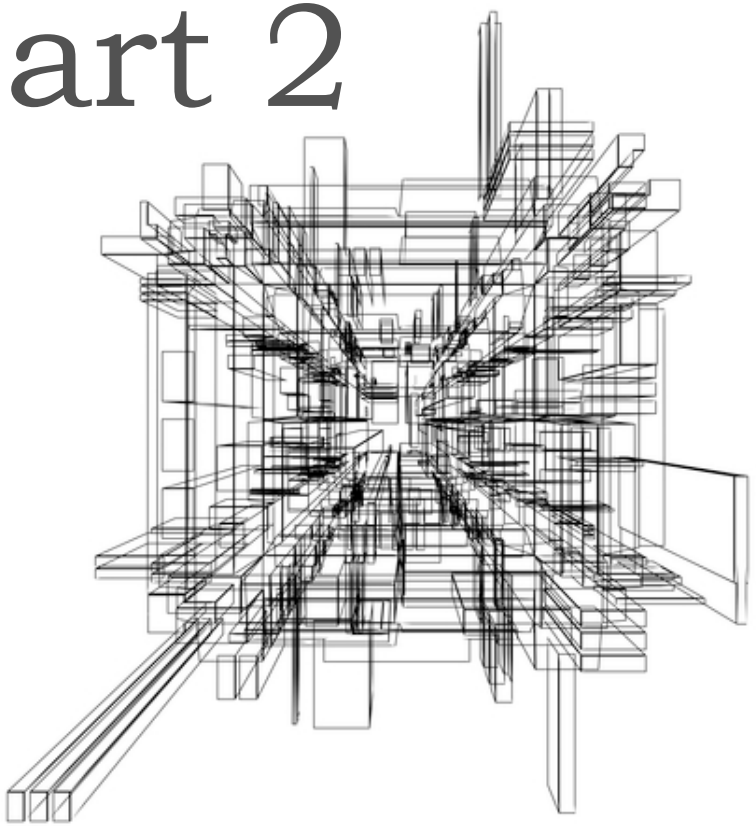
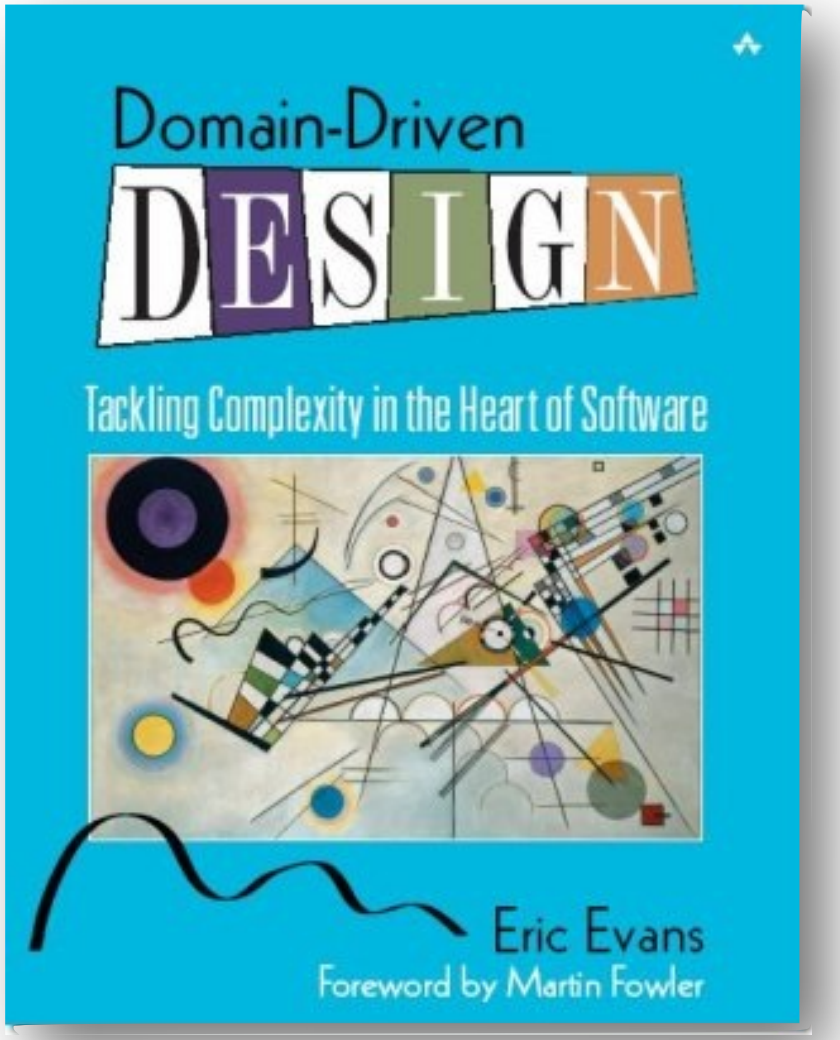
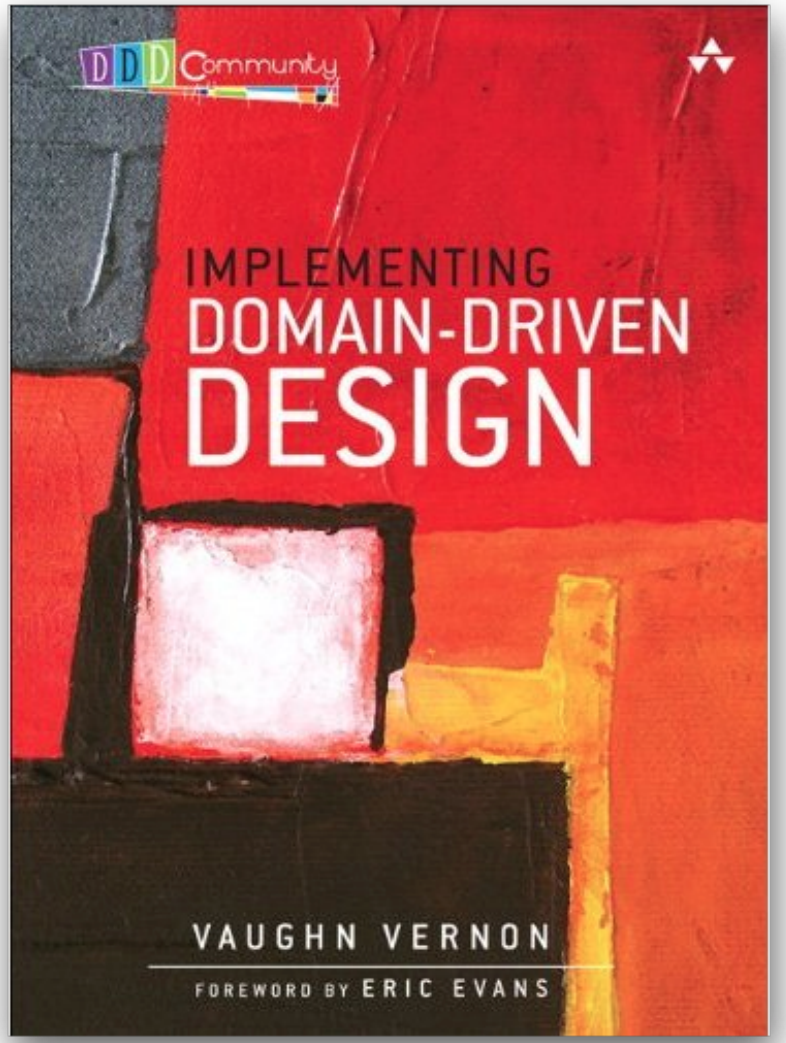
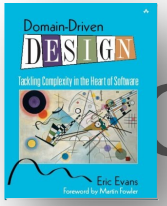
continuous delivery for architects part 2



Domain Driven Design



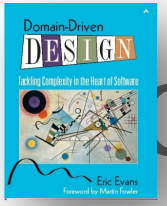




core definitions

Domain: A sphere of knowledge (ontology), influence, or activity.

Model: A system of abstractions that describes selected aspects of a domain and can be used to solve problems related to that domain.

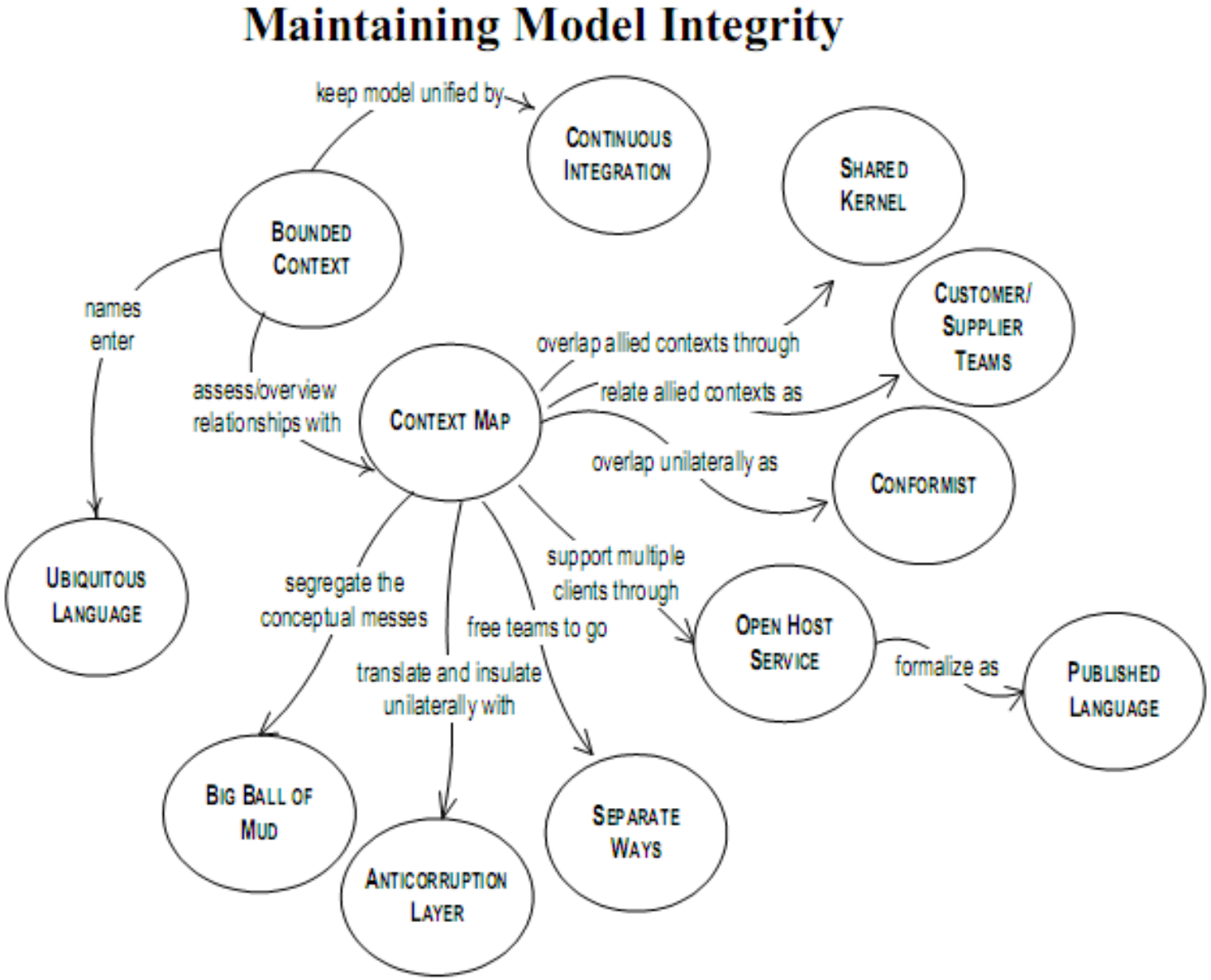


core definitions

Ubiquitous Language: A language structured around the domain model and used by all team members to connect all the activities of the team with the software.

