Complexity Made Simple
Tackling Complexity in Software-Intensive Systems

Philippe Kruchten

Lockheed Martin, March 14, 2013

IEEE computer society

LOCKHEED MARTIN

Philippe Kruchten, Ph.D., P.Eng., CSDP

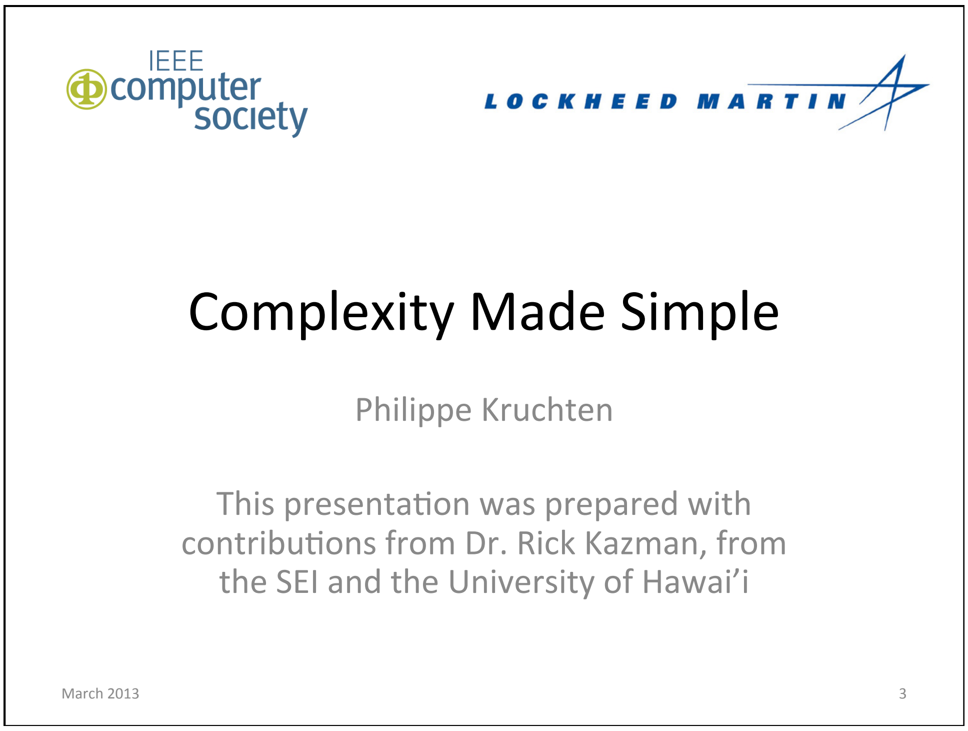
Professor of Software Engineering
NSERC Chair in Design Engineering
Department of Electrical and Computer Engineering
University of British Columbia
Vancouver, BC Canada
pbk@ece.ubc.ca

Founder and president
Kruchten Engineering Services Ltd
Vancouver, BC Canada
philippe@kruchten.com

NSERC CRSNG

Agile Vancouver IEEE 42 ifip

2

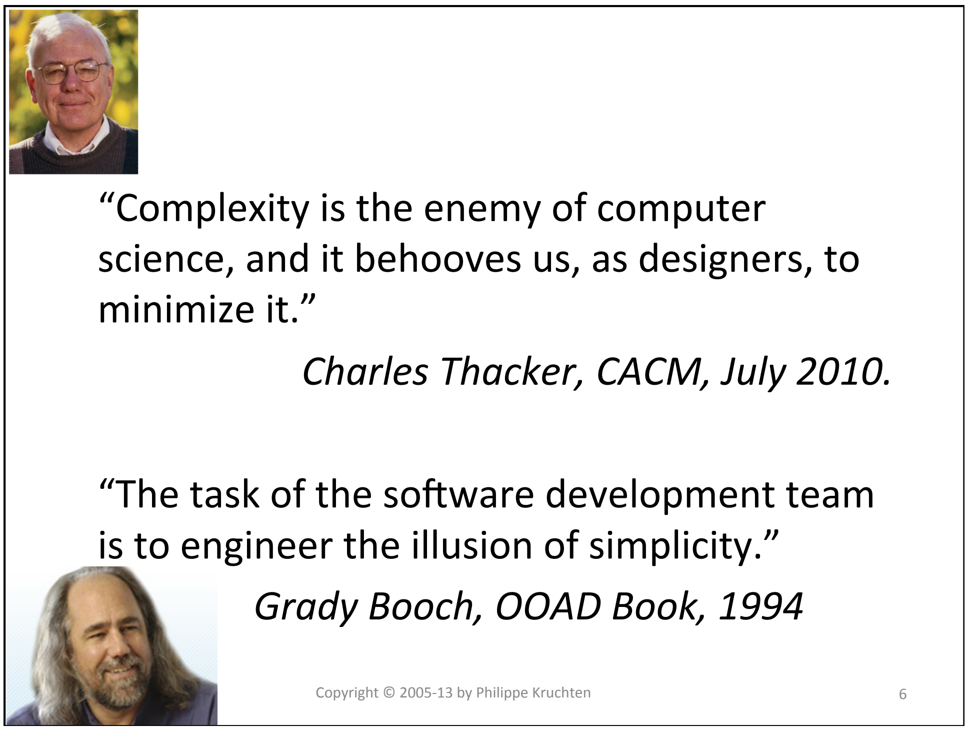


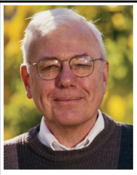
Complexity Made Simple

Philippe Kruchten

This presentation was prepared with contributions from Dr. Rick Kazman, from the SEI and the University of Hawai'i

March 2013 3





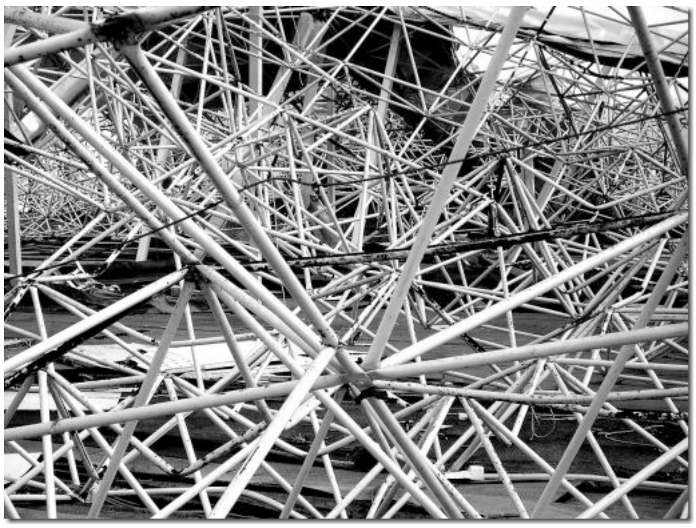
"Complexity is the enemy of computer science, and it behooves us, as designers, to minimize it."

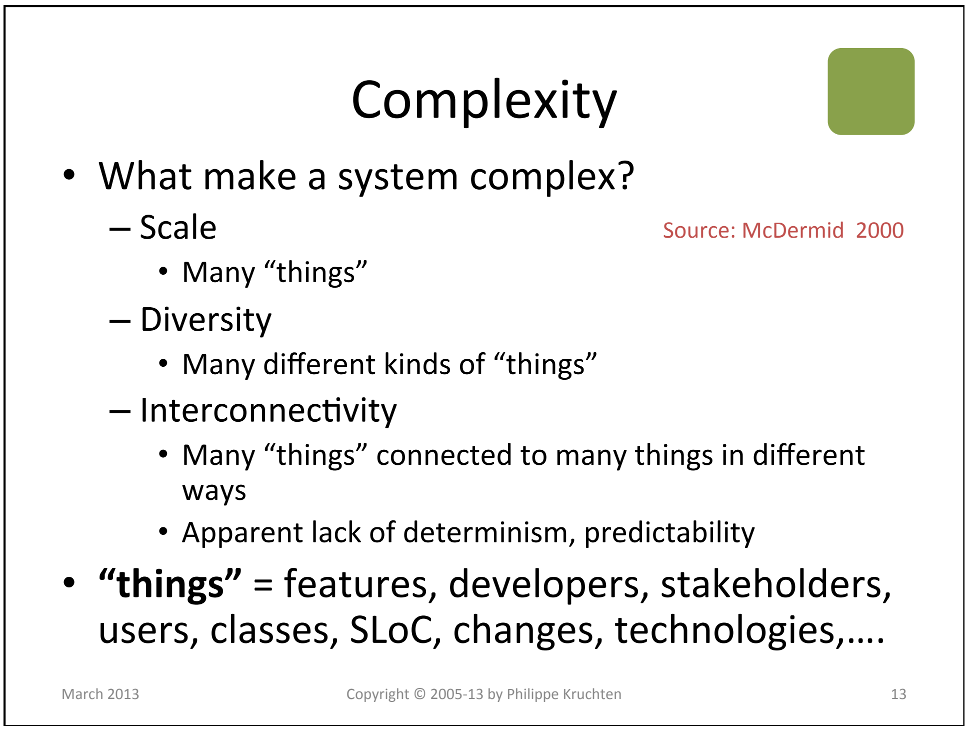
Charles Thacker, CACM, July 2010.

"The task of the software development team is to engineer the illusion of simplicity."

Grady Booch, OOAD Book, 1994

6

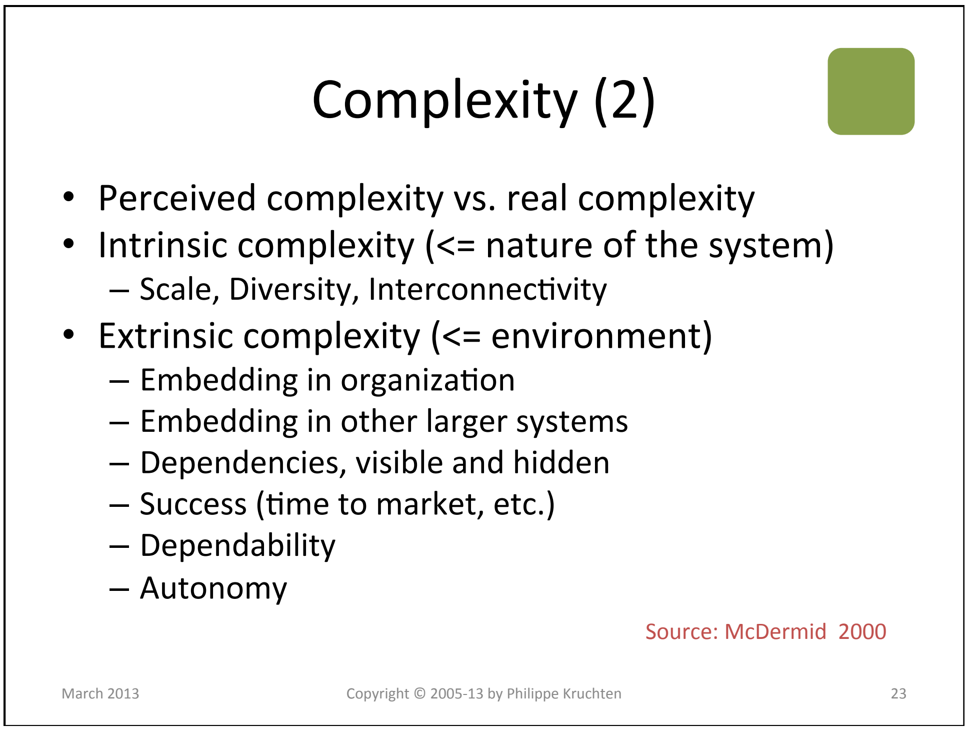




Complexity

- What make a system complex?
  - Scale
    - Many “things”
  - Diversity
    - Many different kinds of “things”
  - Interconnectivity
    - Many “things” connected to many things in different ways
    - Apparent lack of determinism, predictability
- **“things”** = features, developers, stakeholders, users, classes, SLoC, changes, technologies,….

