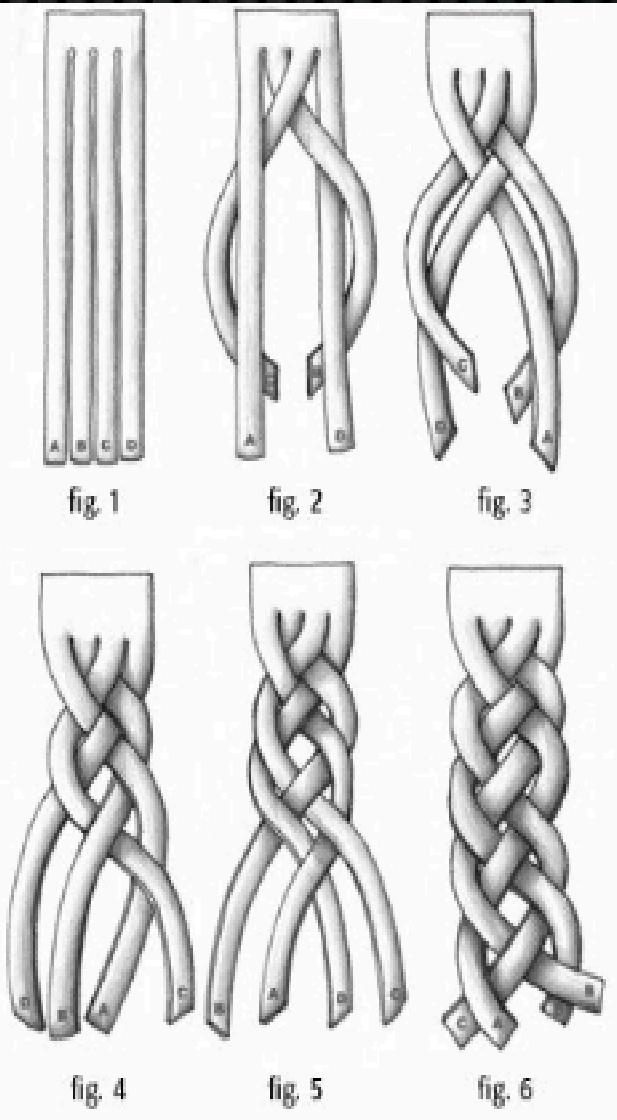
What's in your Toolkit?

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



Complect

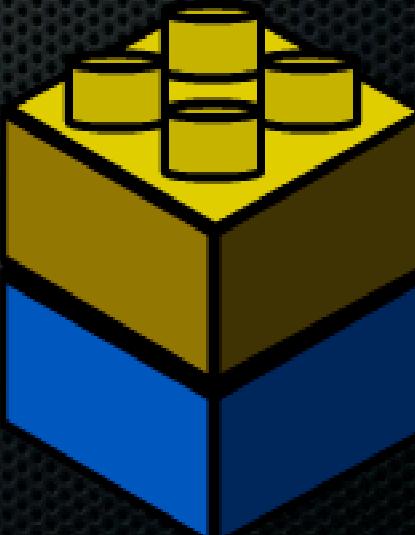
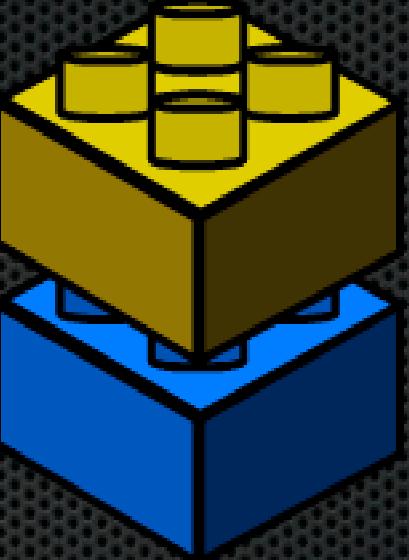
- To interleave, entwine, braid
 - archaic
- Don't do it!
 - Complecting things is the source of complexity
- Best to avoid in the first place