Context: The setting in which a word or statement appears that determines its meaning.

bounded context



bounded context

Explicitly set boundaries in terms of team organization, usage within specific parts of the application, and physical manifestations such as code bases and database schemas.

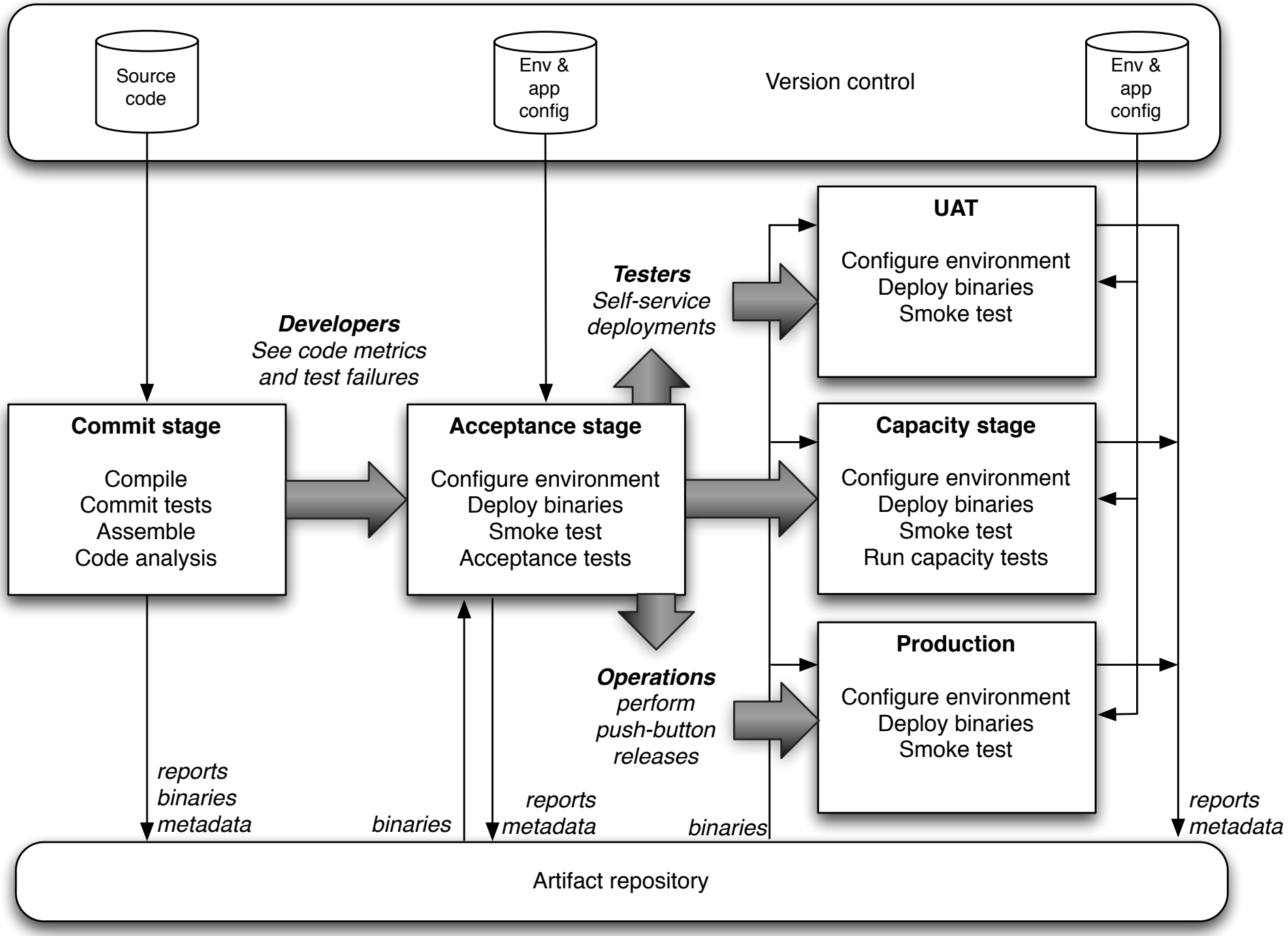
Keep the model strictly consistent within these bounds, but don't be distracted or confused by issues outside.

logical & *physical* context

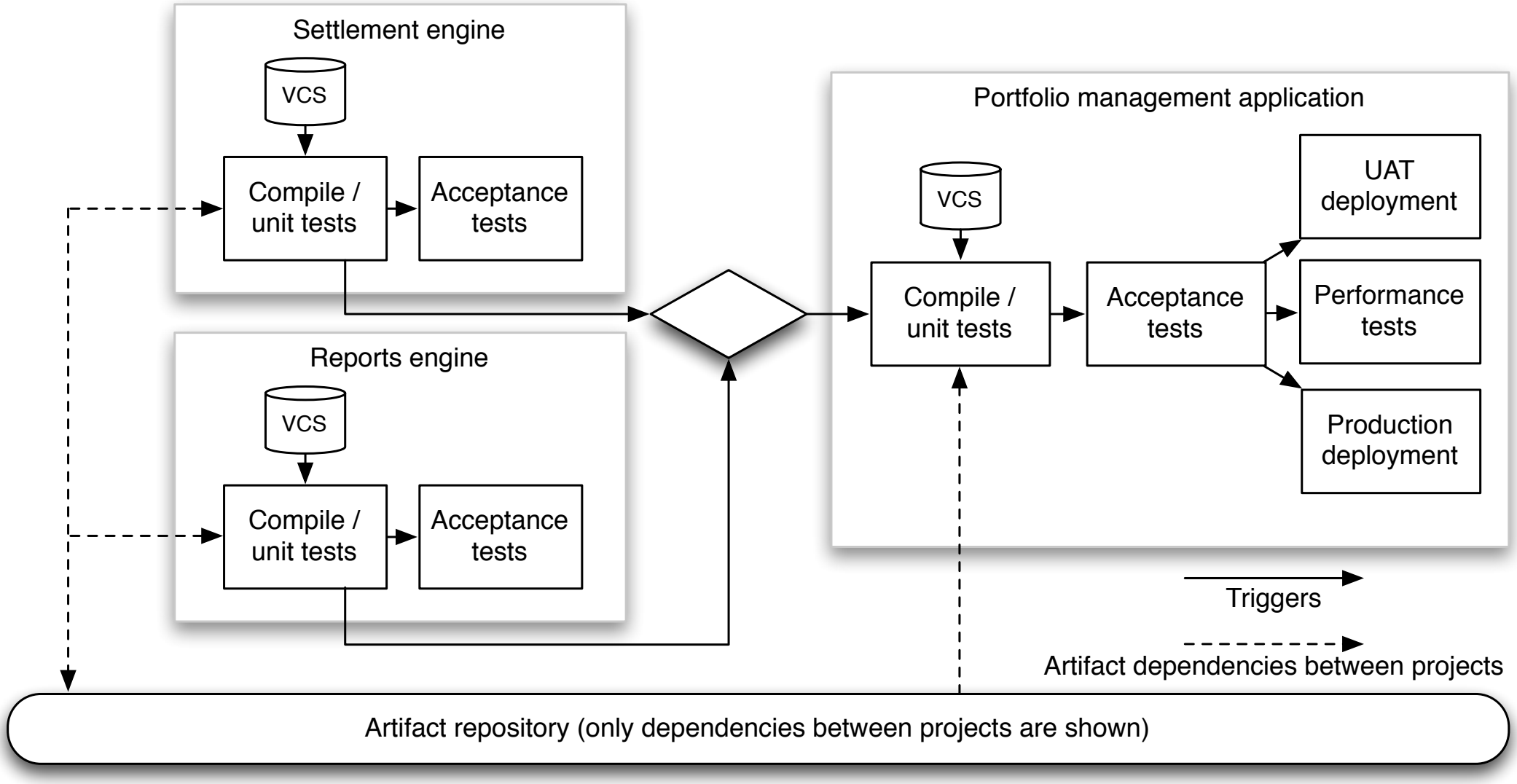
migrate from shared resources to bounded context

component/services encompasses operationalability

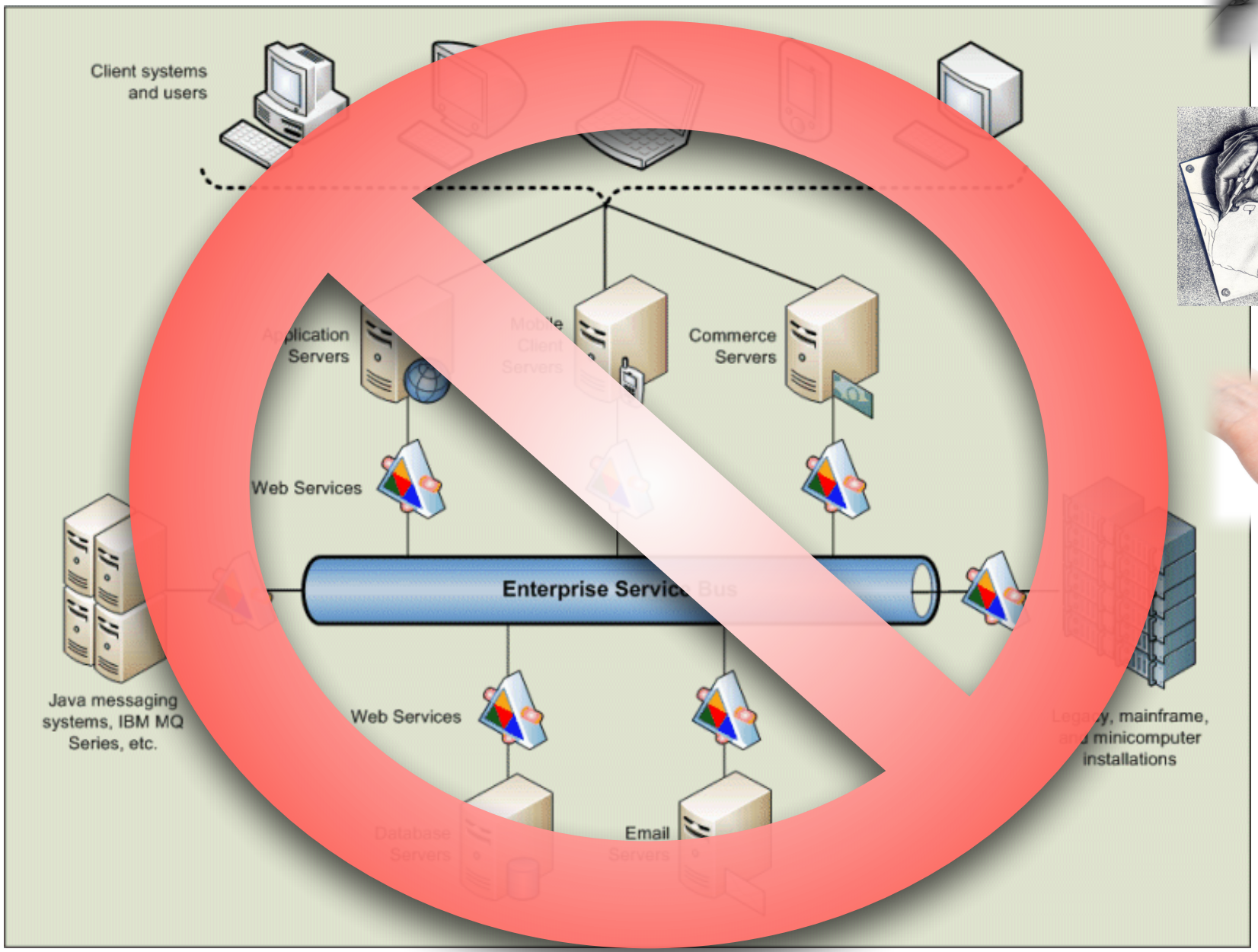
minimizes or eliminates logical *& physical* coupling

