



Fundamentals of Software Architecture

Looking beyond the hype

Markus Völter
(voelter@acm.org)

1

Introduction





**Frustrated by
all the hype?**

The background of the slide is a photograph of a dramatic sky. The upper portion is filled with dark, swirling clouds, with a bright, overexposed area where the sun is partially visible. Below this, the horizon line is dark and hazy, suggesting a body of water or a distant landmass. The overall mood is mysterious and powerful.

If so
this presentation
is for you.



Otherwise
you should leave ☺



People
often
talk
about
technologies
instead of
architectural
concepts.

**Technology-discussions
create a lot of
accidental complexity.**



**Architectural essence
is hidden behind
tech gobbledegook**





Concepts change much
More slowly than the
Hype-driven techno
marketing

This presentation:

Timeless
concepts

This presentation:

Relationships
between them

This presentation:

Examples
languages
architecture
technology

Goals

A grayscale photograph of a vintage-style pocket compass resting on a historical map of Central America. The map shows various countries and regions with their names in Spanish. The compass has a circular dial with cardinal directions (N, S, E, W) and smaller increments. The needle is positioned near the North mark.

awareness when building systems

Goals

checklists

for reviewing systems

Goals

education of developers and architects



Concepts

**Atomic Combinable
Technology Neutral
Describable Named
(Patterns, Laws, Principles)**

Combinators

B ! A

B ⊗ A

B ▷ A



Ok, let's go...