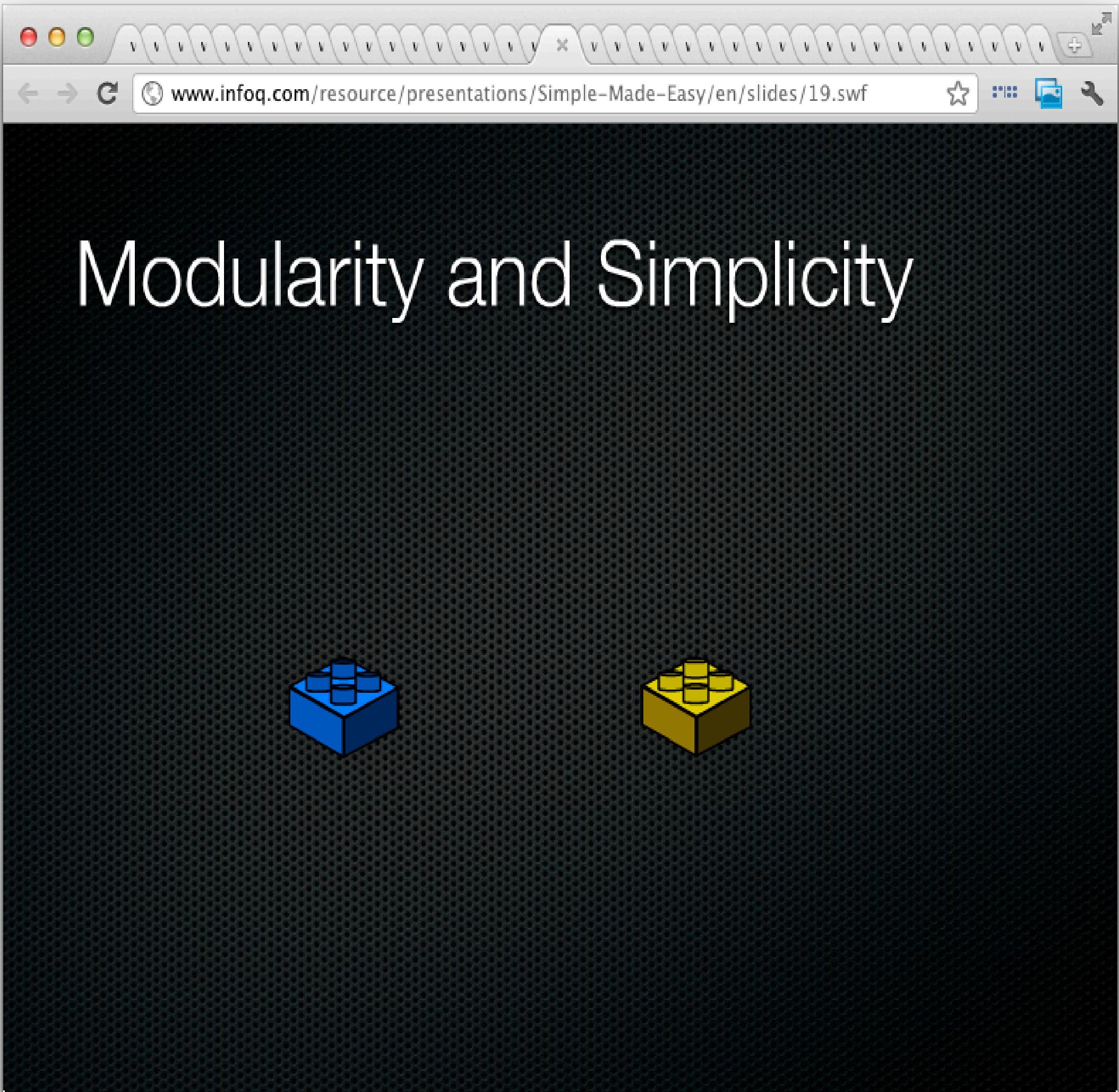


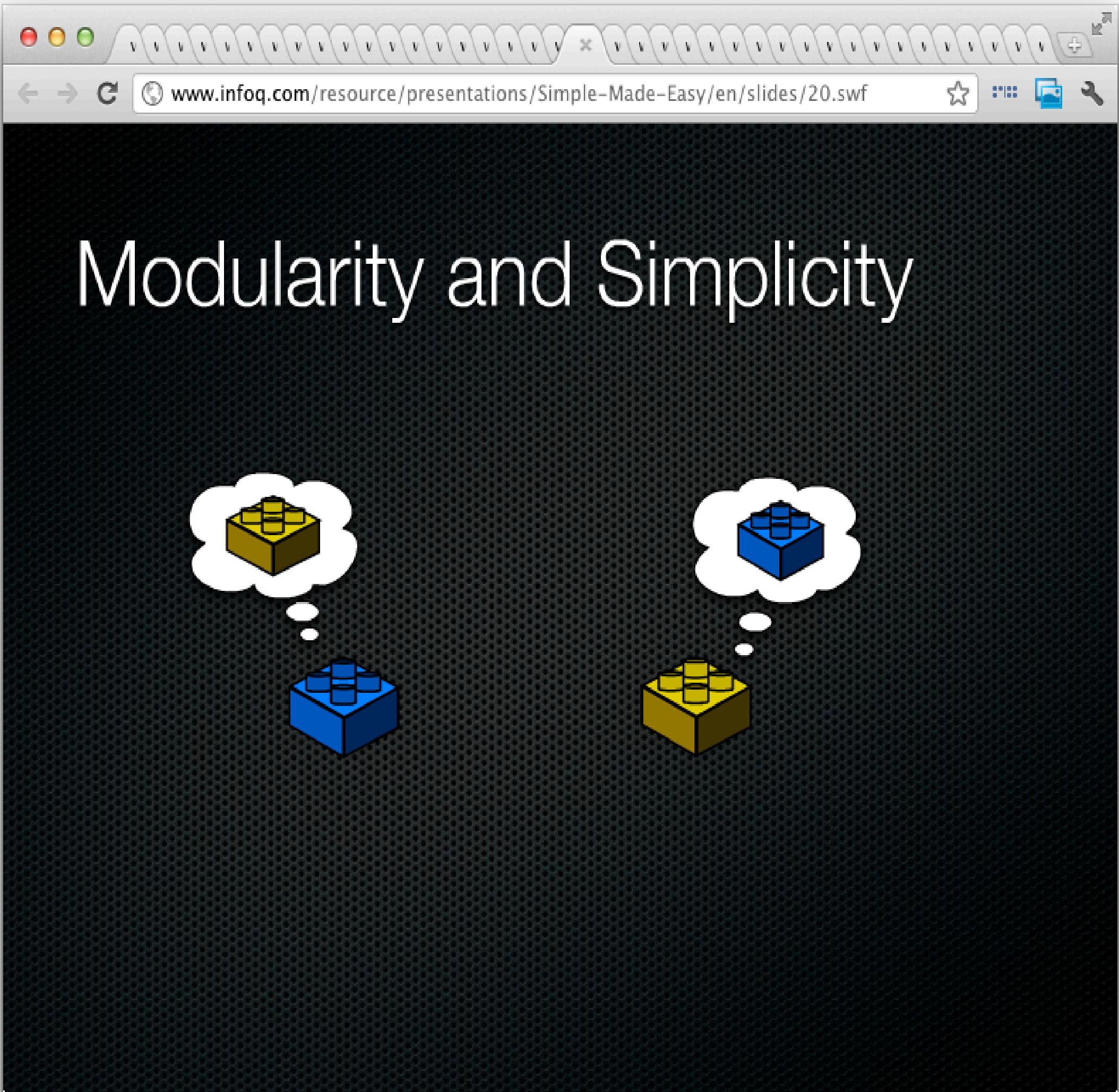
www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/18.swf

Compose

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

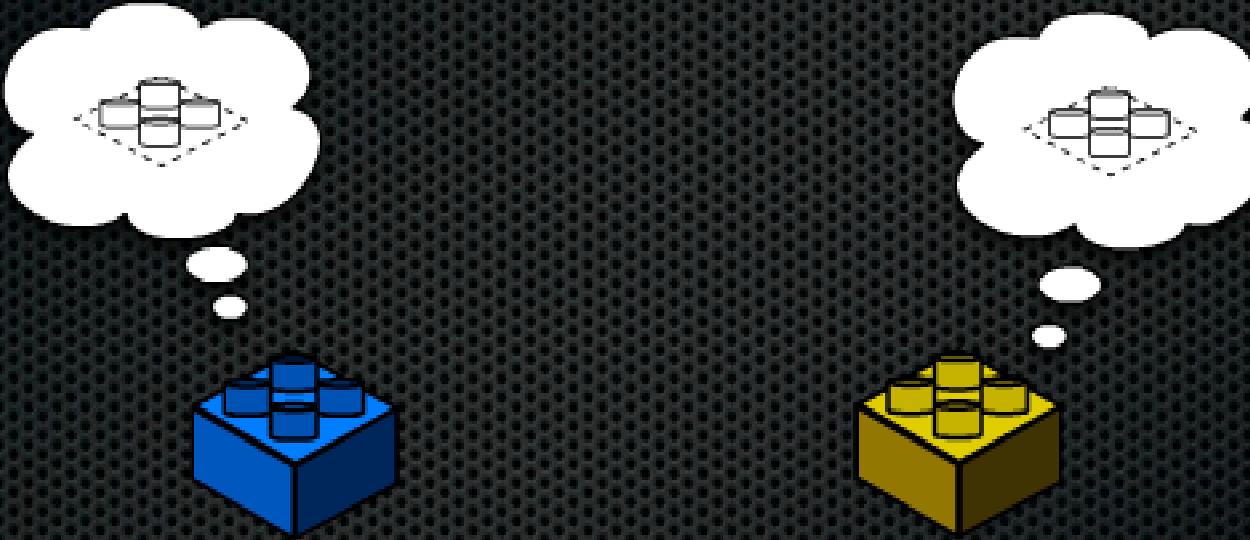






www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/21.swf

Modularity and Simplicity



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

The image shows a screenshot of a web browser window. The address bar at the top displays the URL www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/22.swf. The main content area of the browser features a large, bold title "State is Never Simple" centered on a dark, textured background. Below the title, there is a bulleted list of points, which appears to be part of a presentation slide. The browser interface includes standard window controls (minimize, maximize, close) and a toolbar with various icons.