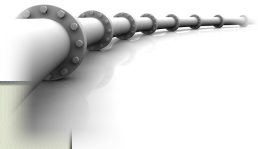

Source code
Env & app config
Version control
Env & app config
UAT
Configure environment
Deploy binaries
Smoke test
Testers
Self-service deployments
Developers
See code metrics and test failures
Commit stage
Compile
Commit tests
Assemble
Code analysis
Acceptance stage
Configure environment
Deploy binaries
Smoke test
Acceptance tests
Capacity stage
Configure environment
Deploy binaries
Smoke test
Run capacity tests
Operations
perform push-button releases
Production
Configure environment
Deploy binaries
Smoke test
reports
binaries
metadata
binaries
reports
metadata
binaries
reports
metadata
Artifact repository

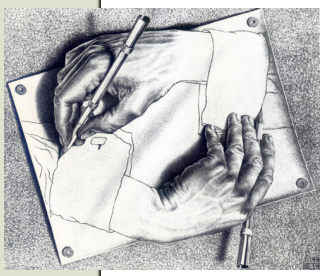
component pipeline



death by coupling!







prefer choreography to orchestration

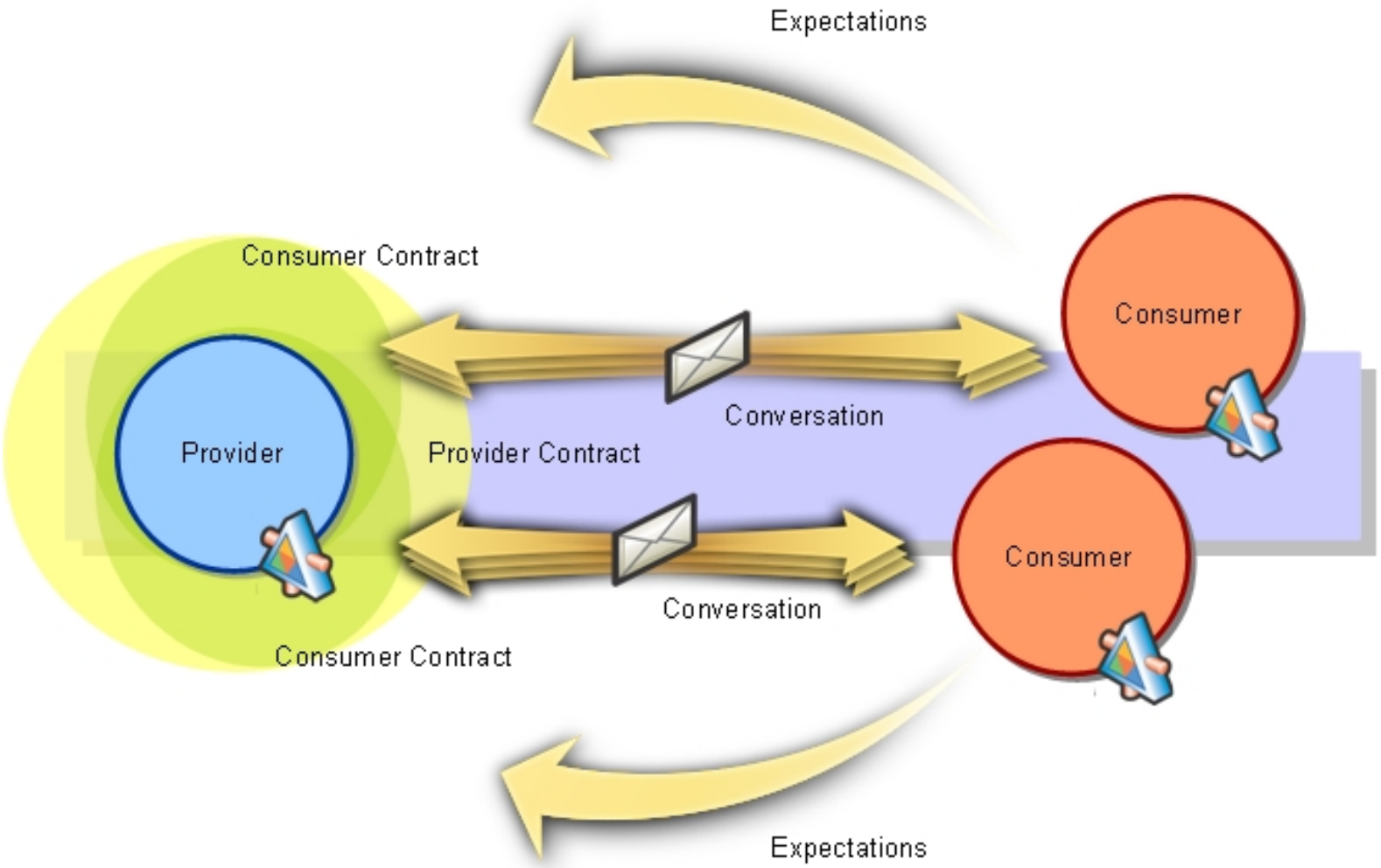
allow endpoints to negotiate protocols

allows endpoints to evolve independently

use consumer-driven contracts to ensure continued viability

consumer-driven contracts

http://martinfowler.com/articles/consumerDrivenContracts.html



Use components to decouple parts of your application that change at different rates.

components: compile-time dependencies

services: run-time dependencies

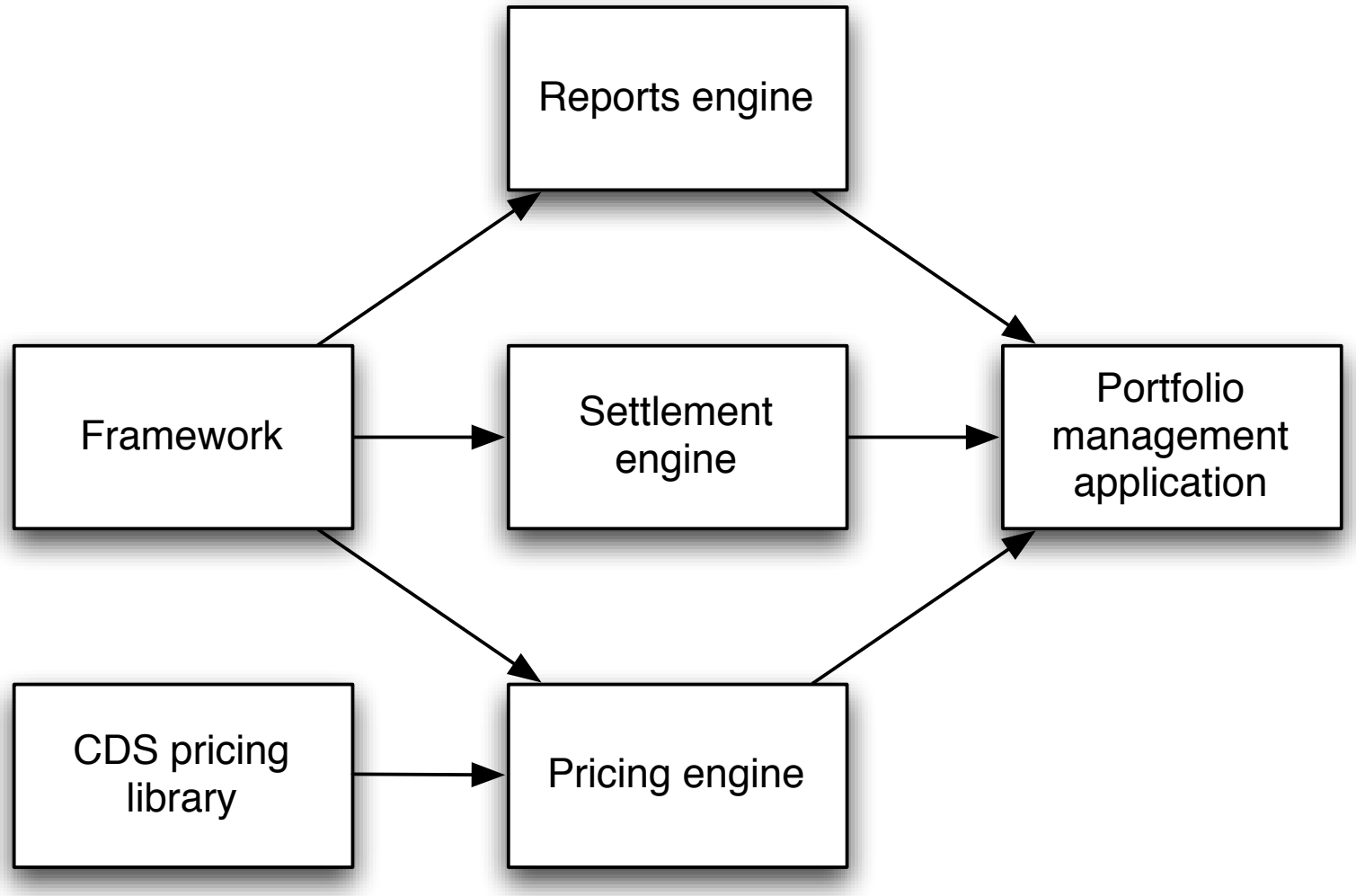
components / services

represent different rates of change and/or lifecycle

clean delineation of responsibility

degrees of freedom in optimizing build & deployment process

components


Reports engine
Framework
Settlement engine
Portfolio management application
CDS pricing library
Pricing engine

managing dependencies

"DLL Hell"

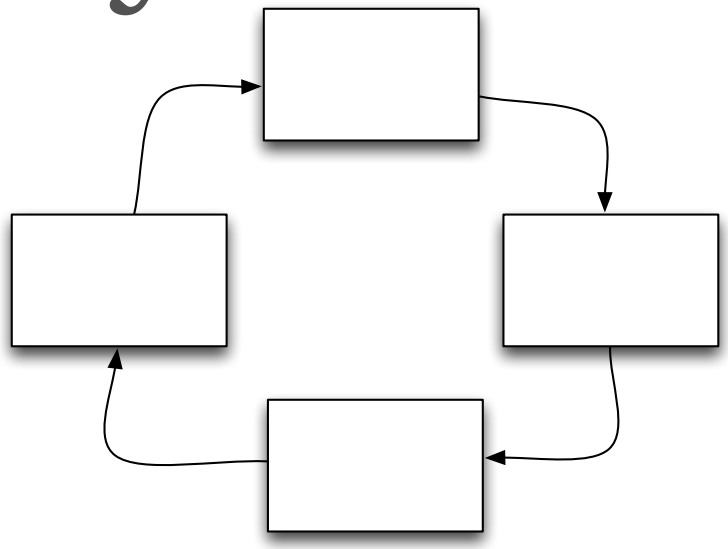
2 approaches:

the ubiquitous lib directory

transitive dependency management tool*

*all platforms (eventually) rely on tooling

anti-pattern: cycles



circular dependencies

Java tooling promotes/ignores cycles

makes the system hard to componentize

makes lifecycle more complex

```
<JarAnalyzer>
 - <Jars>
   - <Jar name="antlr.jar">
     - <Summary>
       - <Statistics>
           <ClassCount>210</ClassCount>
           <AbstractClassCount>48</AbstractClassCount>
           <PackageCount>10</PackageCount>
           <Level>1</Level>
         </Statistics>
       - <Metrics>
           <Abstractness>0.23</Abstractness>
           <Efferent>0</Efferent>
           <Afferent>1</Afferent>
           <Instability>0.00</Instability>
           <Distance>0.77</Distance>
         </Metrics>
       - <Packages>
           <Package>antlr</Package>
           <Package>antlr.build</Package>
           <Package>antlr.collections</Package>
           <Package>antlr.debug</Package>
           <Package>antlr.preprocessor</Package>
           <Package>antlr.actions.cpp</Package>
           <Package>antlr.actions.csharp</Package>
           <Package>antlr.actions.java</Package>
           <Package>antlr.collections.impl</Package>
           <Package>antlr.debug.misc</Package>
         </Packages>
         <OutgoingDependencies> </OutgoingDependencies>
       - <IncomingDependencies>
           <Jar>struts.jar</Jar>
         </IncomingDependencies>
         <Cycles> </Cycles>
         <UnresolvedDependencies> </UnresolvedDependencies>
       </Summary>
     </Jar>
   - <Jar name="commons-beanutils.jar">
     - <Summary>
       - <Statistics>
           <ClassCount>66</ClassCount>
           <AbstractClassCount>7</AbstractClassCount>
           <PackageCount>4</PackageCount>
           <Level>2</Level>
         </Statistics>
       - <Metrics>
           <Abstractness>0.11</Abstractness>
```