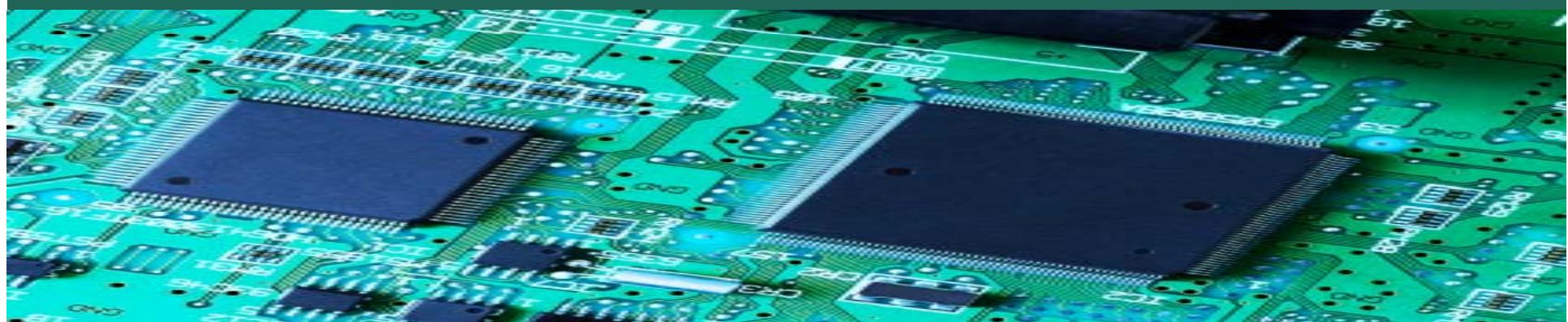
2

Breaking Things Down



Modularize

Procedures, Classes,
Components, Services, User Stories



! Modularize

Encapsulate



**Private Members
Frameworks
Facade Pattern
Components
Layers/Rings/Levels
Packed Data Wrapper**

! Modularize

Contracts

Contract

Interfaces

Pre/Post Conditions

Protocol State Machines

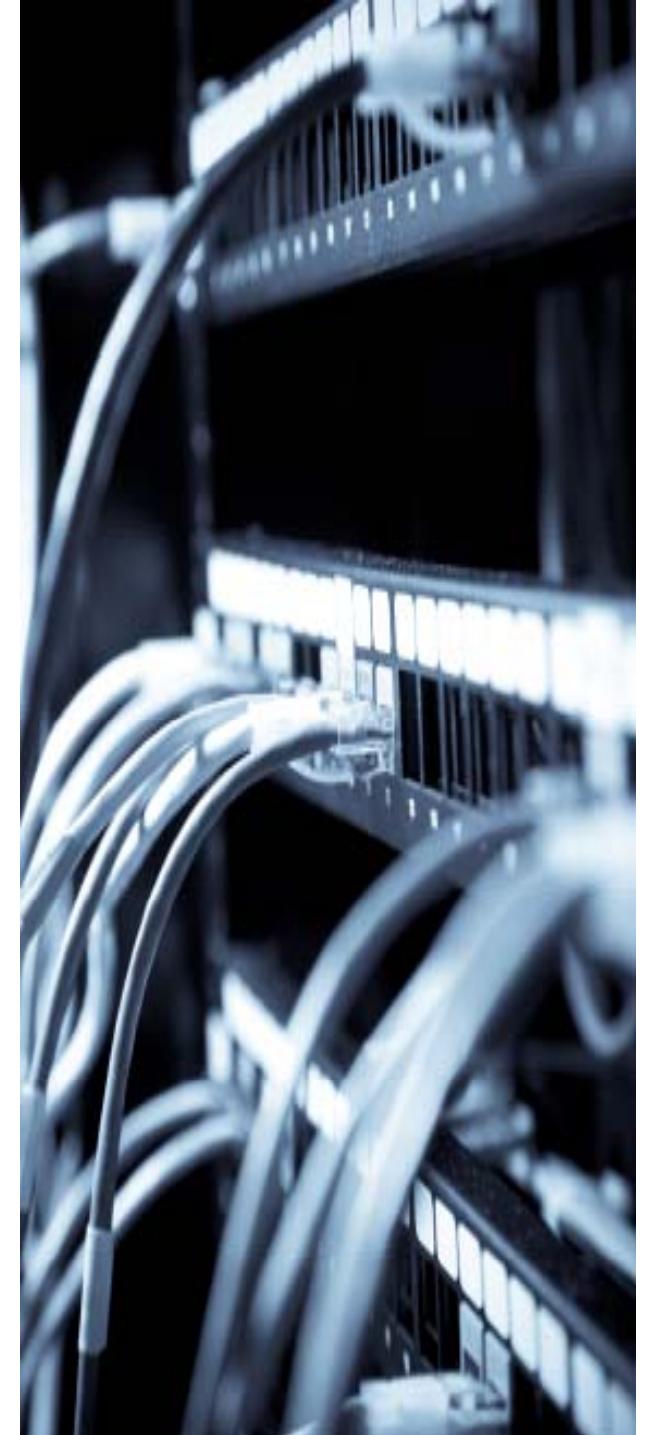
Message Exchange Patterns

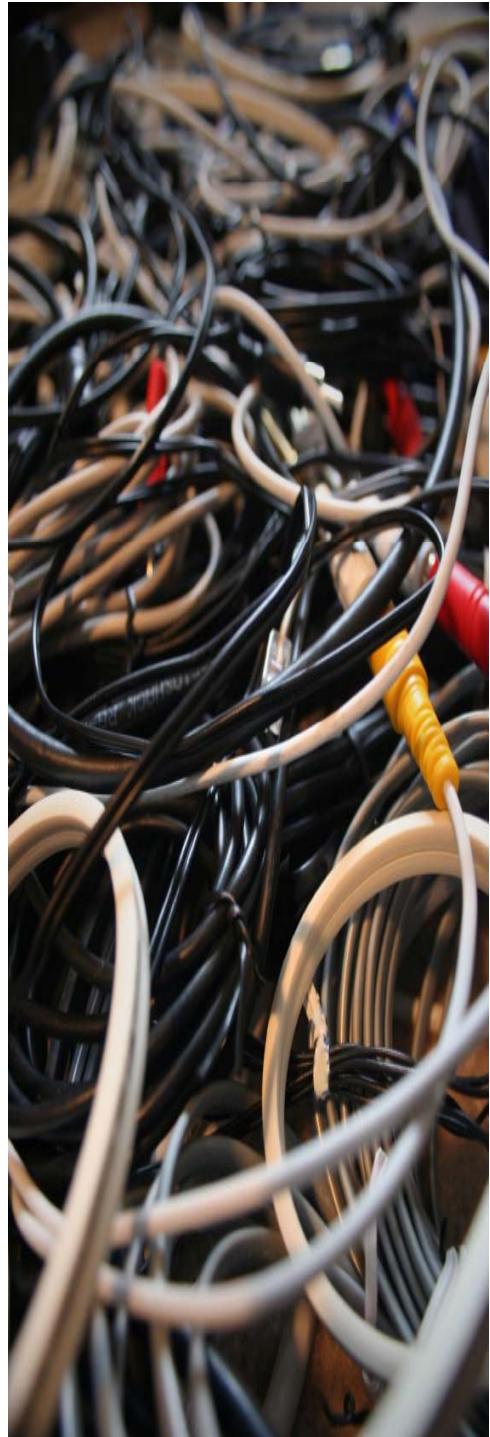
Published APIs

Decoupling

Compensating Tx
Message Queues

! Modularization





► Modularize

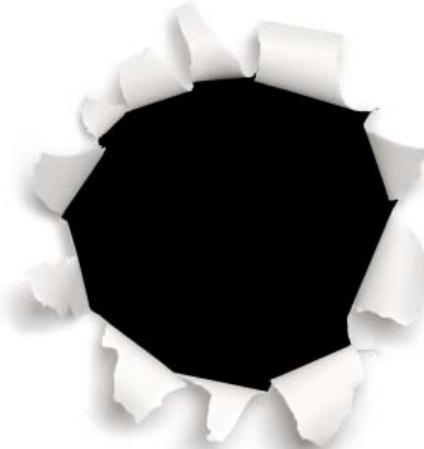
Handle Crosscuts

Aspect Orientation