- 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

The image shows a screenshot of a presentation slide. At the top, there is a browser-style header with a red, yellow, and green window control buttons on the left. The address bar contains the URL www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/23.swf. To the right of the address bar are several icons: a star, a grid, a document, and a wrench. The main content area has a dark, textured background. The title of the slide is "Not all refs/vars are Equal", displayed prominently in large white font. Below the title is a bulleted list of nine items, also in white font, describing various aspects of references and variables.

- 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

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

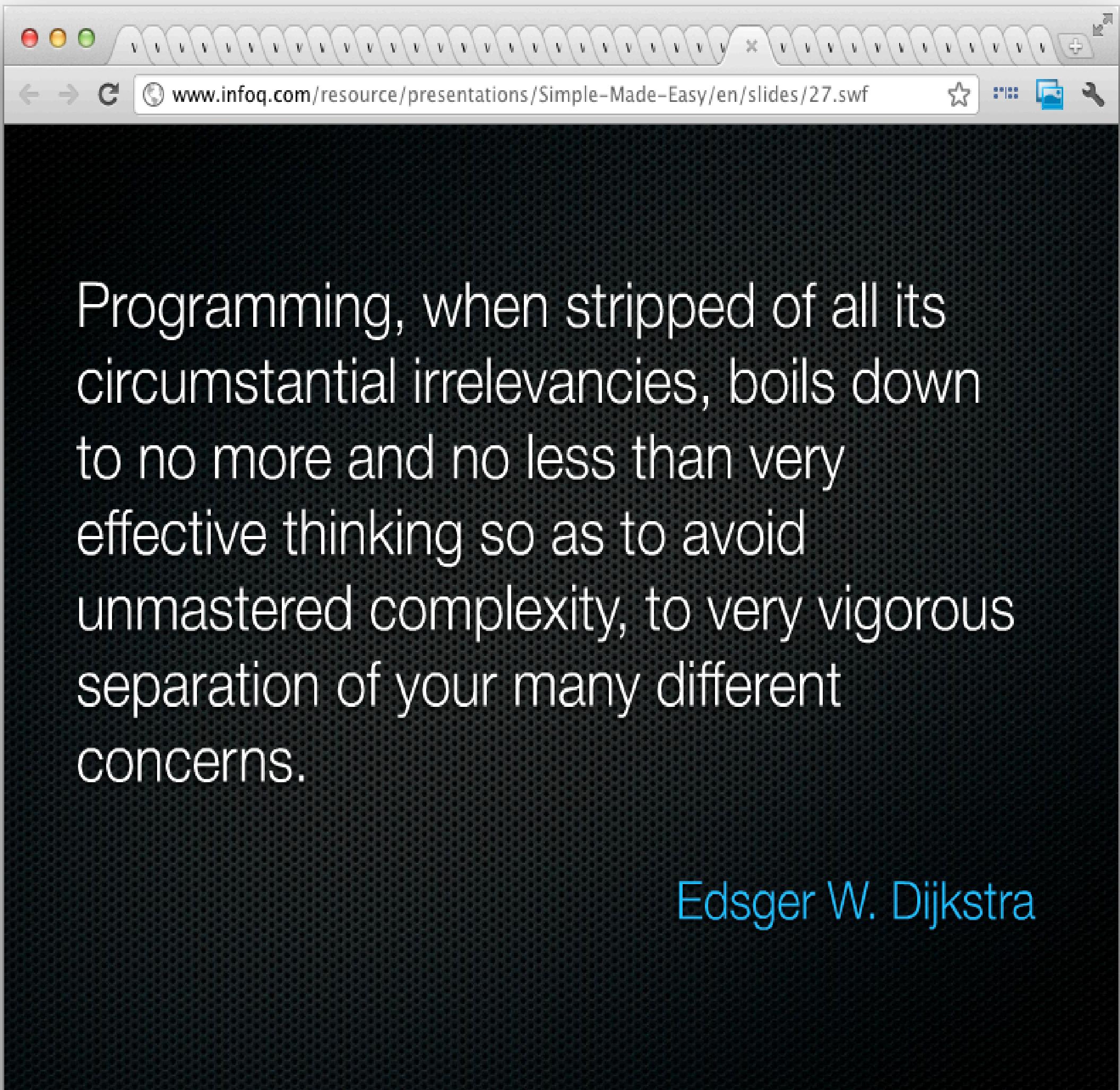
The screenshot shows a web browser window with a dark-themed slide from [www.infoq.com](http://www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/25.swf). The title of the slide is "The Simplicity Toolkit". The slide features a table comparing various programming constructs with their corresponding implementation details.