Source: McDermid 2000

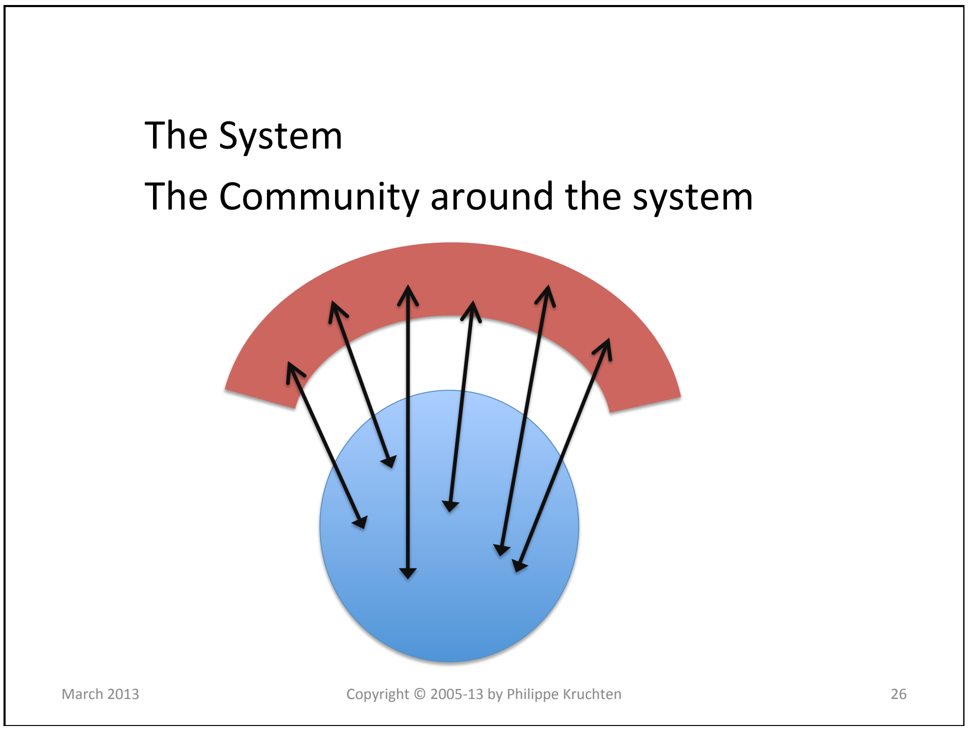


Complexity (2)

- Perceived complexity vs. real complexity
- Intrinsic complexity (<= nature of the system)
  - Scale, Diversity, Interconnectivity
- Extrinsic complexity (<= environment)
  - Embedding in organization
  - Embedding in other larger systems
  - Dependencies, visible and hidden
  - Success (time to market, etc.)
  - Dependability
  - Autonomy

Source: McDermid 2000

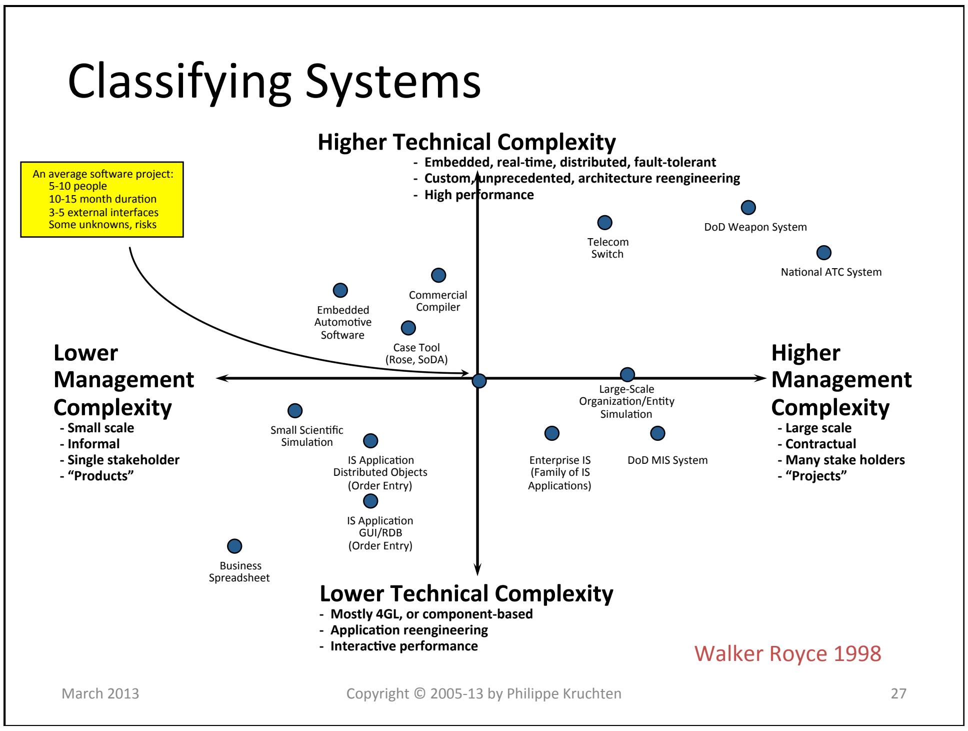
March 2013 Copyright © 2005-13 by Philippe Kruchten 23



The System

The Community around the system

March 2013 Copyright © 2005-13 by Philippe Kruchten 26

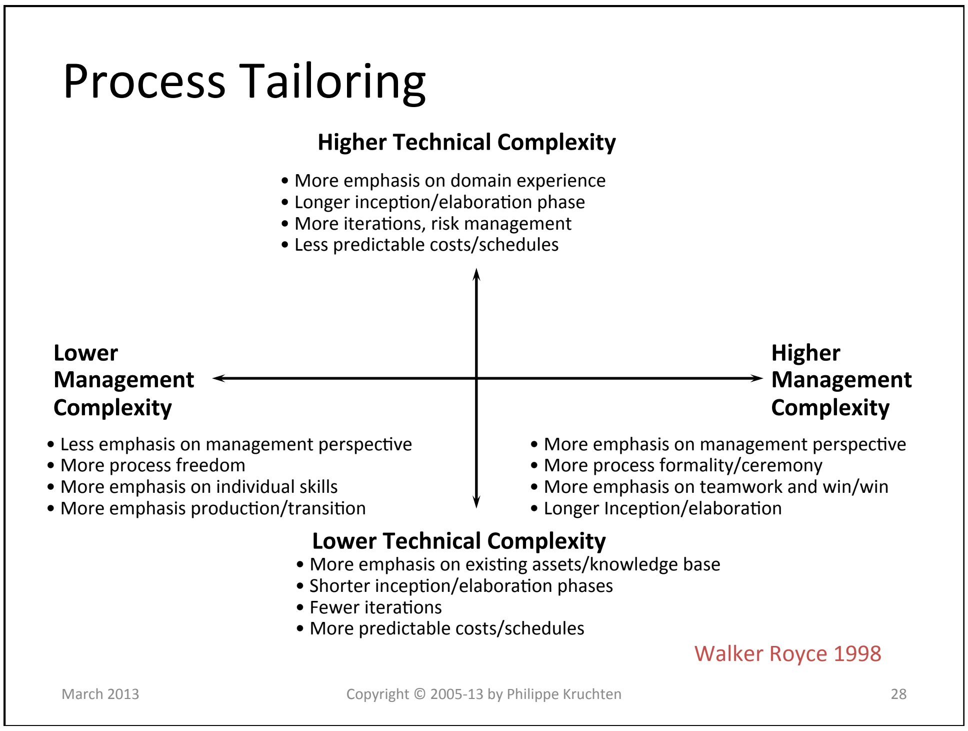


Classifying Systems

Higher Technical Complexity
- Embedded, real-time, distributed, fault-tolerant
- Custom, unprecedented, architecture reengineering
- High performance
An average software project:
5-10 people
10-15 month duration
3-5 external interfaces
Some unknowns, risks
Telecom Switch
DoD Weapon System
National ATC System
Commercial Compiler
Embedded Automotive Software
Case Tool (Rose, SoDA)
Lower Management Complexity
- Small scale
- Informal
- Single stakeholder
- "Products"
Higher Management Complexity
- Large scale
- Contractual
- Many stake holders
- "Projects"
Large-Scale Organization/Entity Simulation
Small Scientific Simulation
IS Application Distributed Objects (Order Entry)
Enterprise IS (Family of IS Applications)
DoD MIS System
IS Application GUI/RDB (Order Entry)
Business Spreadsheet
Lower Technical Complexity
- Mostly 4GL, or component-based
- Application reengineering
- Interactive performance

Walker Royce 1998

March 2013 Copyright © 2005-13 by Philippe Kruchten 27



Process Tailoring

Higher Technical Complexity

- More emphasis on domain experience
- Longer inception/elaboration phase
- More iterations, risk management
- Less predictable costs/schedules

Lower Management Complexity

- Less emphasis on management perspective
- More process freedom
- More emphasis on individual skills
- More emphasis production/transition