JarAnalyzer Analysis

Run with JarAnalyzer on

Summary

[summary] [jars] [cycles] [explanations]

Jar Name	Total Classes	Abstract Classes	Packages	Level	Abstractness	Efferent	Afferent	Instability	Distance
antlr.jar	210	48	10	1	0.23	0	1	0.00	0.77
commons-beanutils.jar	66	7	4	2	0.11	2	3	0.40	0.49
commons-collections.jar	187	15	3	1	0.08	0	4	0.00	0.92
commons-digester.jar	55	9	3	3	0.16	3	2	0.60	0.24
commons-fileupload.jar	16	4	1	1	0.25	0	1	0.00	0.75
commons-logging.jar	18	2	2	1	0.11	0	4	0.00	0.89
commons-validator.jar	30	1	2	4	0.03	5	1	0.83	0.14
jakarta-oro.jar	62	13	6	1	0.21	0	1	0.00	0.79
struts.jar	289	33	25	5	0.11	7	0	1.00	0.11

Jars

[summary] [jars] [cycles] [explanations]

antlr.jar

Level: 1 Afferent Couplings: 1 Efferent Couplings: 0 Abstractness: 0.23 Instability: 0.00 Distance: 0.77

Uses Jars	Used by Jars	Cycles With
None	struts.jar	*None*

Packages within jar	Unresolved Packages
antlr antlr.build antlr.collections antlr.debug antlr.preprocessor antlr.actions.cpp antlr.actions.csharp antlr.actions.java antlr.collections.impl antlr.debug.misc	*None*

commons-beanutils.jar

Level: 2 Afferent Couplings: 3 Efferent Couplings: 2 Abstractness: 0.11 Instability: 0.40 Distance: 0.49

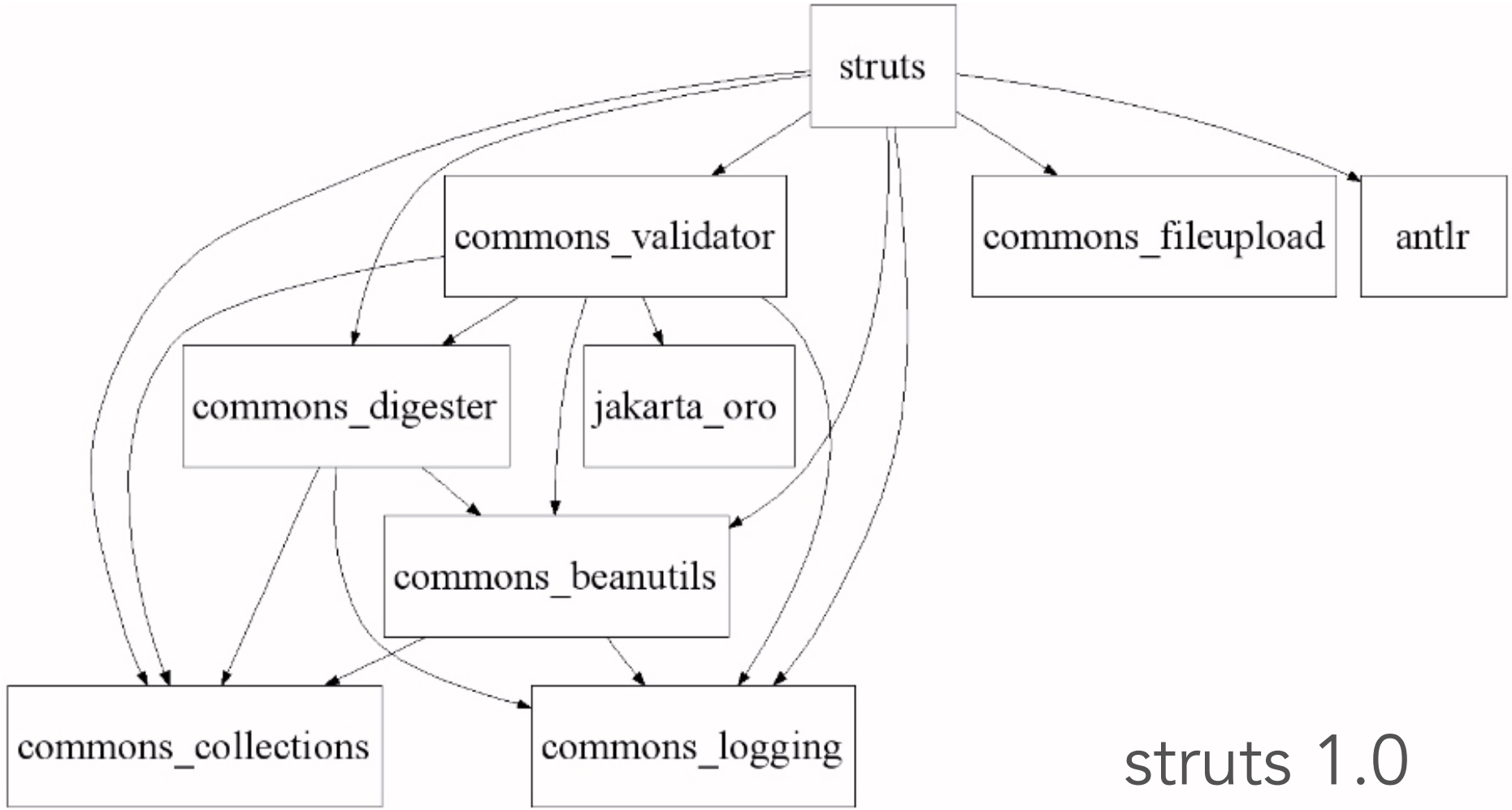
Uses Jars	Used by Jars	Cycles With
commons-collections.jar commons-logging.jar	commons-digester.jar commons-validator.jar struts.jar	*None*

Packages within jar	Unresolved Packages
org.apache.commons.beanutils.converters org.apache.commons.beanutils org.apache.commons.beanutils.locale.converters org.apache.commons.beanutils.locale	*None*

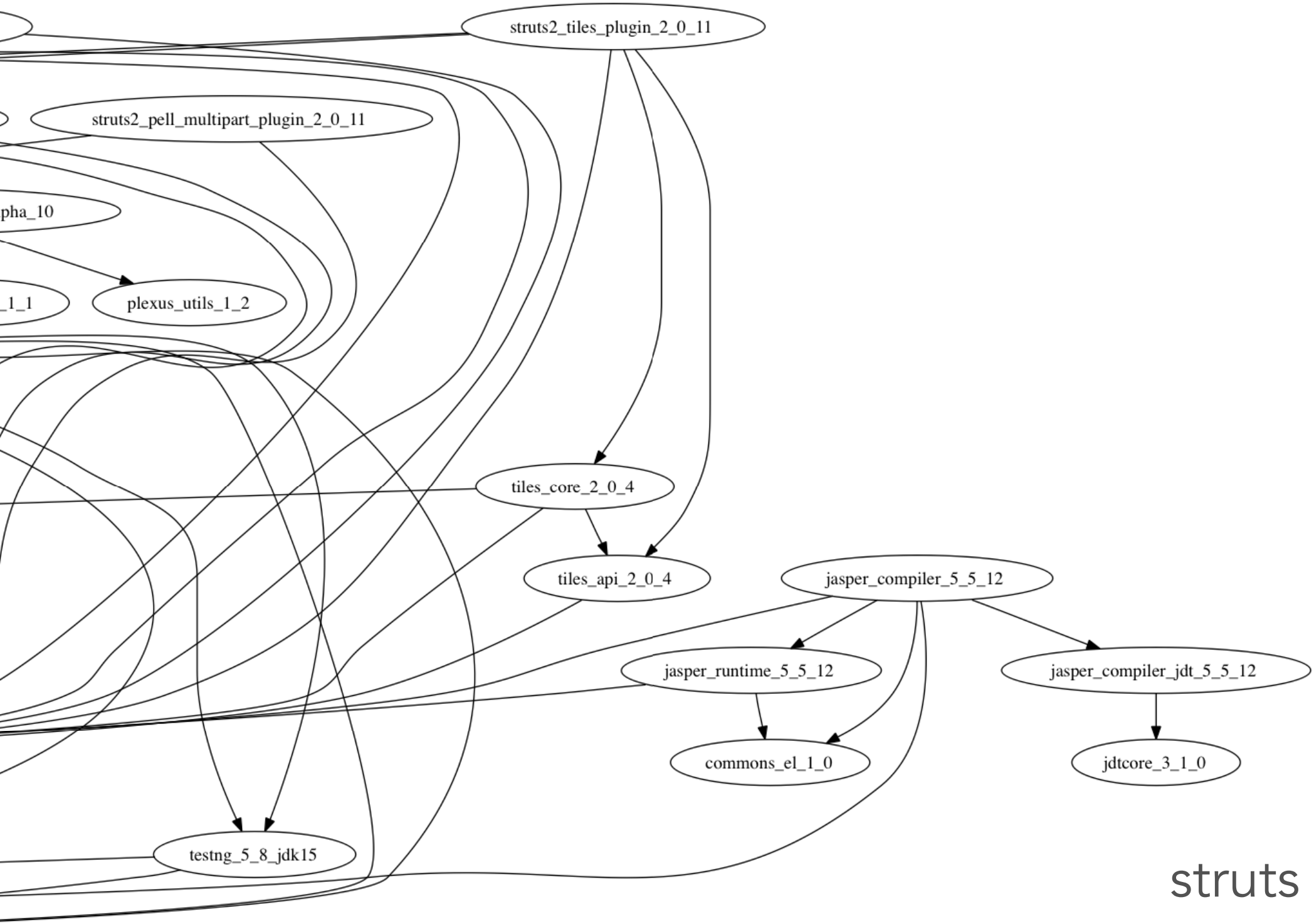
Kirk Knoernschild

http://techdistrict.kirkk.com/

consequences of ignoring



struts 1.0


struts2_tiles_plugin_2_0_11
struts2_pell_multipart_plugin_2_0_11
pha_10
_1_1
plexus_utils_1_2
tiles_core_2_0_4
tiles_api_2_0_4
jasper_compiler_5_5_12
jasper_runtime_5_5_12
jasper_compiler_jdt_5_5_12
commons_el_1_0
jdtcore_3_1_0
testng_5_8_jdk15

struts 2


s2_config_browser_plugin_2_0_11
struts2_jfreechart_plugin_2_0_11
struts2_jasperreports_plugin_2_0_11
struts2_sitegraph_plugin_2
struts2_spring_plugin_2_0_11
struts2_core_2_0_11
struts2_plexus_plugin_2_0_11
dwr_1_1_beta_3
xwork_2_0_4
spring_web_1_2_8
ognl_2_6_11
spring_mock_1_2_8
spring_context_1_2_8
antlr_2_7_2
commons_fileupload_1_1_1
spring_aop_1_2_8
commons_io_1_0
spring_beans_1_2_8
spring_core_1_2_8
org_
commons_logging_1_0_4
junit_3_8_1