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



The image shows a screenshot of a web browser window. The address bar at the top displays the URL: www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/28.swf. The main content area of the browser features a large, bold title "Abstraction for Simplicity" centered on a dark, textured background. Below the title, there is a bulleted list of points, some of which are highlighted in green. The list includes:

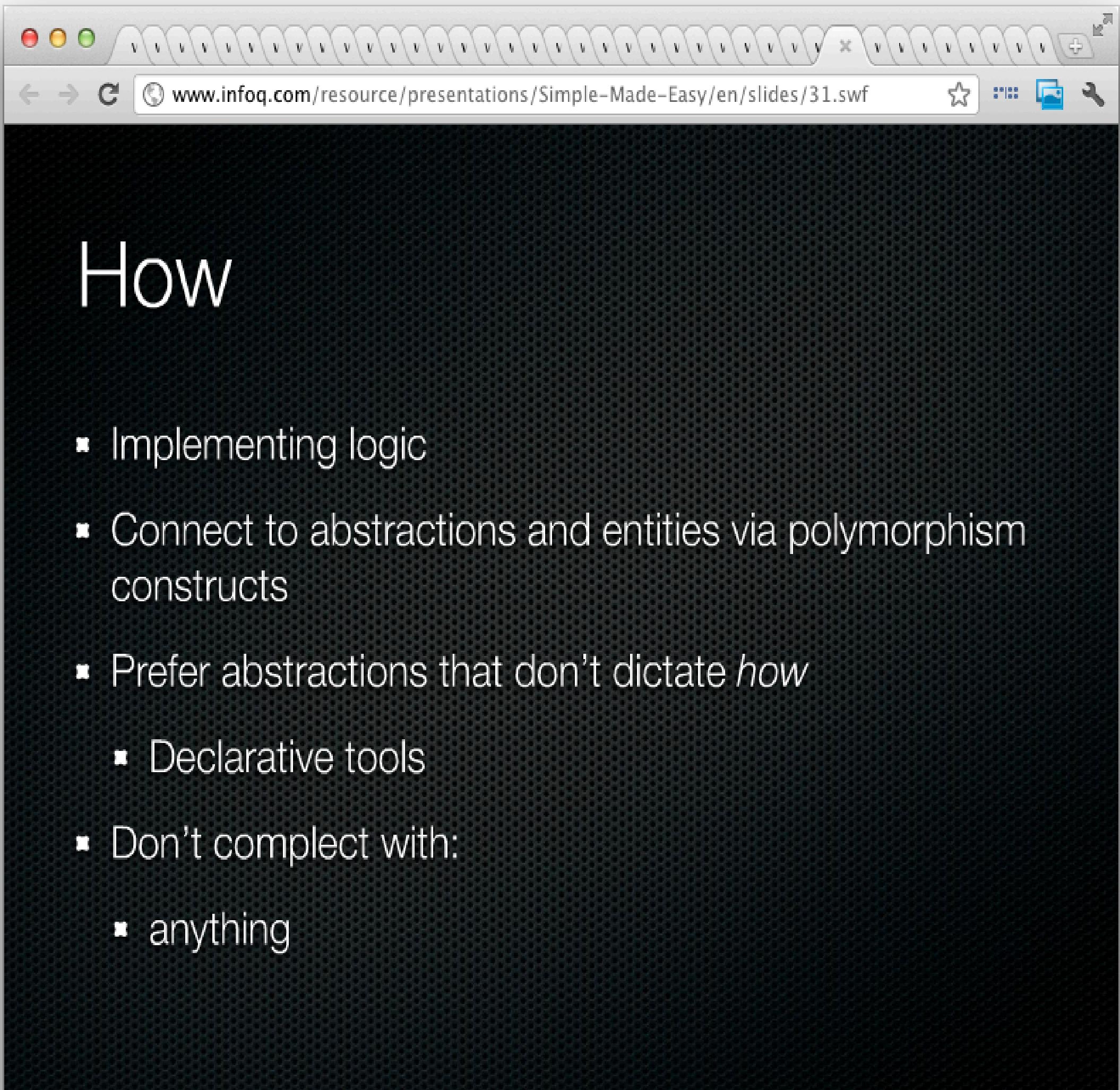
- Abstract
 - drawn away
- vs Abstraction as complexity *hiding*
- Who, What, When, Where, Why and How
- I don't know, I don't want to know

The image shows a screenshot of a presentation slide. The title 'What' is displayed prominently at the top left. Below the title is a bulleted list of 12 items, each preceded by a black square bullet point. The list includes various topics such as operations, abstractions, polymorphism, inputs/outputs, semantics, and completion. One item in the list, 'Small sets', is highlighted with blue text. The background of the slide has a dark, textured pattern.

