Complexity Made Simple

Philippe Kruchten
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
Philippe Kruchten
Complex looks simple...
Complex looks simple...
... but simple is complex
“Complexity is the enemy of computer science, and it behooves us, as designers, to minimize it.”

*Charles Thacker, CACM, July 2010.*
“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
Scale

Many *things*
Scale

Many *things*

Diversity

many different kinds of *things*
Scale

Many *things*

Diversity

Many different kinds of *things*

Interdependencies

between all these many *things*
Scale

Many things

Diversity

many different kinds of things

Interdependencies

between all these many things

Kinds of Things = features, services, LoC, classes, stakeholders, developers, sites, technologies, managers, ....
Complexity

• What make Systems 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,....
Intrinsic complexity
Intrinsic complexity

essential complexity
Intrinsic complexity
essential complexity

Extrinsic complexity
Intrinsic complexity
  essential complexity

Extrinsic complexity
  accidental complexity
Perceived complexity
Perceived complexity

culture, familiarity, education
Perceived complexity

culture, familiarity, education

Determinism
Perceived complexity
culture, familiarity, education

Determinism

Visibility
Complex /= confusing

Complicated
Complex
Chaotic
Complexity

• 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
Simplicity
Simplicity

inverse of complexity?
Simplicity
inverse of complexity?
Parsimony
Uniformity
Simplicity
inverse of complexity?

Parsimony
Uniformity

Predictability
Simple /= simplistic
“Make things as simple as possible, but no simpler.”

Albert Einstein
“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

See: Kurtz Snowden 2003
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
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
• Expose
The System
The System

The Community around the system
The System

The Community around the system
Classifying Systems

Higher Technical Complexity
- Embedded, real-time, distributed, fault-tolerant
- Custom, unprecedented, architecture reengineering
- High performance

Lower Technical Complexity
- Mostly 4GL, or component-based
- Application reengineering
- Interactive performance

Higher Management Complexity
- Large scale
- Contractual
- Many stakeholders
- “Projects”

Lower Management Complexity
- Small scale
- Informal
- Single stakeholder
- “Products”

An average software project:
5-10 people
10-15 month duration
3-5 external interfaces
Some unknowns, risks

DoD MIS System
DoD Weapon System
National ATC System
Telecom Switch
Large-Scale Organization/Entity Simulation
Enterprise IS (Family of IS Applications)
IS Application GUI/RDB (Order Entry)
IS Application Distributed Objects (Order Entry)
Small Scientific Simulation
Embedded Automotive Software
Commercial Compiler
Case Tool (Rose, SoDA)
Business Spreadsheet
GUISystem

Version 2.3 43
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
How do we *design* for complexity?
Architecture to the Rescue!
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.
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
Patterns
Tactics

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

Patterns describe holistic solutions; tactics describe “atomic” architectural strategies.
Patterns: 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
Patterns: P2P

• 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
Patterns: 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
Tactics

Scalability

Building blocks
- Modularity
- Self description
- Environment Models

Aggregation
- Self-similar structure
- Heterogeneity
- Parallelism
- Abstract Connections

Interaction
- Connection Shuffling
- Load Balancing
- Gossiping
- Tagging
Tactics: Modularity

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

• Nodes need to interact to adapt to their ever-changing state and environment.

• Neighboring nodes need to be constantly “gossipping”, exchanging topological and task-specific information.
Tactics: 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

• 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
  – Tactic: Nodes may be nested (i.e. a hierarchy). In fact, nodes exhibit self-similar (fractal) structure.
Conclusions - 1

• We began by claiming that a designer needs to *reduce, hide, shrink,* and *organize* to be able to tame complexity.

• We claim that patterns and tactics are templates for achieving these goals in a systematic way.
Conclusions - 2

By employing 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.
Readings

More references