Structure 101

http://structure101.com/

analyzes codebase

provides “to do” list of refactorings

allows developers to untangle cycles

JDepend dependency constraint

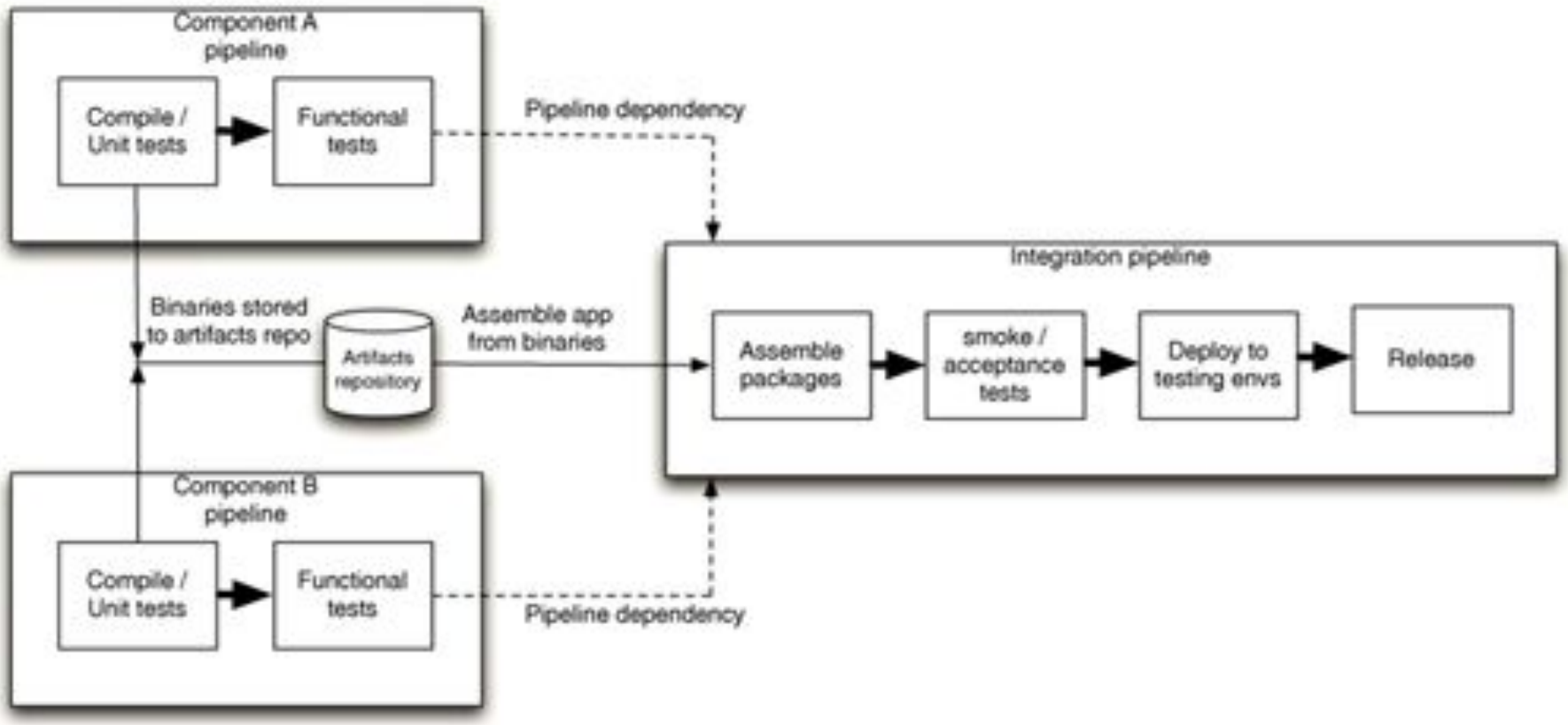
http://clarkware.com/software/JDepend.html

```
protected void setUp() throws IOException {
    jdepend = new JDepend();
    jdepend.addDirectory("/path/to/project/util/classes");
    jdepend.addDirectory("/path/to/project/ejb/classes");
    jdepend.addDirectory("/path/to/project/web/classes");
}


public void testMatch() {
    DependencyConstraint constraint = new DependencyConstraint();
    JavaPackage ejb  = constraint.addPackage("com.xyz.ejb");
    JavaPackage web  = constraint.addPackage("com.xyz.web");
    JavaPackage util = constraint.addPackage("com.xyz.util");

    ejb.dependsUpon(util);
    web.dependsUpon(util);

    jdepend.analyze();

    assertEquals("Dependency mismatch",
            true, jdepend.dependencyMatch(constraint));
}
```

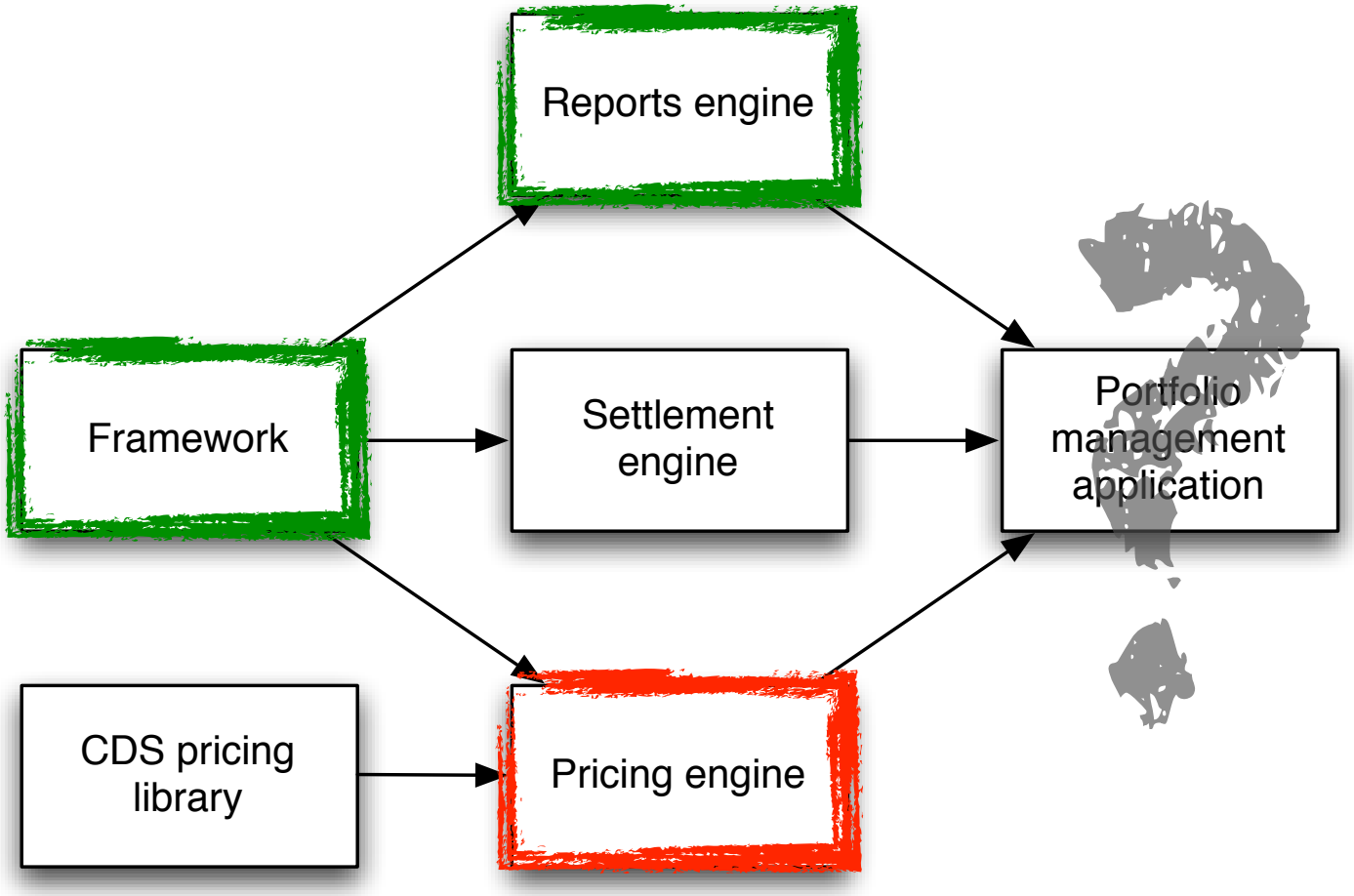
dependency cycle

```
/**
 * Tests that a single package does not contain
 * any package dependency cycles.
 */
public void testOnePackage() {

    jdepend.analyze();

    JavaPackage p = jdepend.getPackage("com.xyz.ejb");

    assertEquals("Cycle exists: " + p.getName(),
           false, p.containsCycle());
}

/**
 * Tests that a package dependency cycle does not
 * exist for any of the analyzed packages.
 */
public void testAllPackages() {

    Collection packages = jdepend.analyze();

    assertEquals("Cycles exist",
                 false, jdepend.containsCycles());
}
```