Higher Management Complexity

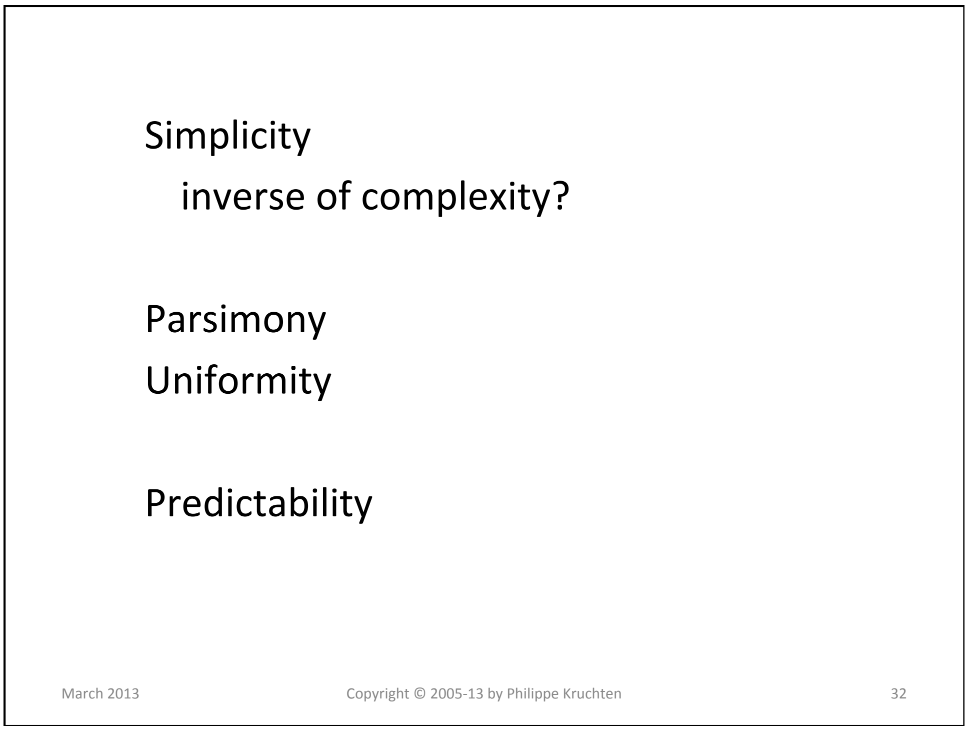
- More emphasis on management perspective
- More process formality/ceremony
- More emphasis on teamwork and win/win
- Longer Inception/elaboration

Lower Technical Complexity

- More emphasis on existing assets/knowledge base
- Shorter inception/elaboration phases
- Fewer iterations
- More predictable costs/schedules

Walker Royce 1998

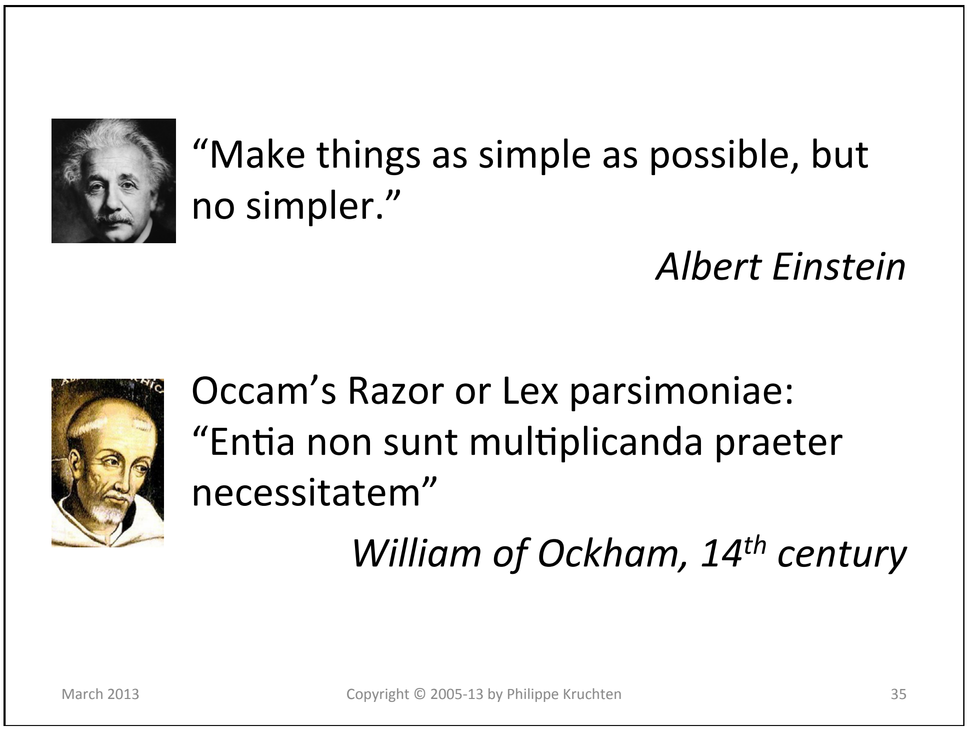
March 2013 Copyright © 2005-13 by Philippe Kruchten 28

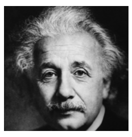


Simplicity
inverse of complexity?

Parsimony
Uniformity

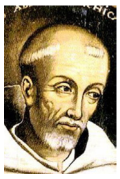
Predictability





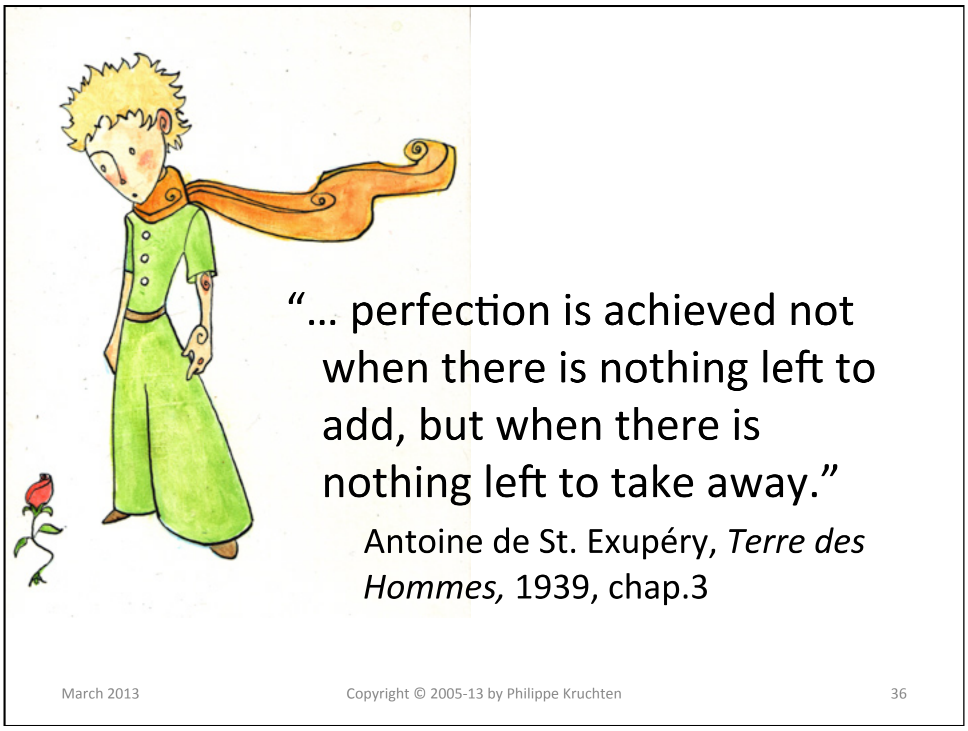
"Make things as simple as possible, but no simpler."

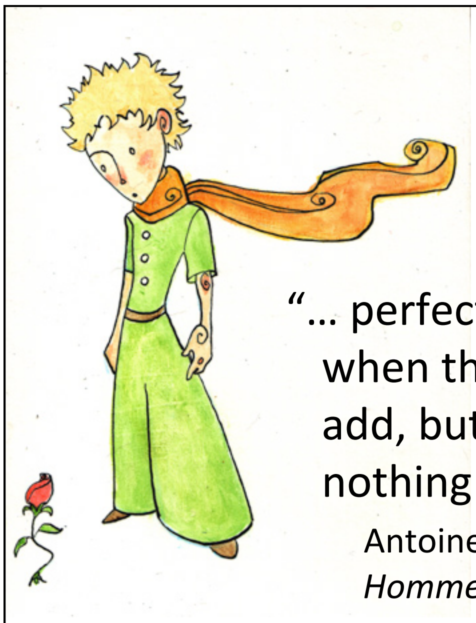
Albert Einstein



Occam's Razor or Lex parsimoniae:
"Entia non sunt multiplicanda praeter necessitatem"

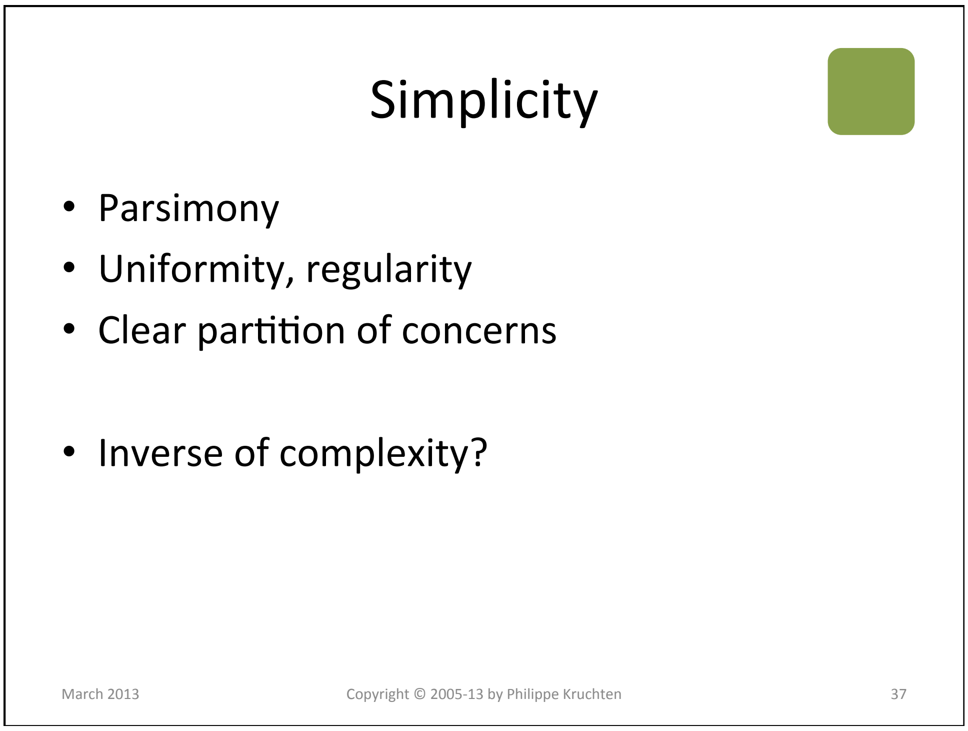
William of Ockham, 14th century





“… perfection is achieved not when there is nothing left to add, but when there is nothing left to take away.”