Interceptors
Application Servers
Exception Handling

Go Down

⊗ Encapsulate

! Modularize



Assembler in C
C in Python/Ruby
SQL in Java

Isolate



Pure functional vs. Impure
Safety Critical Parts
Real-Time Kernel
OS Processes

! Modularize

Isolate Technology

► Isolate
! Modularize

POJOs

HALs

JBI/SCA

Code Gen/MDSD



3

Scaling things up



Parametrization

Function Arguments
Command-Line Args
Configuration Files



Simplicity

Web
Lisp
XML



Decentralization

! Contracts

The Internet

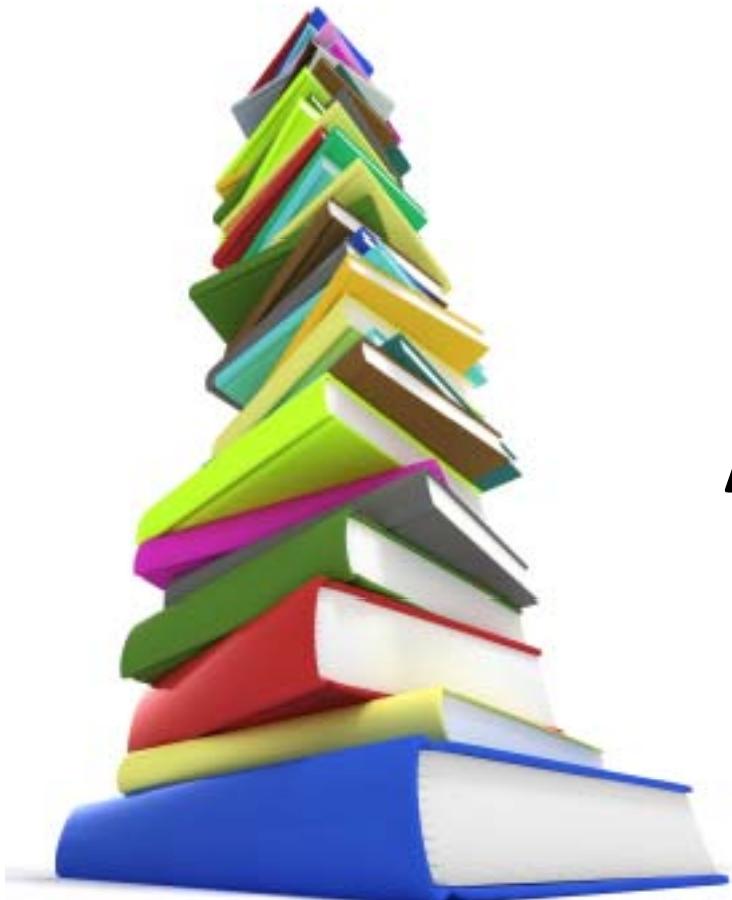
Service-Oriented Architecture

Emergent Behaviour

Bootstrapping

Languages
Compilers