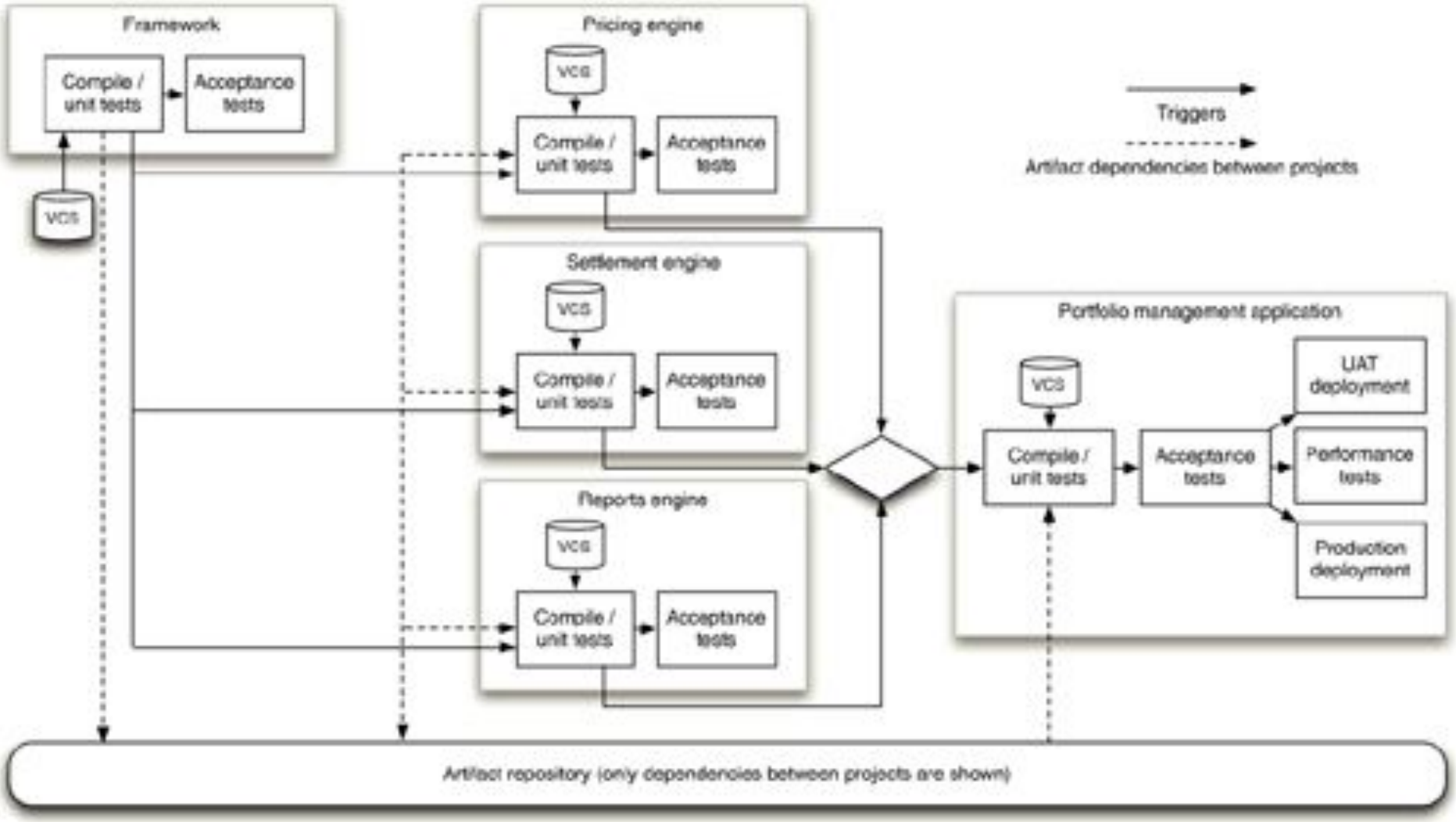
pipelining components



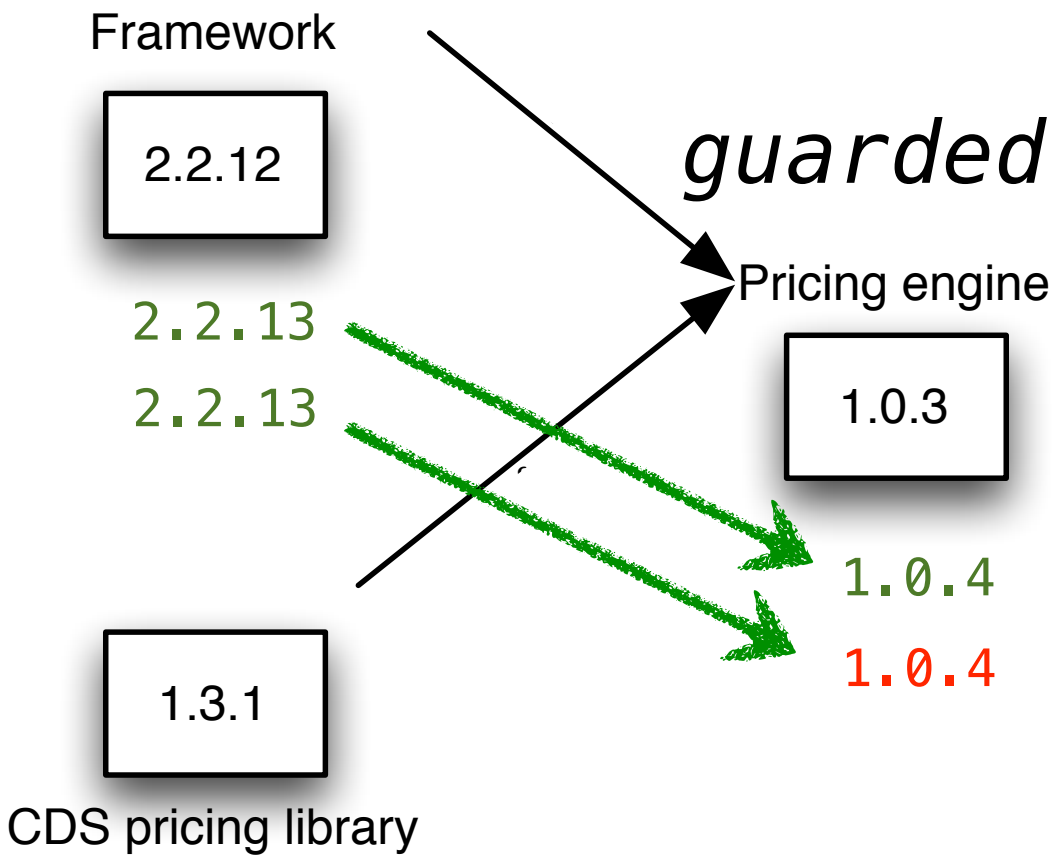
anti-pattern: diamond dependencies



pipelining components



upstream trust?



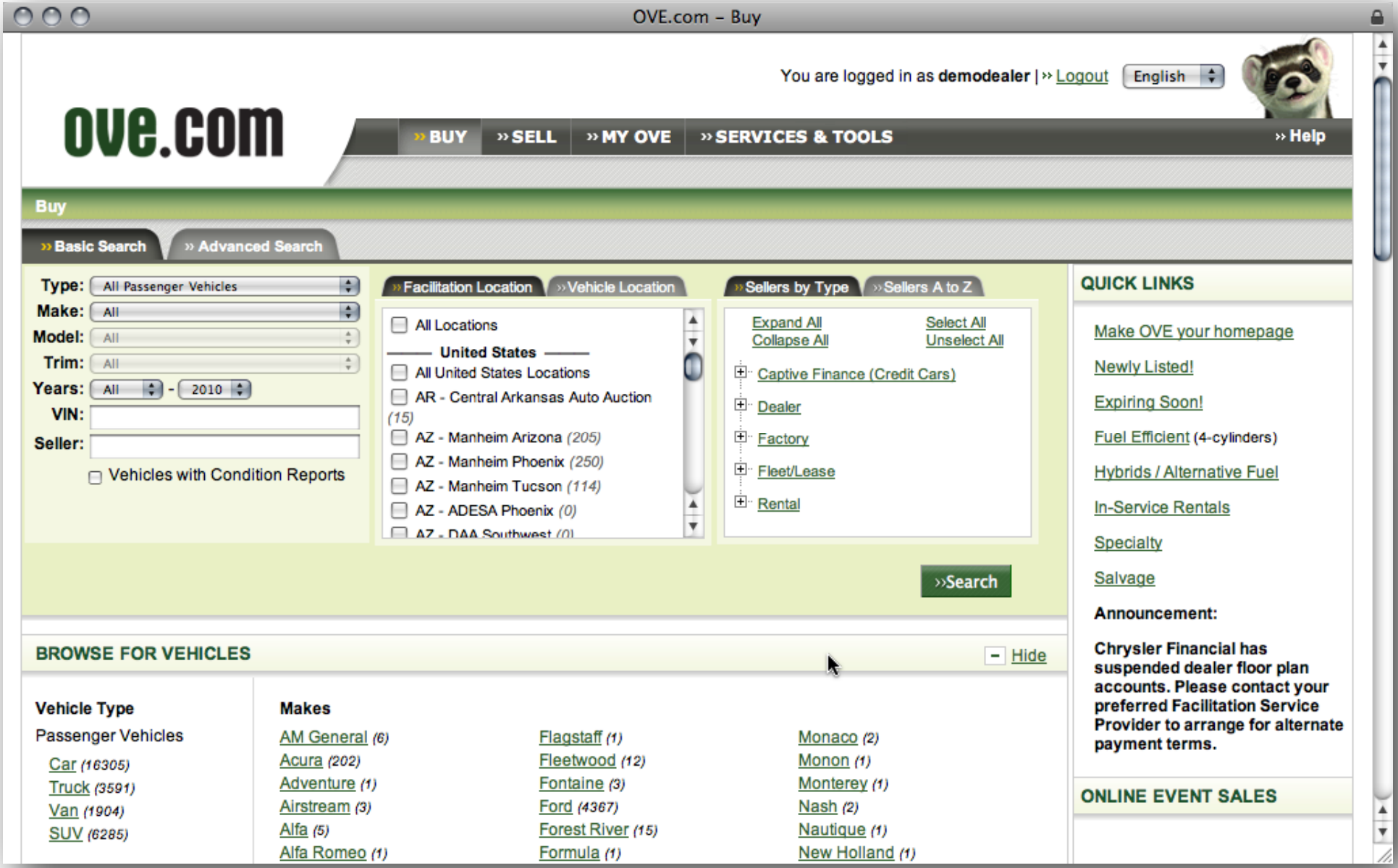
service dependencies

prefer REST over SOAP

try to avoid versioning endpoints

use semantic versioning instead

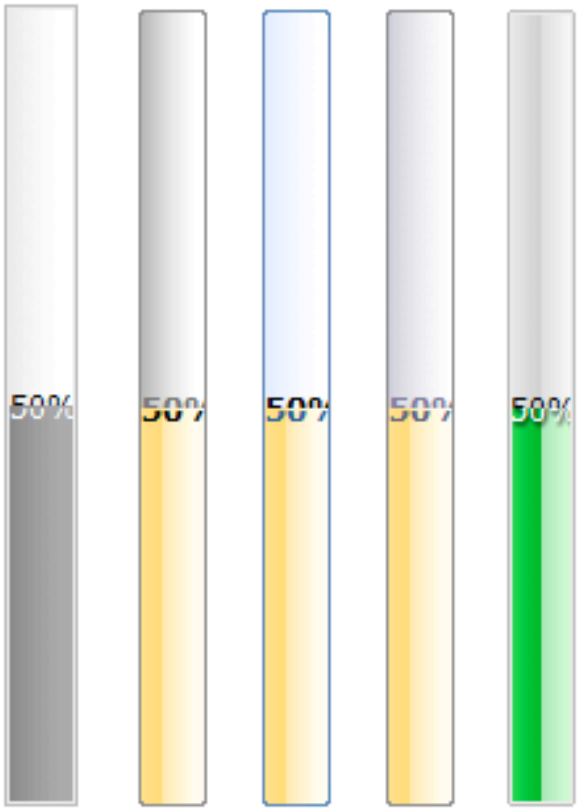
use expand/contract to support multiple version temporarily