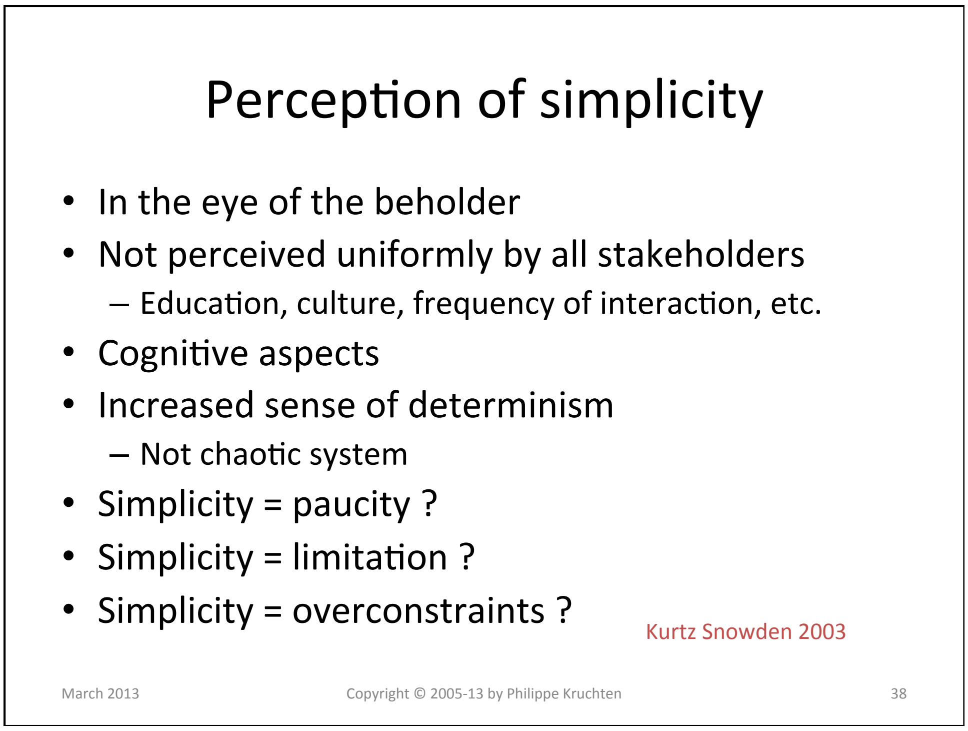
Antoine de St. Exupéry, *Terre des Hommes,* 1939, chap.3



Simplicity

- Parsimony
- Uniformity, regularity
- Clear partition of concerns

- Inverse of complexity?

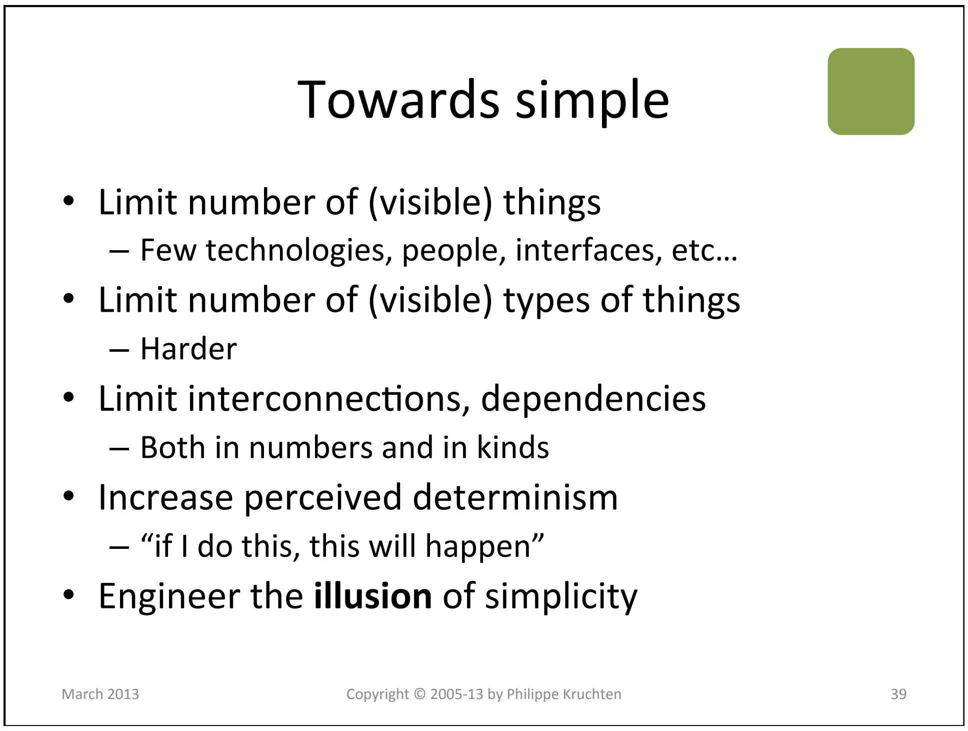


Perception of simplicity

- In the eye of the beholder
- Not perceived uniformly by all stakeholders
  - Education, culture, frequency of interaction, etc.
- Cognitive aspects
- Increased sense of determinism
  - Not chaotic system
- Simplicity = paucity ?
- Simplicity = limitation ?
- Simplicity = overconstraints ?

Kurtz Snowden 2003

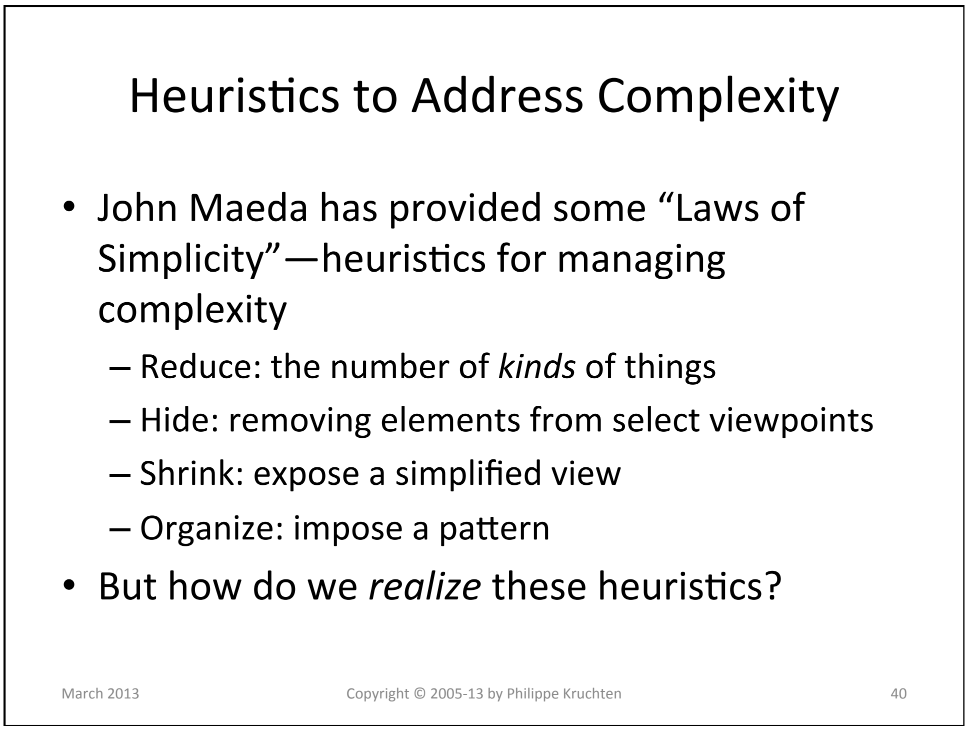
March 2013 Copyright © 2005-13 by Philippe Kruchten 38



Towards simple

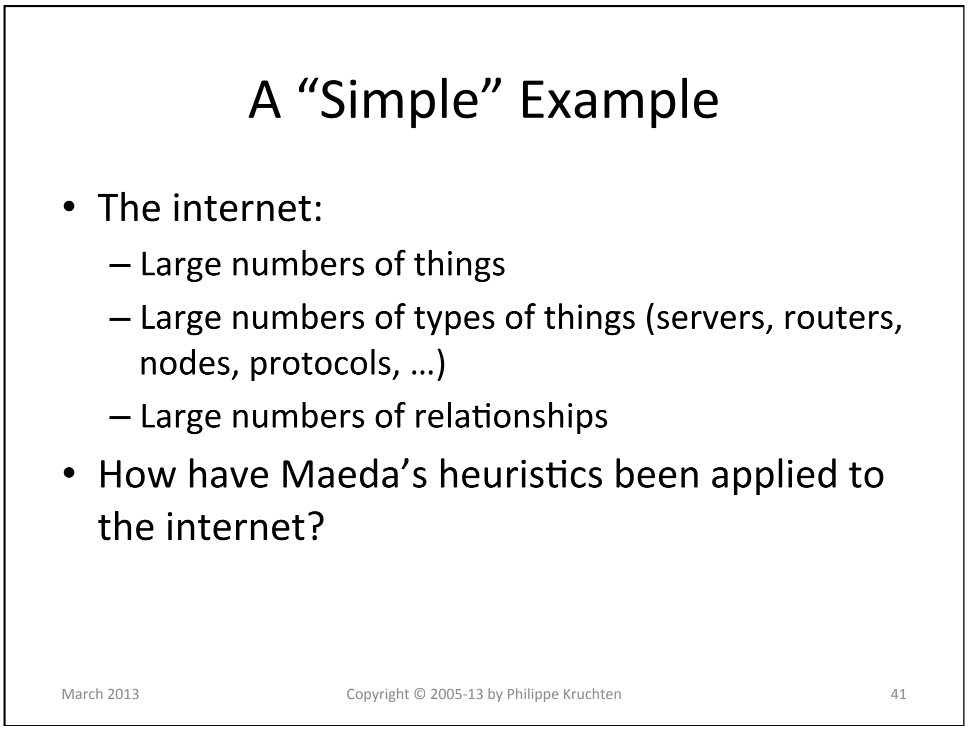
- Limit number of (visible) things
  - Few technologies, people, interfaces, etc…
- Limit number of (visible) types of things
  - Harder
- Limit interconnections, dependencies
  - Both in numbers and in kinds
- Increase perceived determinism
  - “if I do this, this will happen”
- Engineer the **illusion** of simplicity

March 2013 Copyright © 2005-13 by Philippe Kruchten 39



Heuristics to Address Complexity

- John Maeda has provided some “Laws of Simplicity”—heuristics for managing complexity
  - Reduce: the number of *kinds* of things
  - Hide: removing elements from select viewpoints
  - Shrink: expose a simplified view
  - Organize: impose a pattern
- But how do we *realize* these heuristics?