IDEs

Standard Library



**Lisp (Grow A Language)
Autosar Sys Components**

Microkernel OSs

**! Modularize
! Types & Instances**

Orthogonality

Closures, Program As Data,
Macros, Higher-Order
Functions



Identity



**Pointers
GUIDs
MAC-Address
URI
Qualified Names**

4

Conceptualization





Abstraction

Operating Systems
High-Level Languages
Models, DSLs

Types & Instances

Programming Languages
Components

Models & Metamodels
RDBMS/XML Schemas



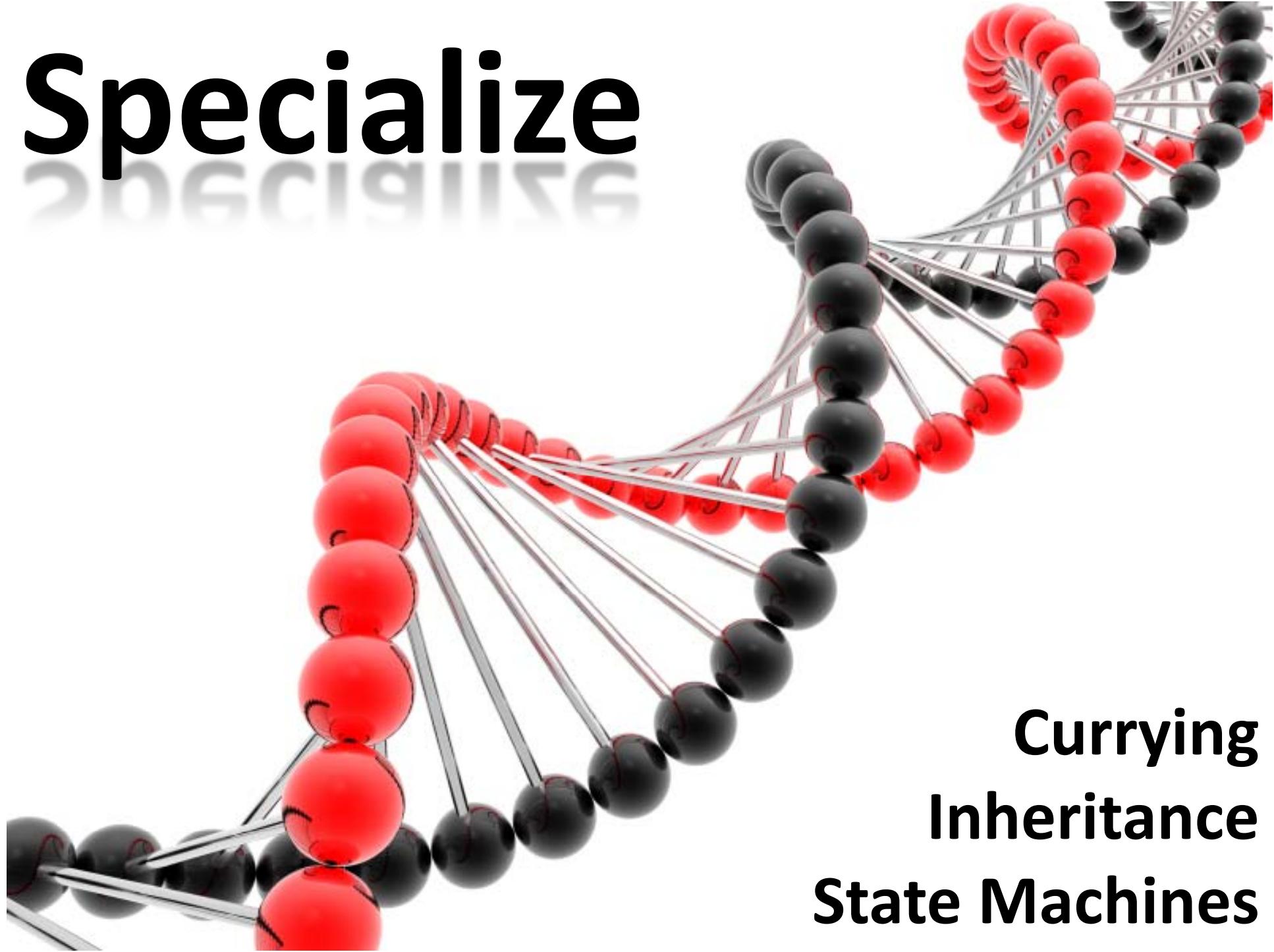
Hierarchical Decomposition

- ▶ Modularize

A photograph of the Great Pyramids of Giza under a blue sky with scattered white clouds. In the foreground, a group of people are riding camels across the sandy desert floor.