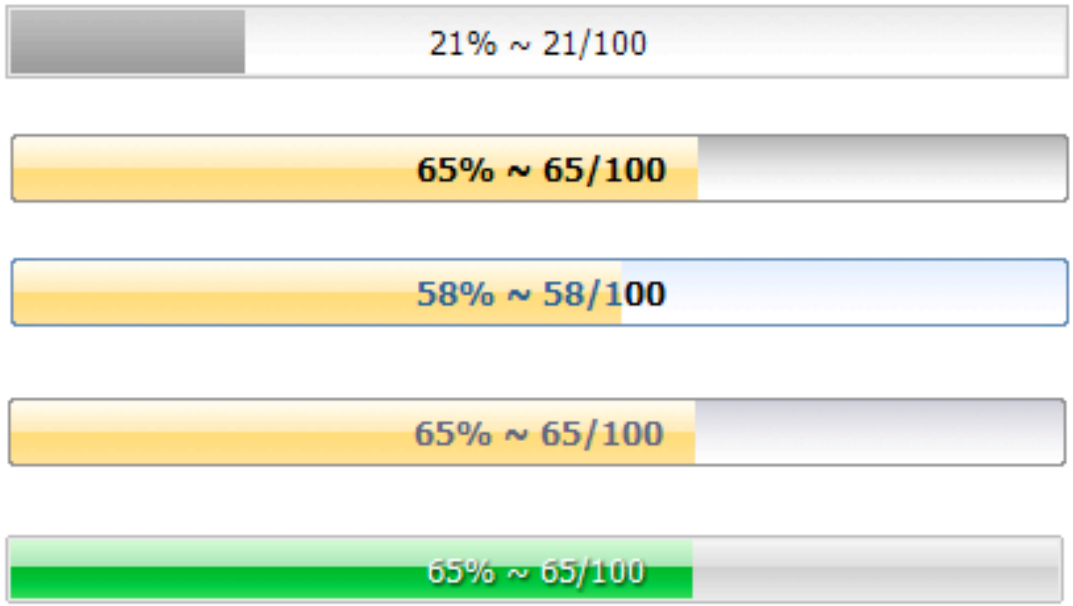

OVE.com - Buy
You are logged in as demodealer | ›› Logout
English
ove.com
›› BUY
›› SELL
›› MY OVE
›› SERVICES & TOOLS
›› Help
Buy
›› Basic Search
›› Advanced Search
Type: All Passenger Vehicles
Make: All
Model: All
Trim: All
Years: All - 2010
VIN:
Seller:
Vehicles with Condition Reports
›› Facilitation Location
›› Vehicle Location
All Locations
United States
All United States Locations
AR - Central Arkansas Auto Auction (15)
AZ - Manheim Arizona (205)
AZ - Manheim Phoenix (250)
AZ - Manheim Tucson (114)
AZ - ADESA Phoenix (0)
›› Sellers by Type
›› Sellers A to Z
Expand All
Collapse All
Select All
Unselect All
Captive Finance (Credit Cars)
Dealer
Factory
Fleet/Lease
Rental
›› Search
QUICK LINKS
Make OVE your homepage
Newly Listed!
Expiring Soon!
Fuel Efficient (4-cylinders)
Hybrids / Alternative Fuel
In-Service Rentals
Specialty
Salvage
Announcement:
Chrysler Financial has suspended dealer floor plan accounts. Please contact your preferred Facilitation Service Provider to arrange for alternate payment terms.
ONLINE EVENT SALES
BROWSE FOR VEHICLES
Hide
Vehicle Type
Passenger Vehicles
Car (16305)
Truck (3591)
Van (1904)
SUV (6285)
Makes
AM General (6)
Acura (202)
Adventure (1)
Airstream (3)
Alfa (5)
Alfa Romeo (1)
Flagstaff (1)
Fleetwood (12)
Fontaine (3)
Ford (4367)
Forest River (15)
Formula (1)
Monaco (2)
Monon (1)
Monterey (1)
Nash (2)
Nautique (1)
New Holland (1)