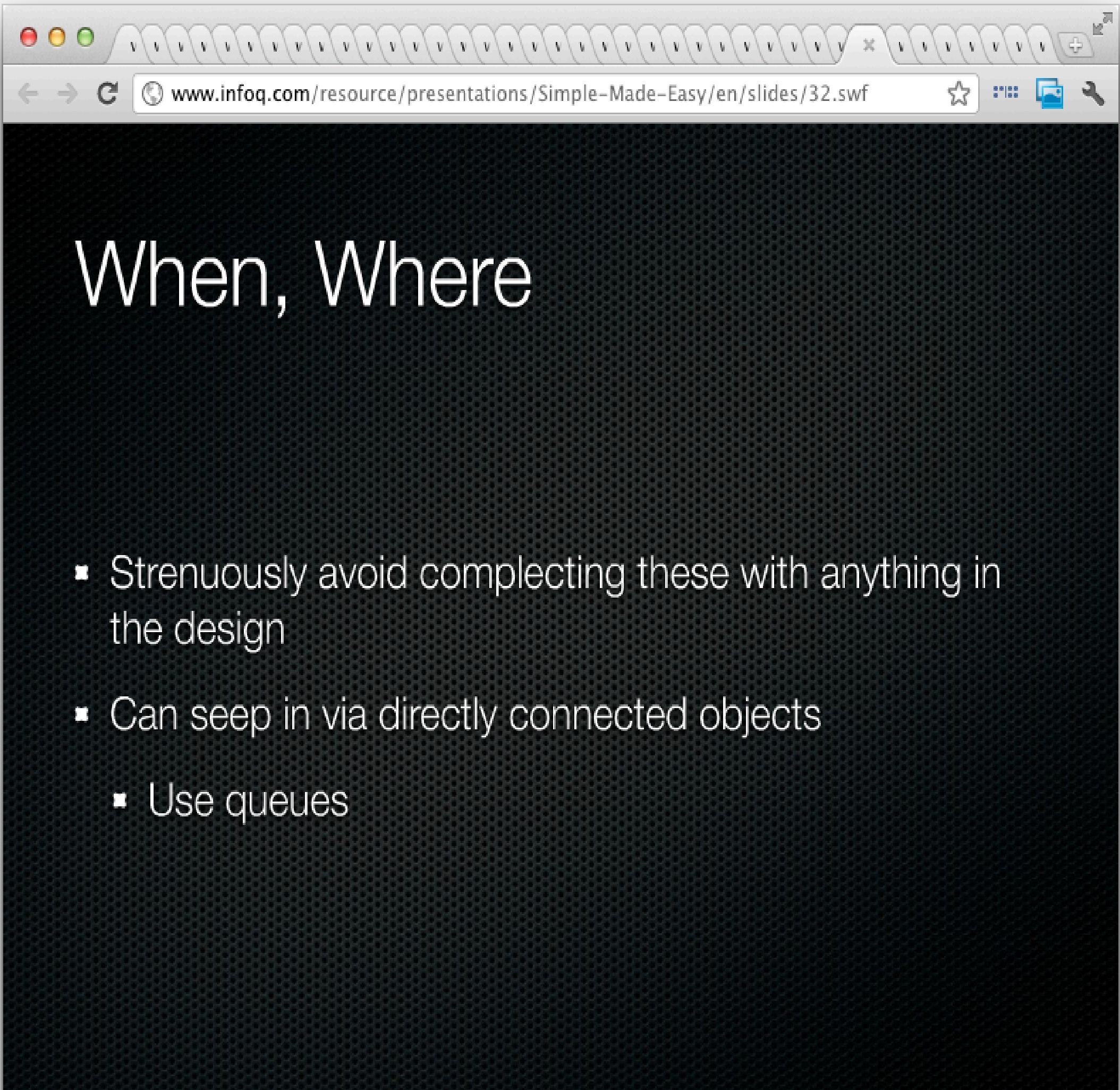
- Operations
- Form abstractions from related sets of functions
 - *Small* sets
- Represent with polymorphism constructs
- Specify inputs, outputs, semantics
- Use only values and other abstractions
- Don't complete with:
 - How

The image shows a screenshot of a web browser window. The address bar at the top displays the URL www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/30.swf. The main content area of the browser features a large, bold, white text header "Who" centered on a dark, textured background. Below the header, there is a bulleted list of guidelines for component-based design:

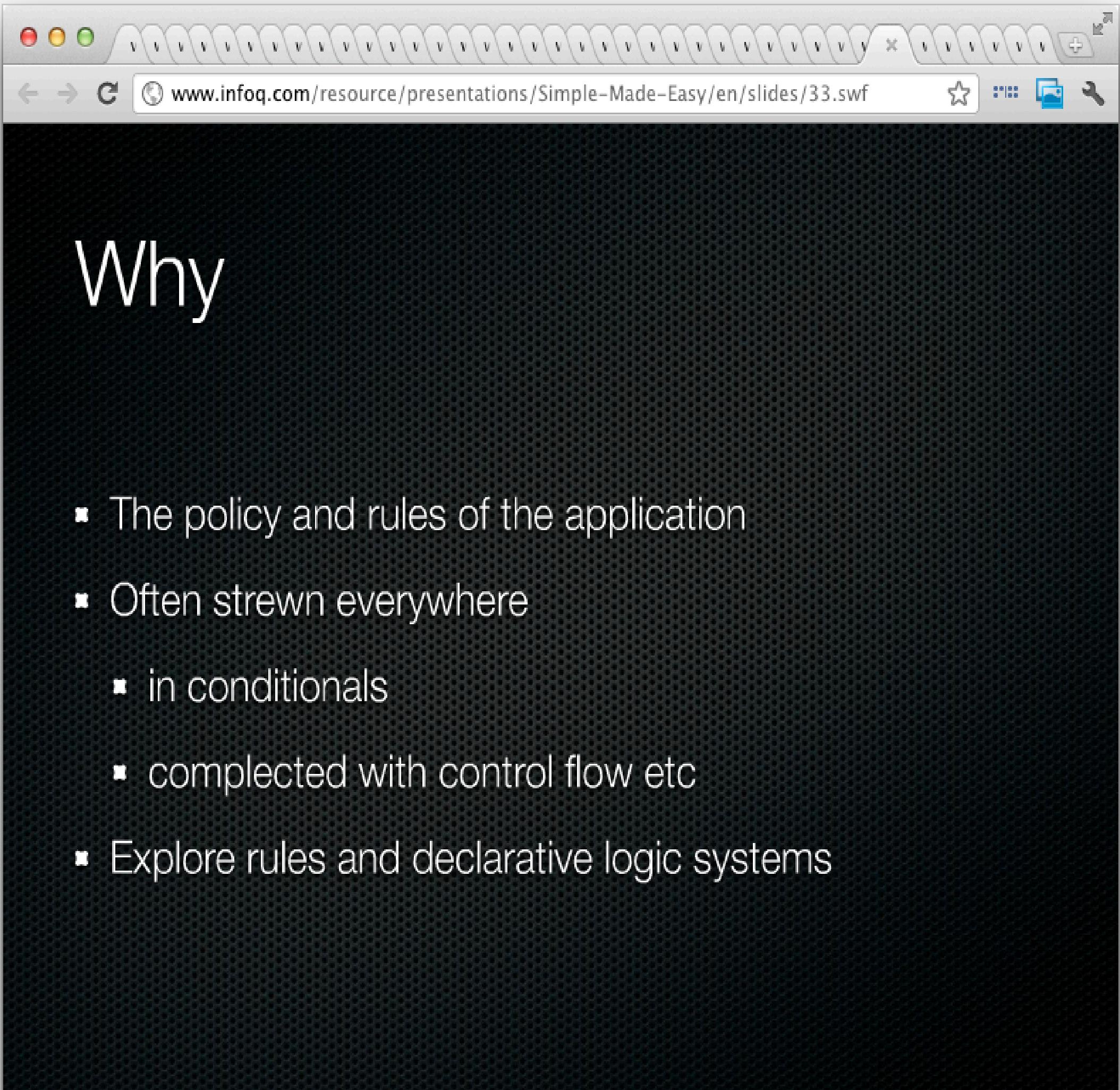
- Entities implementing abstractions
- Build from subcomponents direct-injection style
 - Pursue many subcomponents
 - e.g. policy
 - Don't complicate with:
 - component details
 - other entities