Procedures/Methods
State Machines, Components

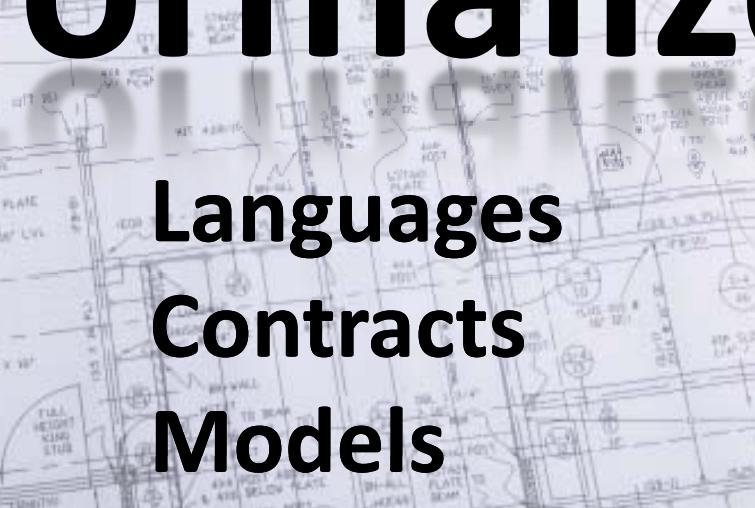
Specialize



**Currying
Inheritance
State Machines**

! Abstraction

Formalize



Languages Contracts Models State Machines

A scenic view of a coastal mountain range. In the foreground, a person wearing a hat and a green jacket stands on a stone wall, looking through binoculars. The middle ground shows a steep, rocky mountain face overlooking a town built along a coastline. The background features more mountains under a blue sky with scattered clouds.

