Fundamentals
of Software Architecture

Looking beyond the hype

Markus Völter
(voelter@acm.org)
1

Introduction
Frustrated by all the hype?
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 gobbledygook
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
awareness when building systems

Goals
checklists for reviewing systems

Goals
education of developers and architects

Goals
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
Contracts

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

POJOs
HALs
JBI/SCA
Code Gen/MDSD

Modularize
3

Scaling things up
Parametrization

Function Arguments
Command-Line Args
Configuration Files

! Modularization
Simplicity

Web
Lisp
XML
Decentralization

The Internet Emergent Behaviour Service-Oriented Architecture

Contracts
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
Conceptualization
Abstraction

Operating Systems
High-Level Languages
Models, DSLs
Types & Instances

Programming Languages
Components
Models & Metamodels
RDBMS/XML Schemas
Hierarchical Decomposition

- Modularize

Procedures/Methods
State Machines, Components
Formalize

Languages
Contracts
Models
State Machines
Viewpoints

Configuration Files
4+1 Model
Blackbox/Whitebox
Types/Instances/Deployment

Formalization
Formalization

Notation

UML
Lisp
Java
Ruby
Go Meta

Translators, Reflection, Meta Programming, AOP
Reflection

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

Go Meta

Translate

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

Business Process Engines
Data Driven Systems
(Dynamic) Languages
Tracking

Impurity in Haskell: \texttt{io(\ldots)}
Tainting (Static Analysis)
Session State
ACT Pattern
Automate
Build, Test, Translate

! Formalization
5

Dos and Don’ts
Protocols

Transactions

Locking/Synchronization

Resource Access

Formalization
DOC Middleware
Orthogonal Persistence
(OR Mappers)
Make Explicit

- Dependencies
- SOA, Messaging
- Functional Programming
- PLE Variabilities
- Persistence: Loading Data
Software Architecture
DSL: expressiveness
MDSD: skeletons
Scade/SystemC

Limit
Freedom
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 Limit Freedom

Avoid Sideeffects

Functional Programming
Concurrency (Sharing)
Distribution
6

Process
Iterate

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.