A “Simple” Example

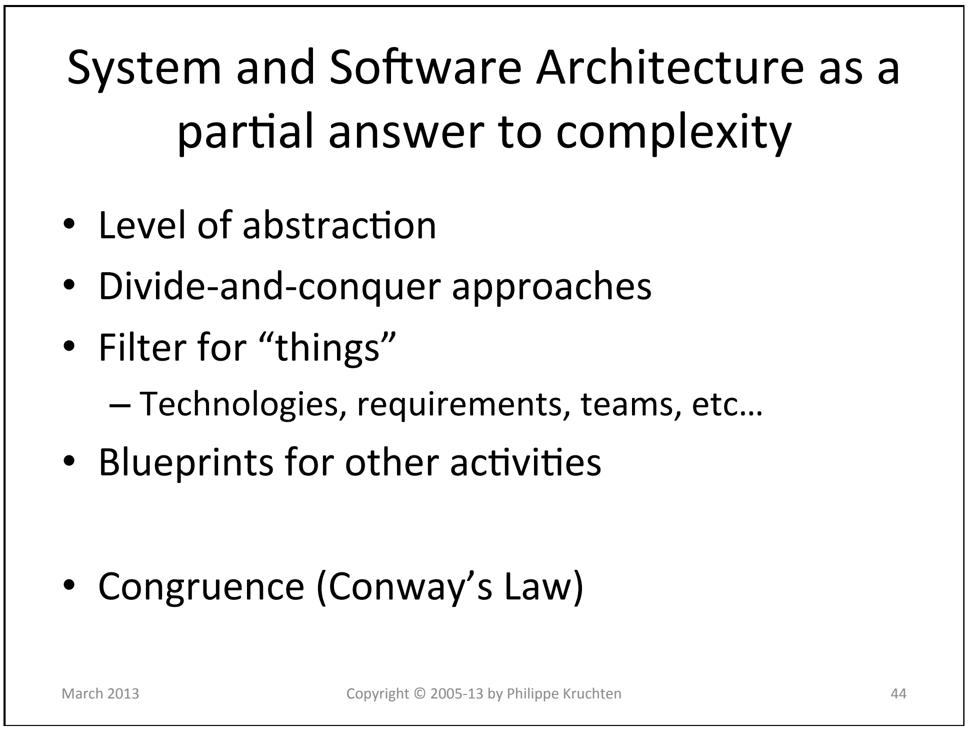
- The internet:
  - Large numbers of things
  - Large numbers of types of things (servers, routers, nodes, protocols, …)
  - Large numbers of relationships
- How have Maeda’s heuristics been applied to the internet?

A “Simple” Example - 2

- *Reduce*: strict control on the number of types of things (managed by the W3C)
- *Hide*: lower level protocols and physical infrastructure are all hidden
- *Shrink*: Google.com is a shrunk representation of millions of nodes
- *Organize*: many patterns—e.g. P2P, client-server, SOA, broker—are used to structure the internet

How do we *design* for complexity?

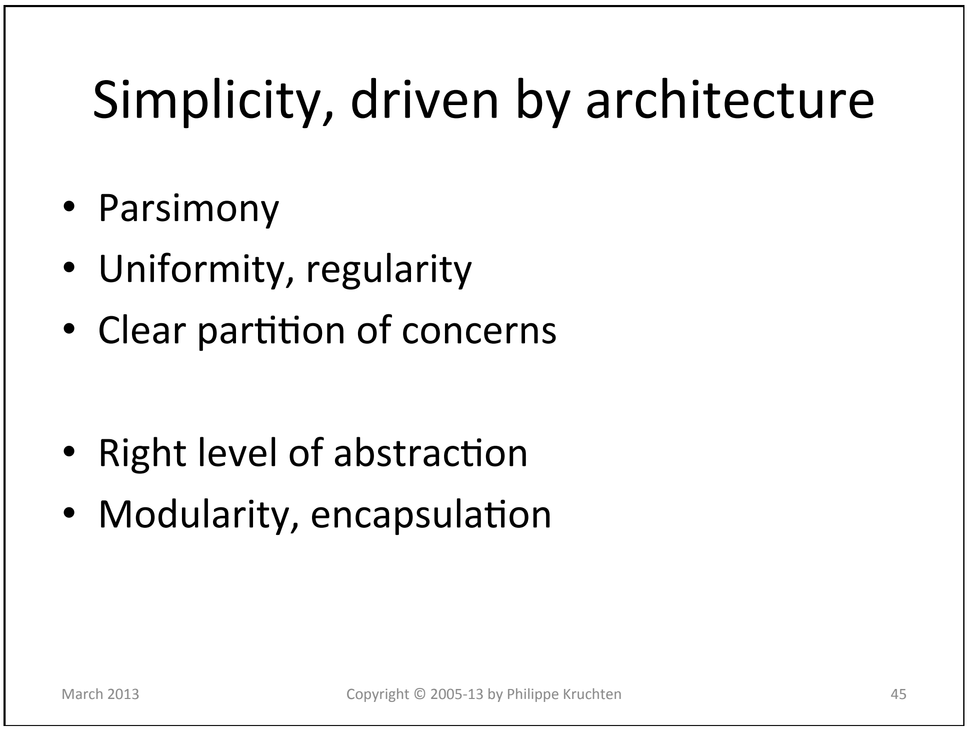
Architecture to the Rescue!



System and Software Architecture as a partial answer to complexity

- Level of abstraction
- Divide-and-conquer approaches
- Filter for "things"
  - Technologies, requirements, teams, etc…
- Blueprints for other activities

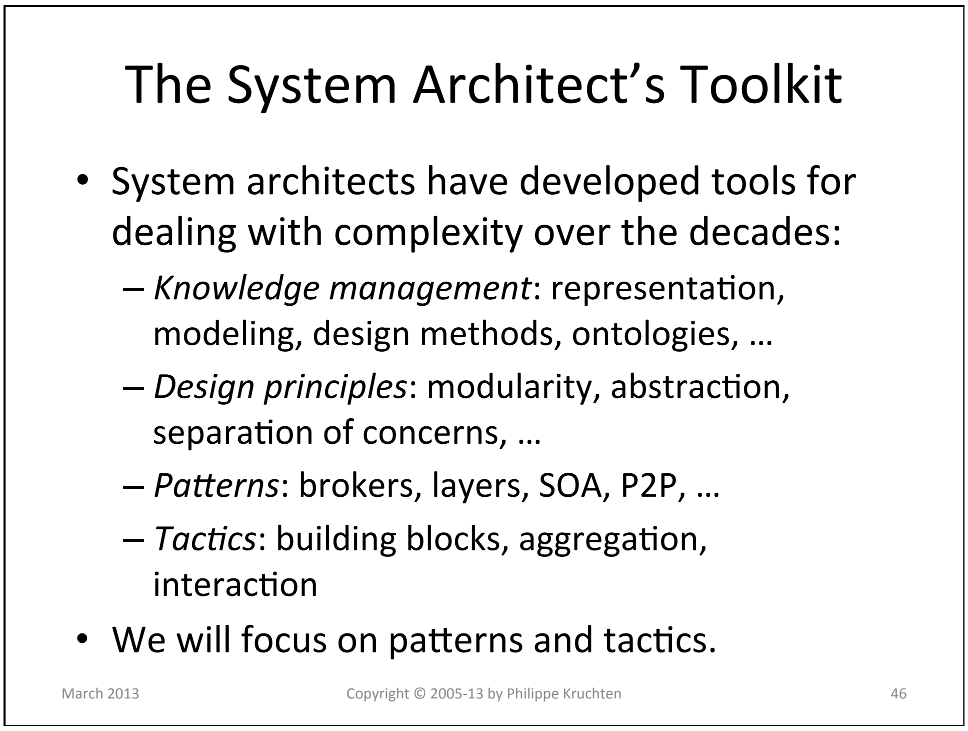
- Congruence (Conway's Law)



Simplicity, driven by architecture

- Parsimony
- Uniformity, regularity
- Clear partition of concerns

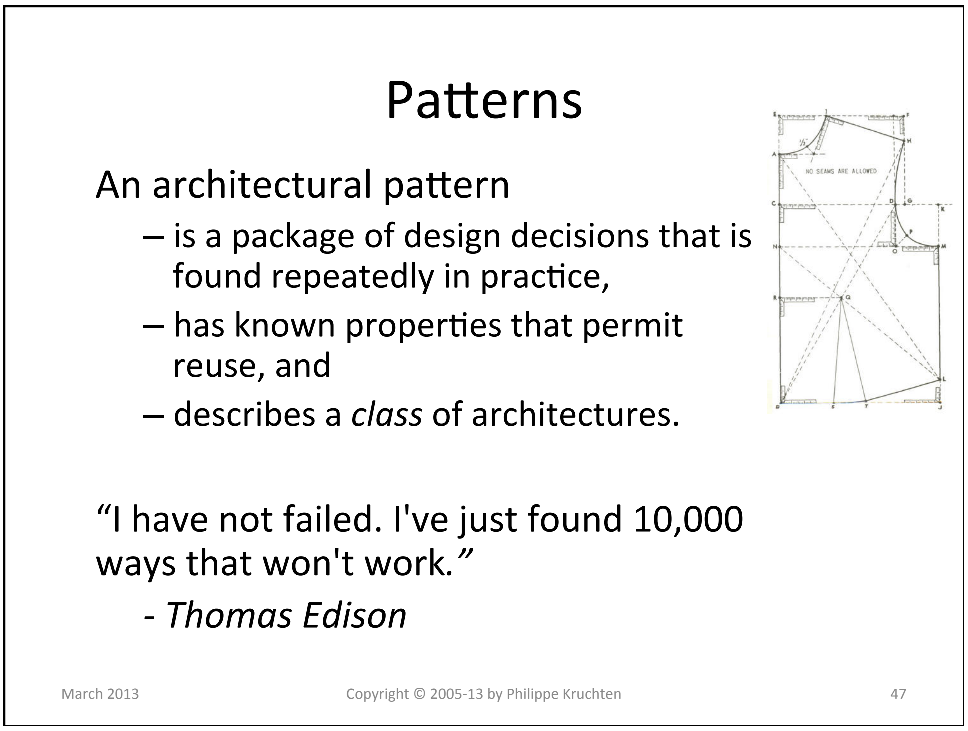
- Right level of abstraction
- Modularity, encapsulation



The System Architect's Toolkit

- System architects have developed tools for dealing with complexity over the decades:
  - *Knowledge management*: representation, modeling, design methods, ontologies, …
  - *Design principles*: modularity, abstraction, separation of concerns, …
  - *Patterns*: brokers, layers, SOA, P2P, …
  - *Tactics*: building blocks, aggregation, interaction
- We will focus on patterns and tactics.

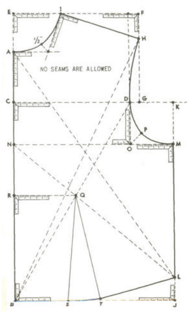
March 2013 Copyright © 2005-13 by Philippe Kruchten 46



Patterns

An architectural pattern

- is a package of design decisions that is found repeatedly in practice,
- has known properties that permit reuse, and
- describes a *class* of architectures.

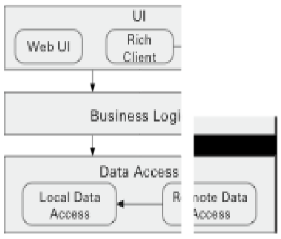


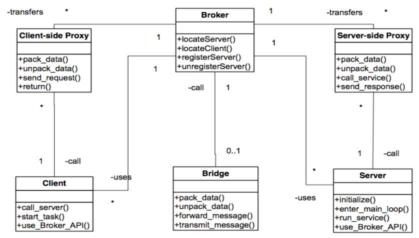
"I have not failed. I've just found 10,000 ways that won't work."