Viewpoints

! Formalization

Configuration Files

4+1 Model

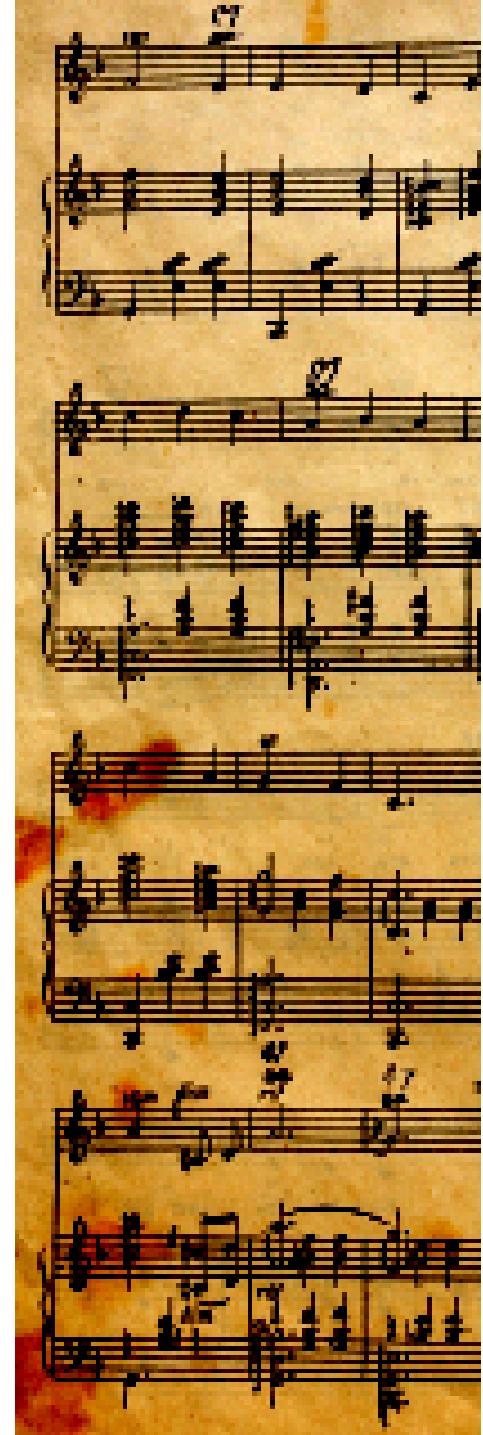
Blackbox/Whitebox

Types/Instances/Deployment

! Formalization

Notation

UML
Lisp
Java
Ruby



A photograph of an airplane wing and tail against a cloudy sky at sunset. The sky is a mix of blue, orange, and pink hues. The airplane's wing is visible in the foreground, curving upwards towards the right. The tail is also visible, showing a red, white, and blue vertical stabilizer.

! Modularize

Go Meta

**Translators, Reflection,
Meta Programming, AOP**

Reflection

! Go Meta

Languages (Lisp, Smalltalk, Ruby)
Embedded Systems (Static)

! Formalization

! Go Meta



Translate

I L Q U I ? I 9 r G

Compilers, Transformers, Generators,
Macros (Lisp, Converge)

! Formalization

! Go Meta

Interpret

Business Process Engines
Data Driven Systems
(Dynamic) Languages

Tracking



Impurity in Haskell: io(...)
Tainting (Static Analysis)
Session State
ACT Pattern

Automate

Build, Test, Translate



! Formalization

5

Dos and Don'ts



! Formalization

Protocols

Protocol?

Transactions
Locking/Synchronization
Resource Access





⊗ Make Explicit

**DOC Middleware
Orthogonal Persistence
(OR Mappers)**

**Make
Transparent**

⊗ Make Transparent



Make Explicit
Dependencies
SOA, Messaging
Functional Programming
PLE Variabilities
Persistence: Loading Data

Software Architecture

DSL: expressiveness

MDSD: skeletons

Scade/SystemC

**Limit
Freedom**



! Formalization

! Go Meta

Declaration

Implementation

App Servers (EJB), Plugin RT (Eclipse)
Models, Transactional Memory

Test semantics, not syntax (code gen)
Higher Order Functions (map, foreach)
Transactional Memory