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



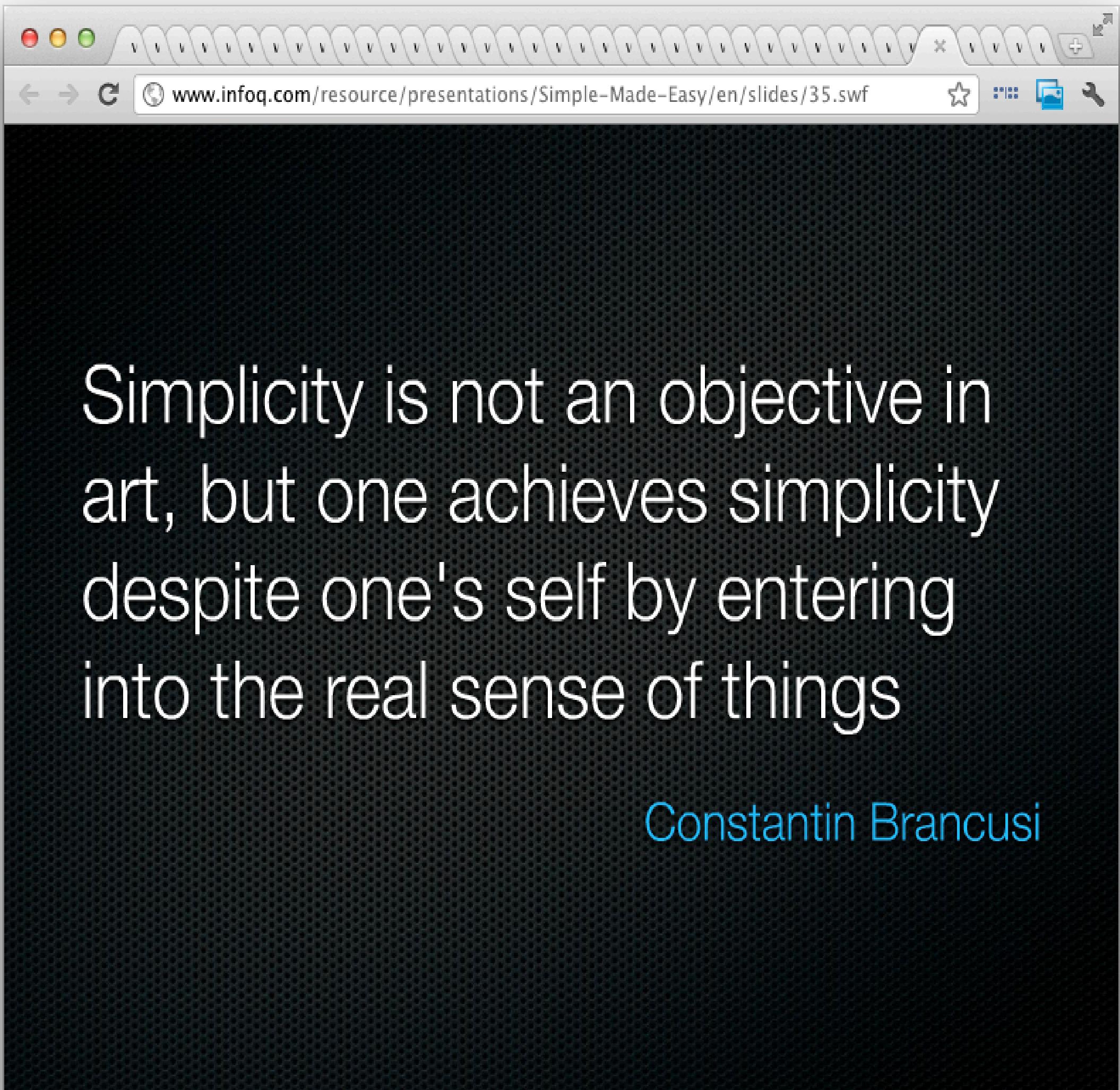
- Strenuously avoid completing these with anything in the design
- Can seep in via directly connected objects
 - Use queues



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

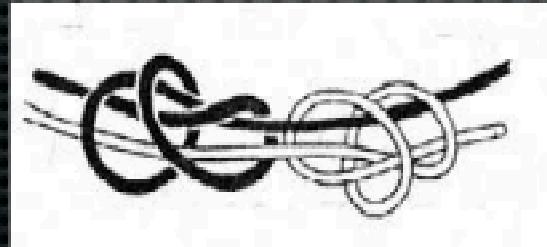
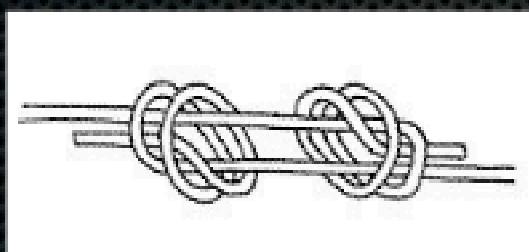
The image shows a screenshot of a web browser window. The title bar at the top displays the URL www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/34.swf. The main content area of the browser features a large, bold, white sans-serif font title "Information is Simple" centered on a dark gray textured background. Below the title, there is a bulleted list of nine items, each preceded by a small black square bullet point. The list discusses various ways to complicate or simplify information representation.

- 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



Simplifying

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



The image shows a screenshot of a web browser window. The title bar at the top displays the URL www.infoq.com/resource/presentations/Simple-Made-Easy/en/slides/37.swf. The main content area of the browser features a large, bold, white text title 'Simplicity is a Choice' centered on a dark, textured background. Below the title, there is a bulleted list of points, also displayed in white text. The browser interface includes standard window controls (minimize, maximize, close) and a toolbar with various icons.

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!