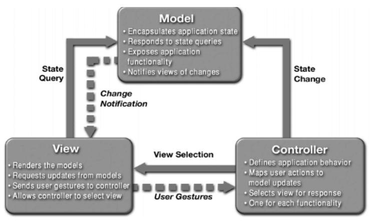
- Thomas Edison

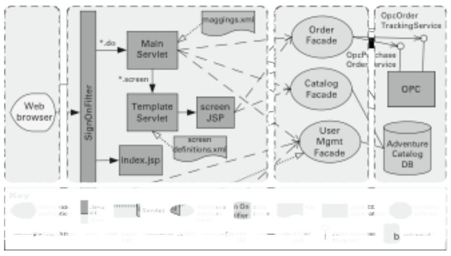
March 2013 Copyright © 2005-13 by Philippe Kruchten 47

Patterns





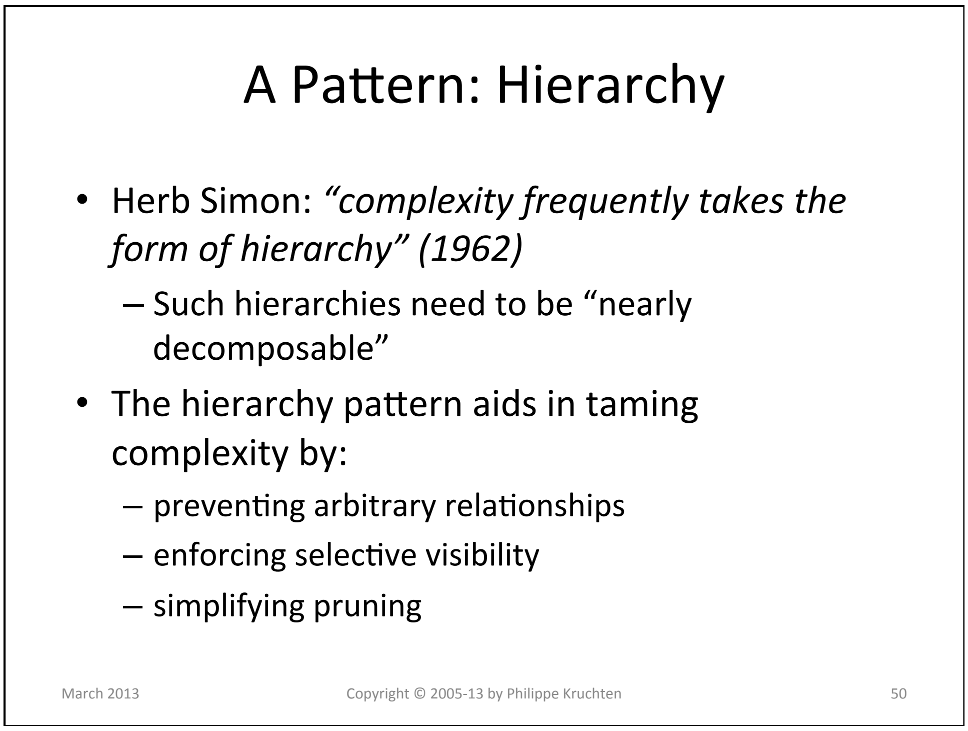

Model
• Encapsulates application state
• Responds to state queries
• Exposes application functionality
• Notifies views of changes
State Query
Change Notification
State Change
View Selection
View
• Renders the models
• Requests updates from models
• Sends user gestures to controller
• Allows controller to select view
User Gestures
Controller
• Defines application behavior
• Maps user actions to model updates
• Selects view for response
• One for each functionality



Tactics

An architectural tactic is a design decision that affects a quality attribute response

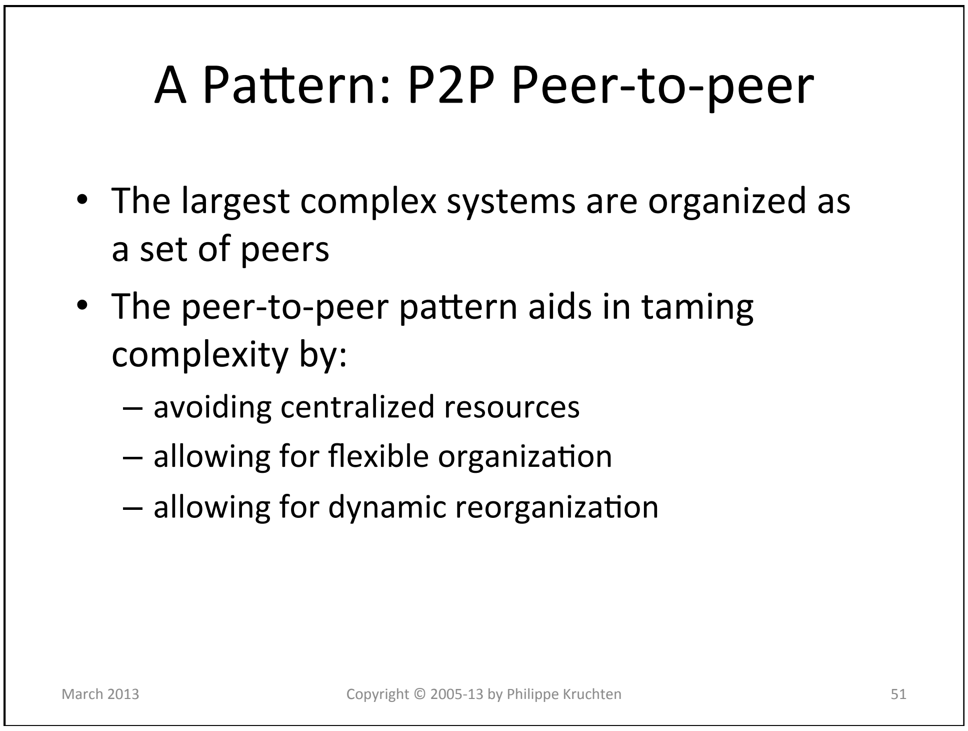
Patterns describe holistic solutions; tactics describe “atomic” architectural strategies.



A Pattern: Hierarchy

- Herb Simon: *"complexity frequently takes the form of hierarchy" (1962)*
  - Such hierarchies need to be "nearly decomposable"
- The hierarchy pattern aids in taming complexity by:
  - preventing arbitrary relationships
  - enforcing selective visibility
  - simplifying pruning

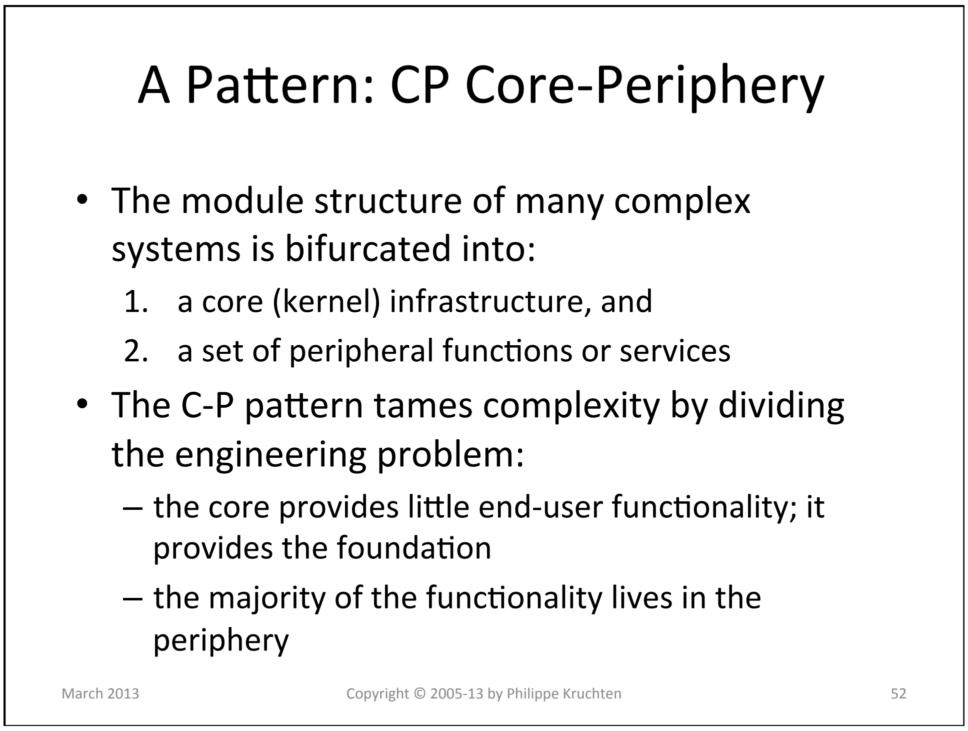
March 2013 Copyright © 2005-13 by Philippe Kruchten 50



A Pattern: P2P Peer-to-peer

- The largest complex systems are organized as a set of peers
- The peer-to-peer pattern aids in taming complexity by:
  - avoiding centralized resources
  - allowing for flexible organization
  - allowing for dynamic reorganization