Don't
Overspecify



Avoid

Sideeffects

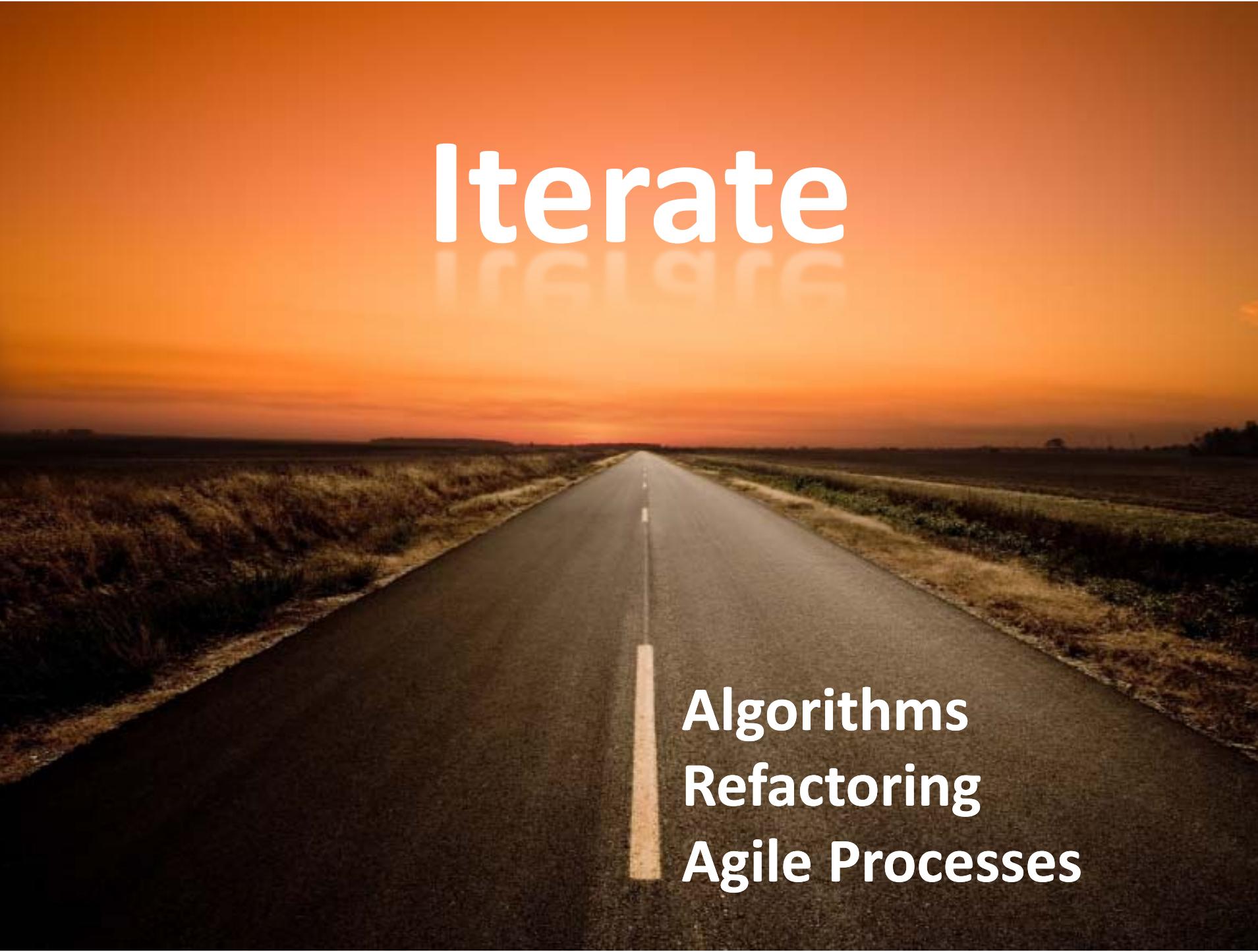
► Limit Freedom

Functional Programming
Concurrency (Sharing)
Distribution

6

Process





Iterate

Integrate

Algorithms
Refactoring
Agile Processes

Capture Best Practices

**Patterns, DSLs,
Models, Translators**



More I

Ownership vs. Reference

Build Languages

Cohesion and Coupling (do one thing right, composability)

Indirection (polymorphism, mem references for compaction, naming service)

Prevention vs. Compensation

More II

Build Platforms

Versioning

Pessimistic/Optimistic/Compensating

Localize (Sync in MP, UML-M2M)

Container (AppServer)

Lazy/On-demand, Eager

Self Modification (Meta Prog, MOPs,

Embedded Optimization

Measure: metrics, performance tuning, scalability,
test coverage



What do you think?
More Fundamentals?
More Examples?

Please let me know!
voelter@acm.org





THE END.