March 2013 Copyright © 2005-13 by Philippe Kruchten 51

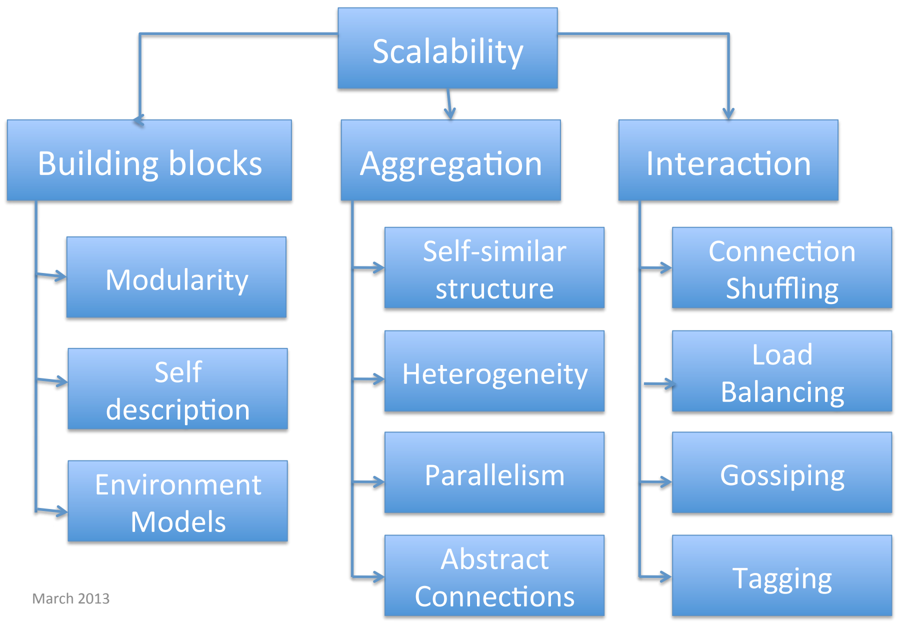


A Pattern: CP Core-Periphery

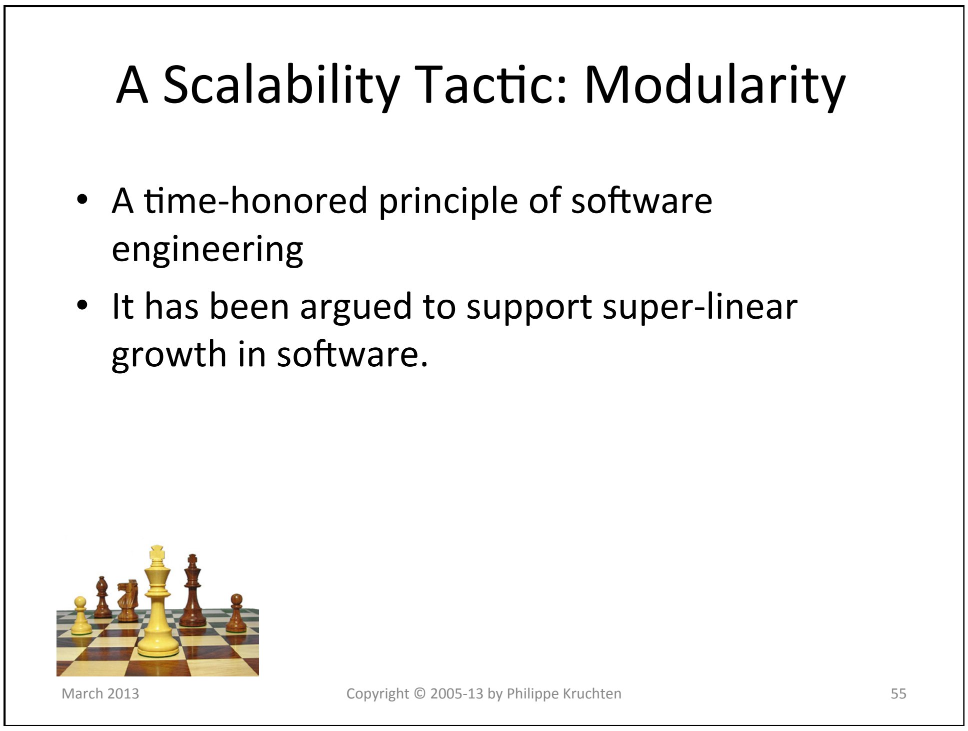
- The module structure of many complex systems is bifurcated into:
  1. a core (kernel) infrastructure, and
  2. a set of peripheral functions or services
- The C-P pattern tames complexity by dividing the engineering problem:
  – the core provides little end-user functionality; it provides the foundation
  – the majority of the functionality lives in the periphery

March 2013 Copyright © 2005-13 by Philippe Kruchten 52

Tactics for scalability

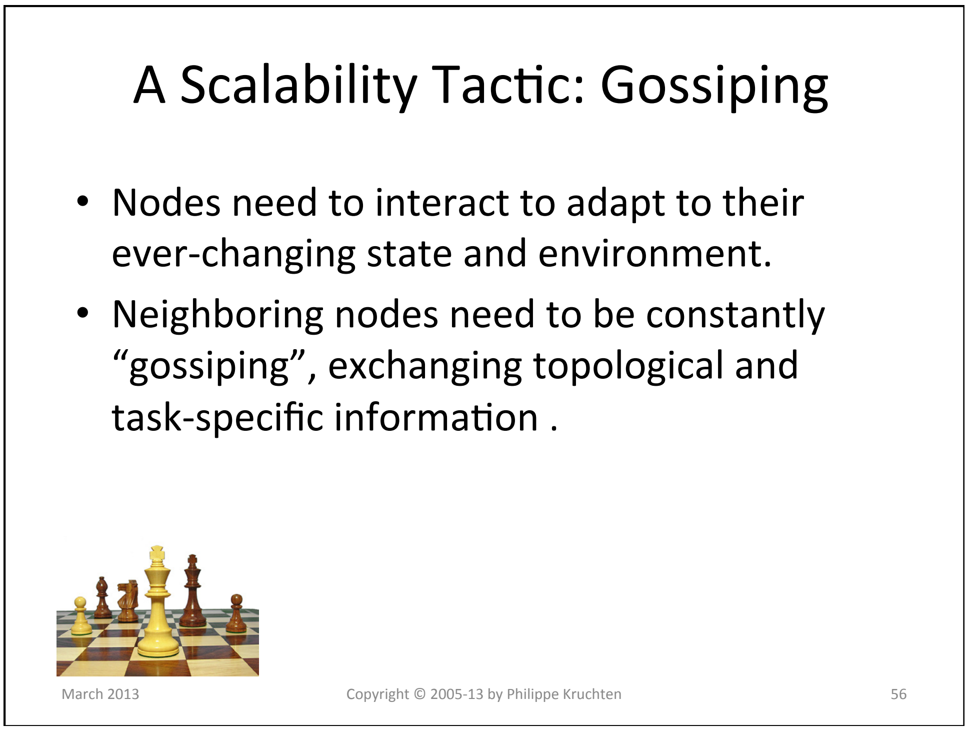


March 2013



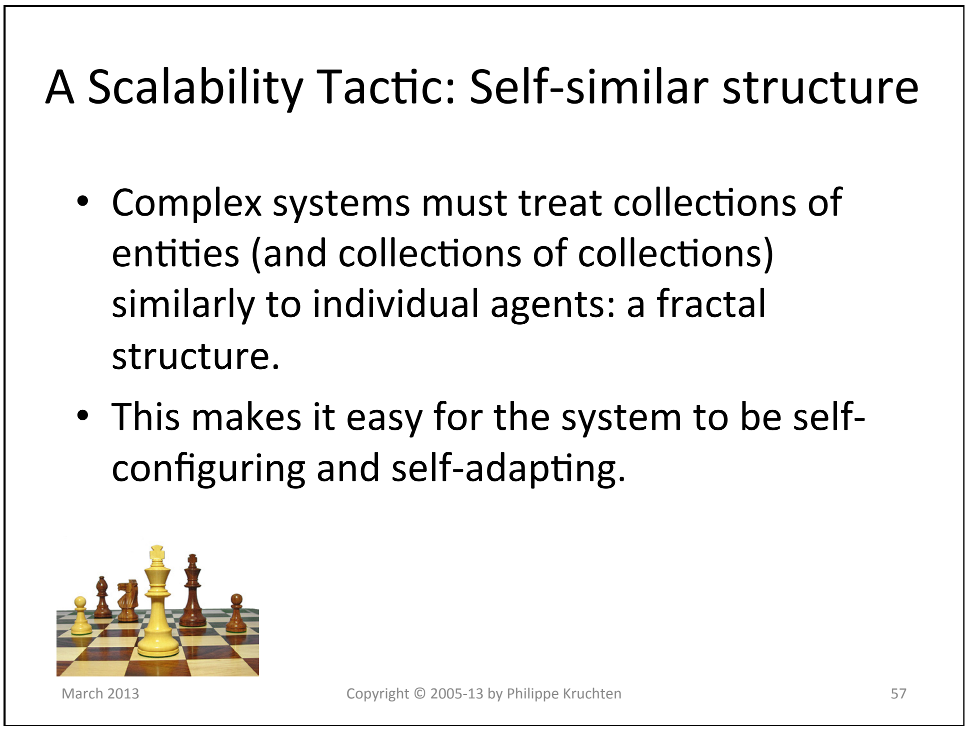
A Scalability Tactic: Modularity

- A time-honored principle of software engineering
- It has been argued to support super-linear growth in software.



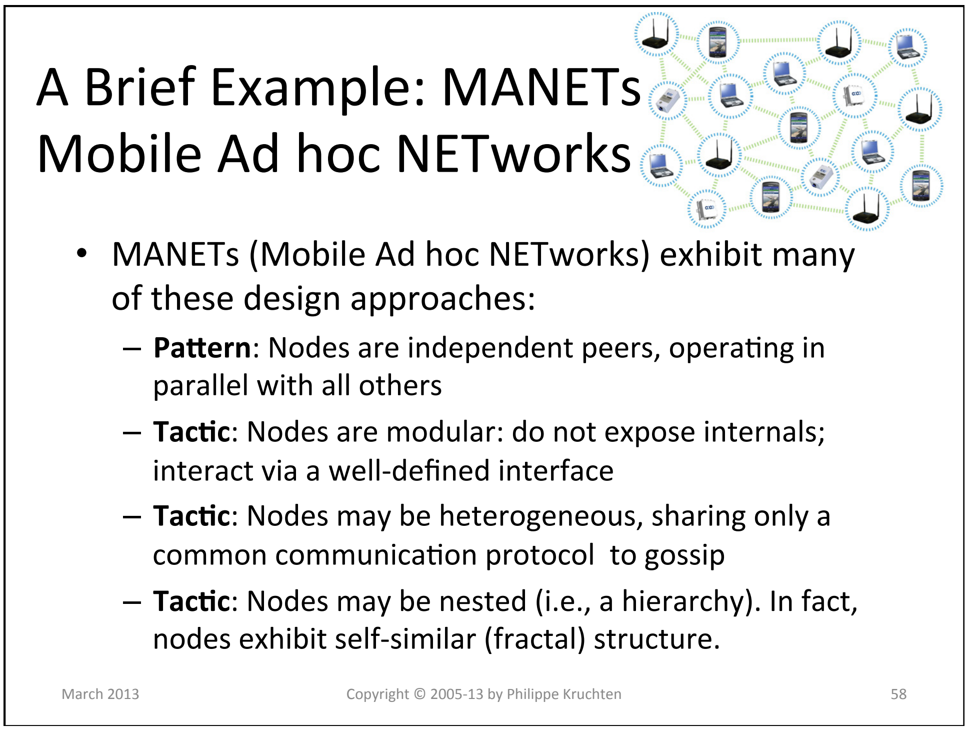
A Scalability Tactic: Gossiping

- Nodes need to interact to adapt to their ever-changing state and environment.
- Neighboring nodes need to be constantly “gossiping”, exchanging topological and task-specific information .

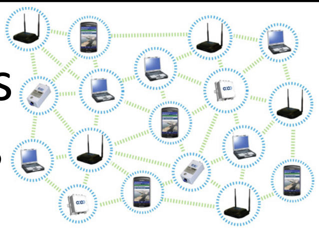


A Scalability Tactic: Self-similar structure

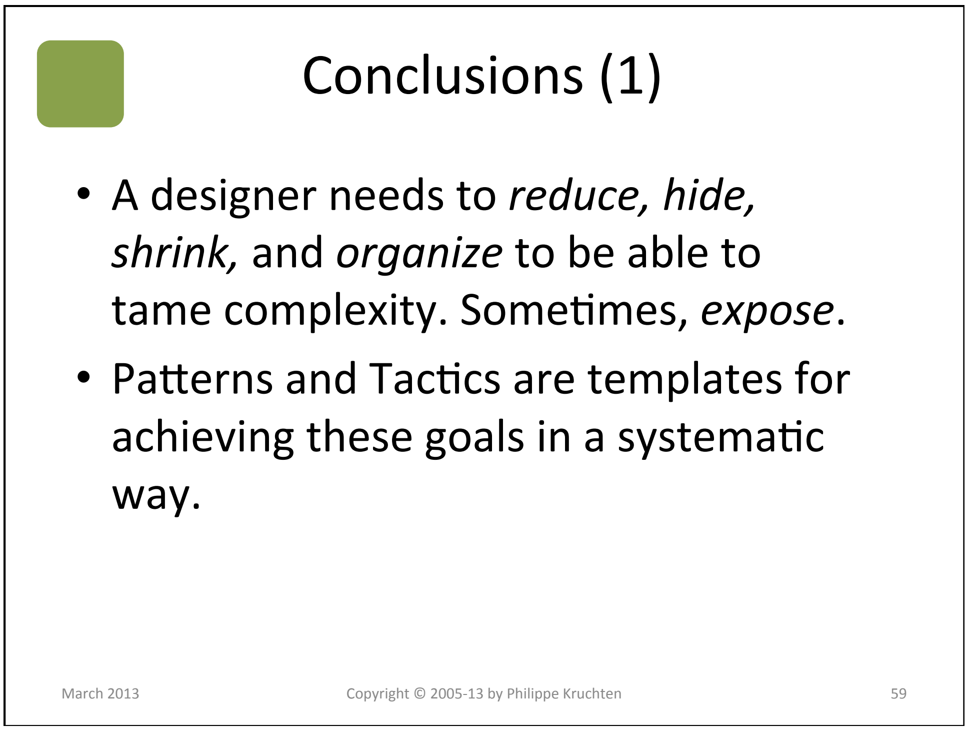
- Complex systems must treat collections of entities (and collections of collections) similarly to individual agents: a fractal structure.
- This makes it easy for the system to be self-configuring and self-adapting.



A Brief Example: MANETs Mobile Ad hoc NETworks

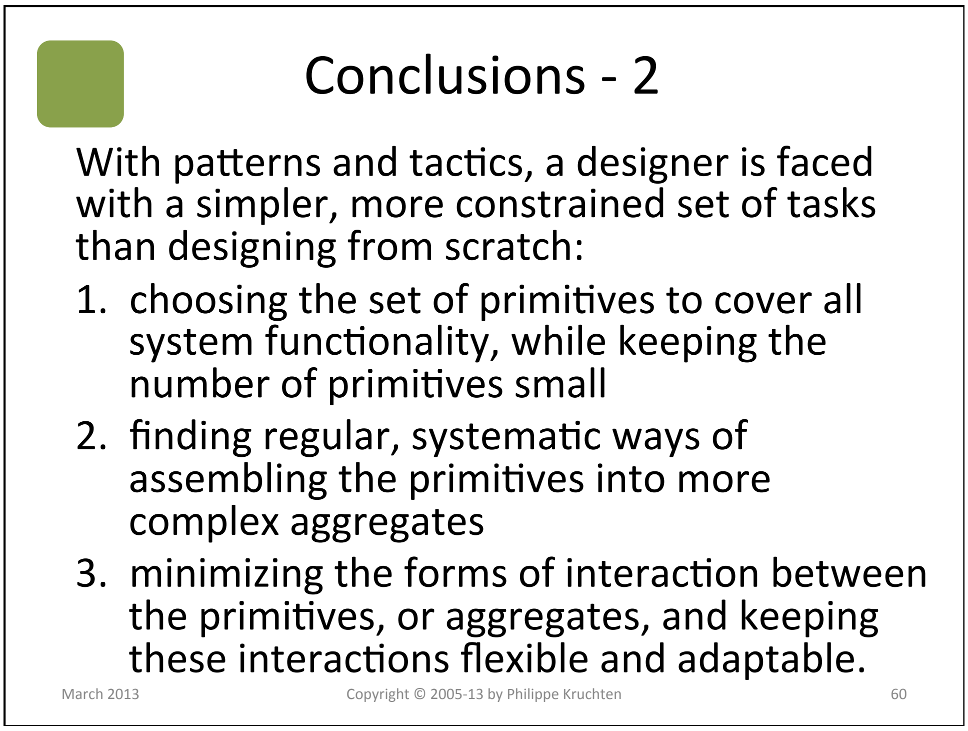


- MANETs (Mobile Ad hoc NETworks) exhibit many of these design approaches:
  - **Pattern**: Nodes are independent peers, operating in parallel with all others
  - **Tactic**: Nodes are modular: do not expose internals; interact via a well-defined interface
  - **Tactic**: Nodes may be heterogeneous, sharing only a common communication protocol to gossip
  - **Tactic**: Nodes may be nested (i.e., a hierarchy). In fact, nodes exhibit self-similar (fractal) structure.



Conclusions (1)

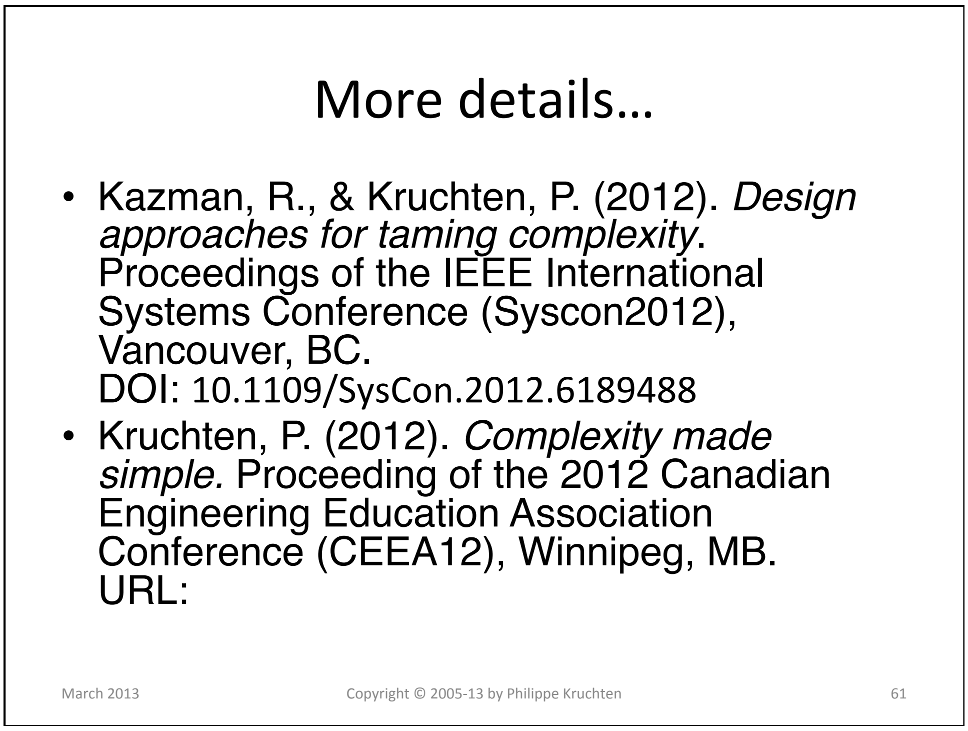
- A designer needs to *reduce, hide, shrink,* and *organize* to be able to tame complexity. Sometimes, *expose*.
- Patterns and Tactics are templates for achieving these goals in a systematic way.



Conclusions - 2

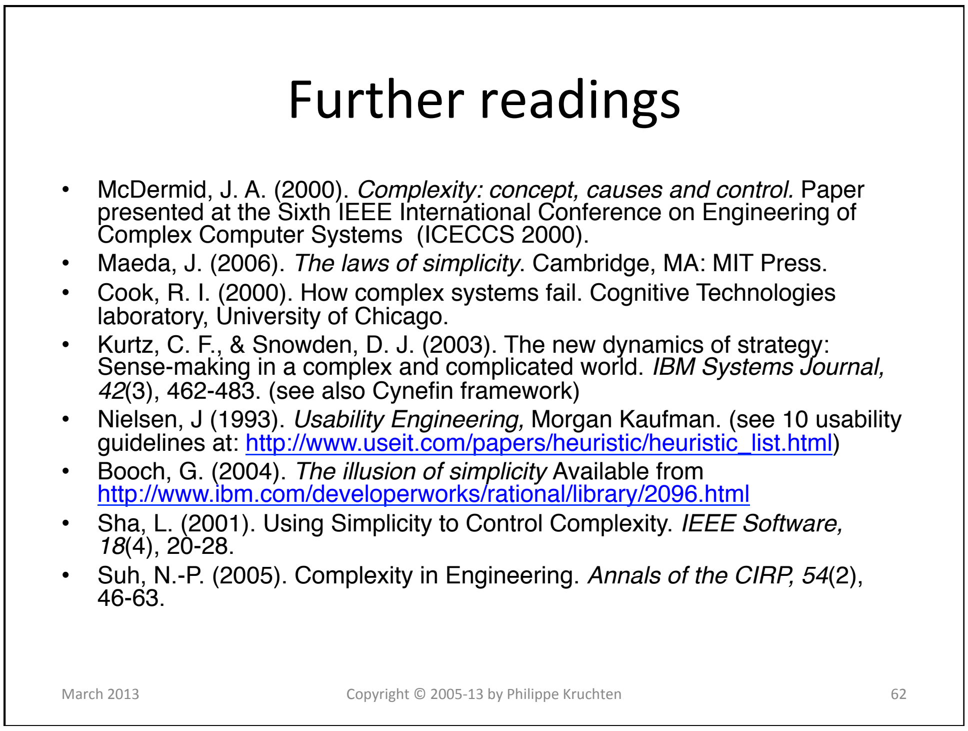
With patterns and tactics, a designer is faced with a simpler, more constrained set of tasks than designing from scratch:

1. choosing the set of primitives to cover all system functionality, while keeping the number of primitives small
2. finding regular, systematic ways of assembling the primitives into more complex aggregates
3. minimizing the forms of interaction between the primitives, or aggregates, and keeping these interactions flexible and adaptable.



More details…

- Kazman, R., & Kruchten, P. (2012). *Design approaches for taming complexity*. Proceedings of the IEEE International Systems Conference (Syscon2012), Vancouver, BC.
DOI: 10.1109/SysCon.2012.6189488
- Kruchten, P. (2012). *Complexity made simple.* Proceeding of the 2012 Canadian Engineering Education Association Conference (CEEA12), Winnipeg, MB.
URL:



Further readings

- McDermid, J. A. (2000). *Complexity: concept, causes and control.* Paper presented at the Sixth IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2000).
- Maeda, J. (2006). *The laws of simplicity*. Cambridge, MA: MIT Press.
- Cook, R. I. (2000). How complex systems fail. Cognitive Technologies laboratory, University of Chicago.
- Kurtz, C. F., & Snowden, D. J. (2003). The new dynamics of strategy: Sense-making in a complex and complicated world. *IBM Systems Journal, 42*(3), 462-483. (see also Cynefin framework)
- Nielsen, J (1993). *Usability Engineering,* Morgan Kaufman. (see 10 usability guidelines at: http://www.useit.com/papers/heuristic/heuristic_list.html)
- Booch, G. (2004). *The illusion of simplicity* Available from http://www.ibm.com/developerworks/rational/library/2096.html
- Sha, L. (2001). Using Simplicity to Control Complexity. *IEEE Software, 18*(4), 20-28.
- Suh, N.-P. (2005). Complexity in Engineering. *Annals of the CIRP, 54*(2), 46-63.