case study

evolution of asynchronous messaging

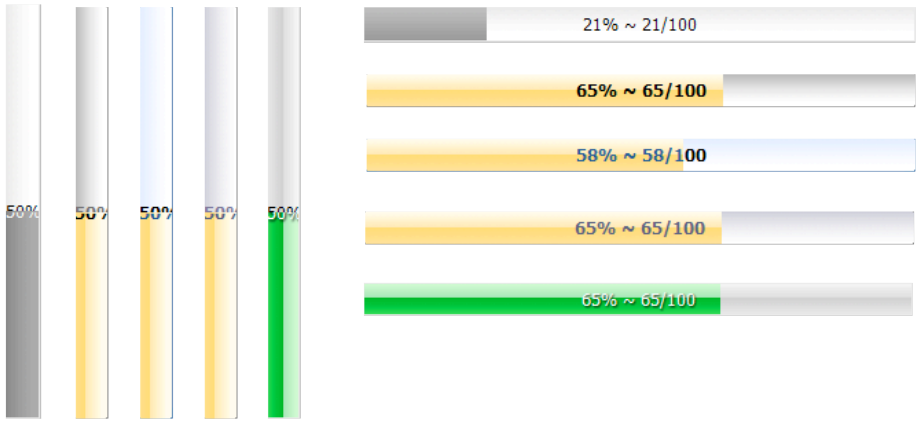
progress bars & async upload



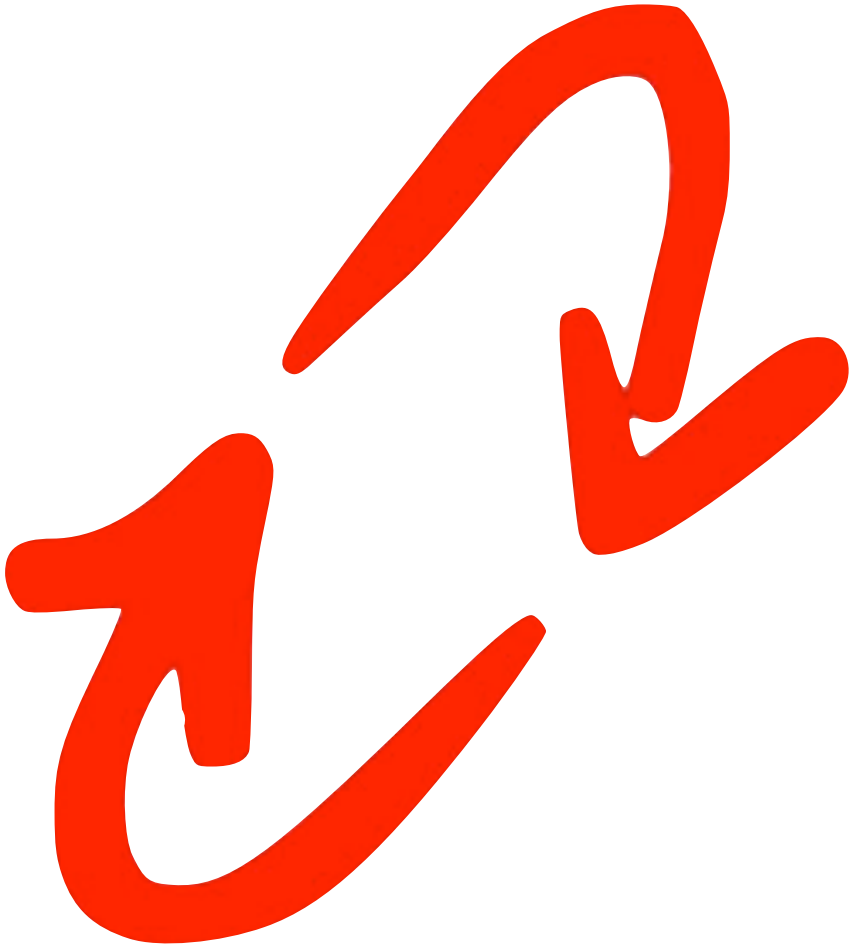


backgrounDrb

http://backgroundrb.rubyforge.org/



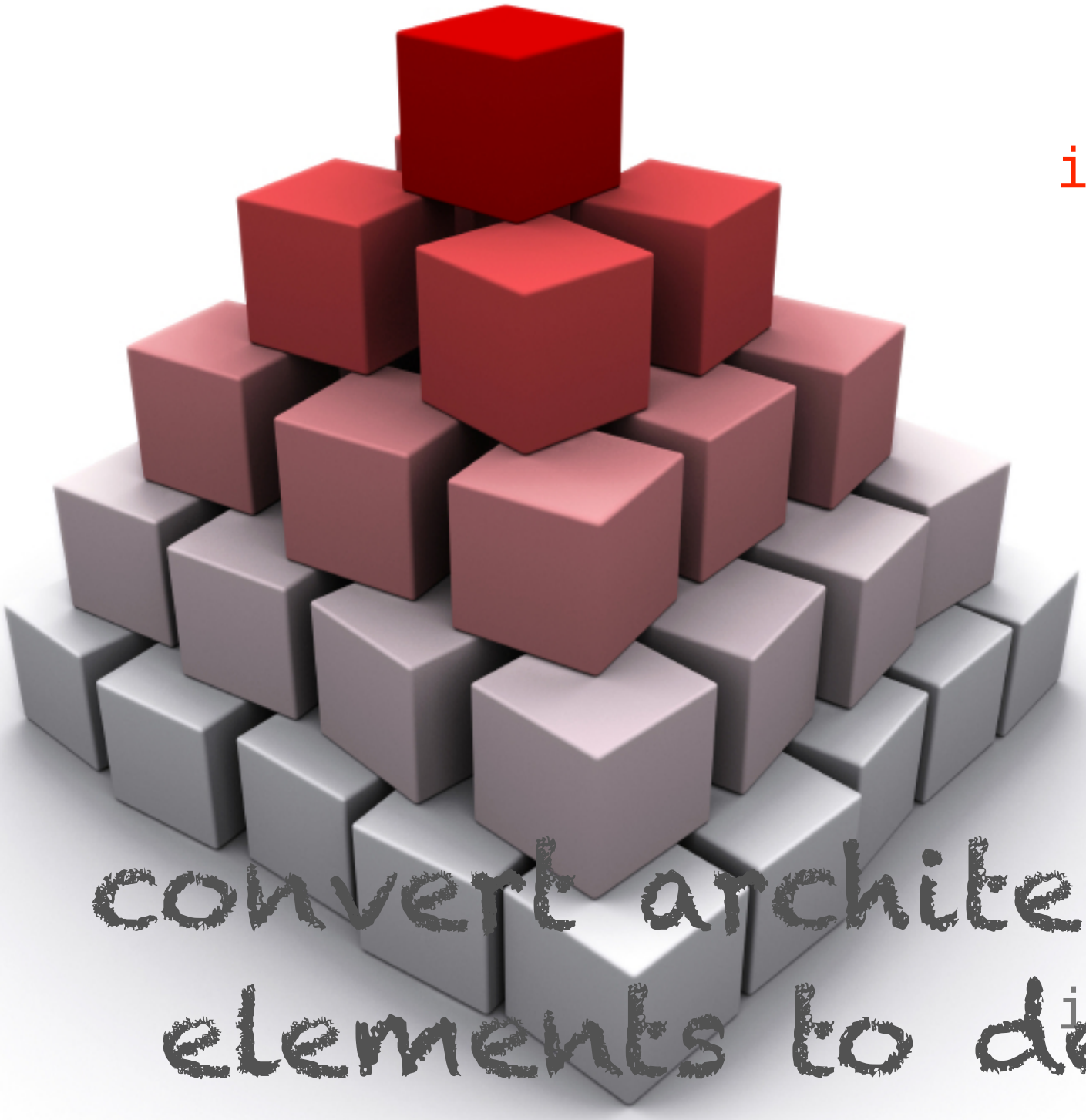
progress bars



continually run

timed events


(Starling)
switch to a real
messaging queue



messaging
infrastructure

convert architectural elements to design

messaging
infrastructure


evolution
strategies

move from shared resources to bounded context

use branch by abstraction/strangler pattern

identify domain boundary

decouple infrastructure

isolate behind a component

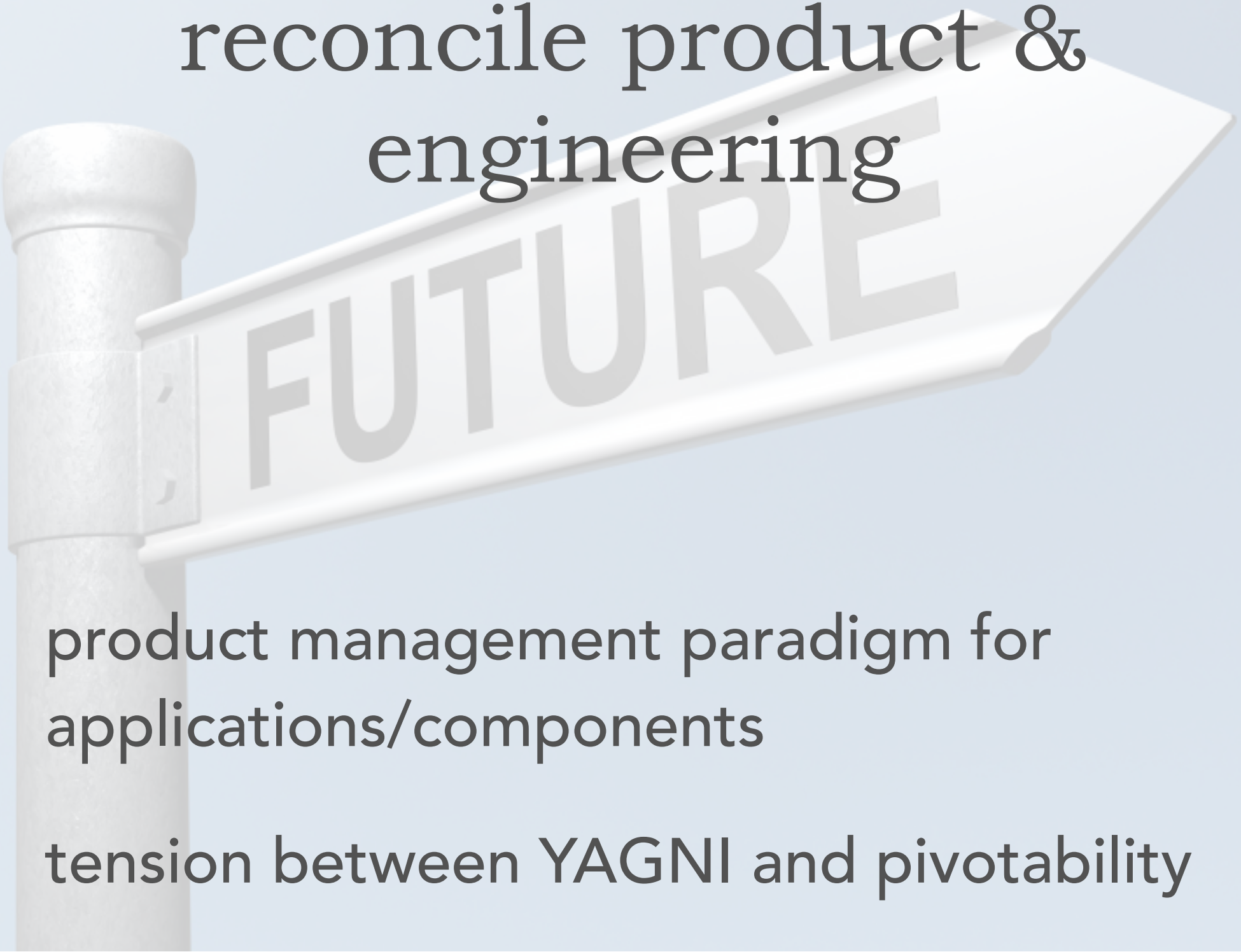
untangle dependencies

beware tools that offer too much help

monitor afferent/efferent coupling

put automated safeguards in place (tools/tests)

reconcile product & engineering

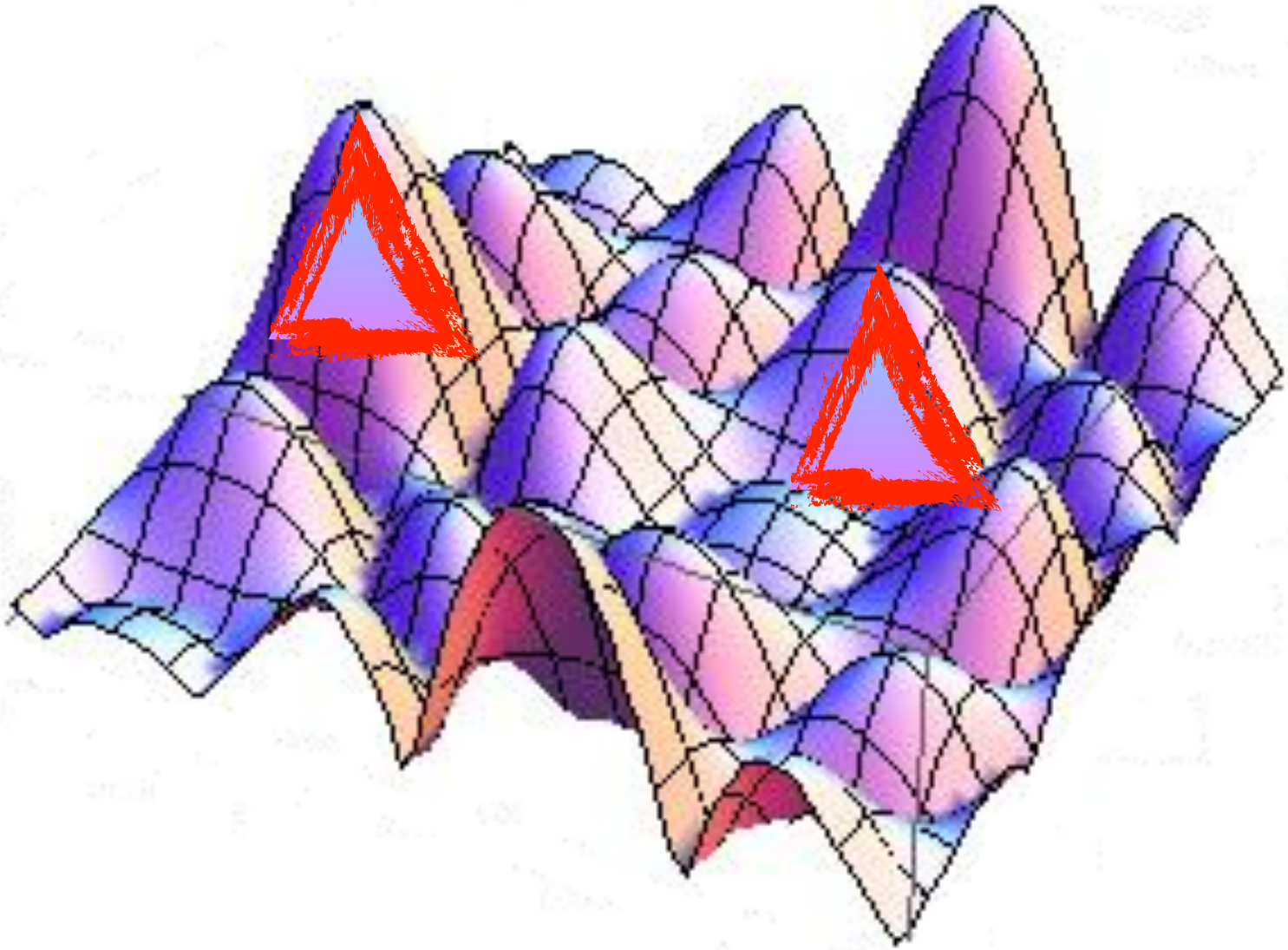


product management paradigm for applications/components

tension between YAGNI and pivotability

greenfield

pivot friendly architecture



design

Editor: Martin Fowler ■ ThoughtWorks ■ fowler@acm.org

Who Needs an Architect?

Martin Fowler

Wandering down our corridor a while ago, I saw my colleague Dave Rice in a particularly grumpy mood. My brief question caused a violent statement, "We shouldn't interview anyone who has 'architect' on his resume." At first blush, this was an odd turn of phrase, because we usually introduce Dave as one of our leading architects.

The reason for his title schizophrenia is the fact that, even by our industry's standards, "architect" and "architecture" are terribly overloaded words. For many, the term "software architect" fits perfectly with the smug controlling image at the end of *Matrix Reloaded*. Yet even in firms that have the greatest contempt for that image, there's a vital role for the technical leadership that an architect such as Dave plays.

What is architecture?

When I was fretting over the title for *Patterns of Enterprise Application Architecture* (Addison-Wesley, 2002), everyone who reviewed it agreed that "architecture" belonged in the title. Yet we all felt uncomfortable defining the word. Because it was my book, I felt compelled to take a stab at defining it.

My first move was to avoid fuzziness by just letting my cynicism hang right out. In a sense, I define *architecture* as a word we use when we want to talk about design but want to puff it up to make it sound important. (Yes, you can imagine a similar phenomenon for architect.) However, as so often occurs, inside the blighted cynicism is a pinch of truth. Understanding came to me after reading a posting from Ralph Johnson on the Extreme Programming mailing list. It's so good I'll quote it all.

A previous posting said

> The RUP, working off the IEEE definition, defines architecture as "the highest level concept of a system in its environment. The architecture of a software system (at a given point in time) is its organization or structure of significant components interacting through interfaces, those components being composed of successively smaller components and interfaces."

Johnson responded:

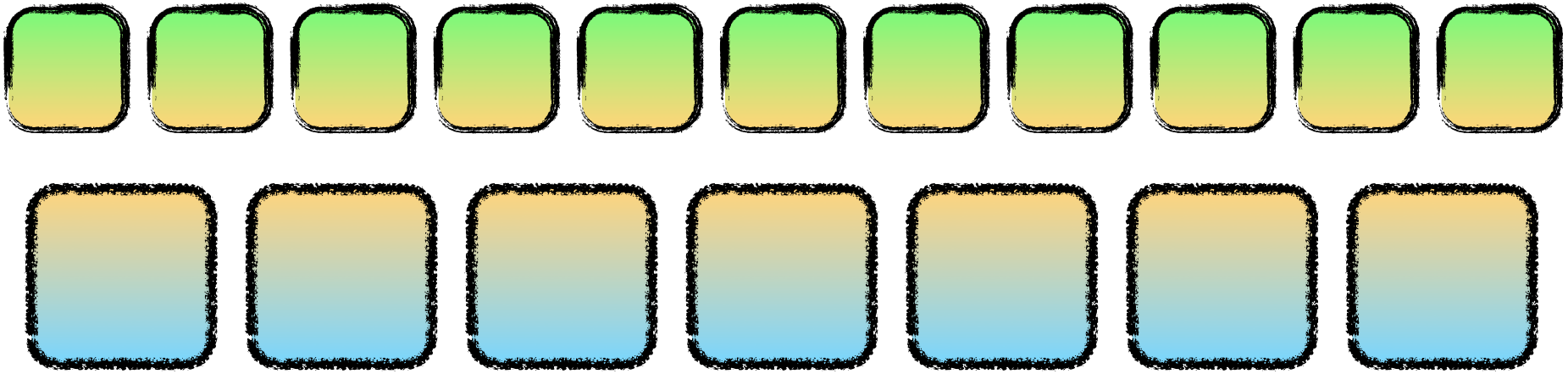
> I was a reviewer on the IEEE standard that used that, and I argued uselessly that this was clearly a completely bogus definition. There is no highest level concept of a system. Customers have a different concept than developers. Customers do not care at all about the structure of significant components. So, perhaps an architecture is the highest level concept that developers have of a system in its environment. Let's forget the developers who just understand their little piece. Architecture is the highest level concept of the expert developers. What makes a component significant? It is significant because the expert developers say so.
>
> So, a better definition would be "In most successful software projects, the expert developers working on that project have a shared understanding of the

0740-7459/03/$17.00 © 2003 IEEE

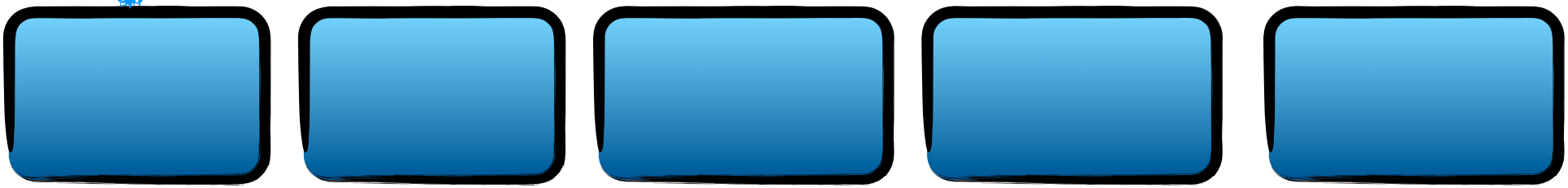
"...things that people perceive as hard to change."

http://martinfowler.com/ieeeSoftware/whoNeedsArchitect.pdf

features



dynamic architectural elements



1. hard to change later

2. as few as possible

architecture

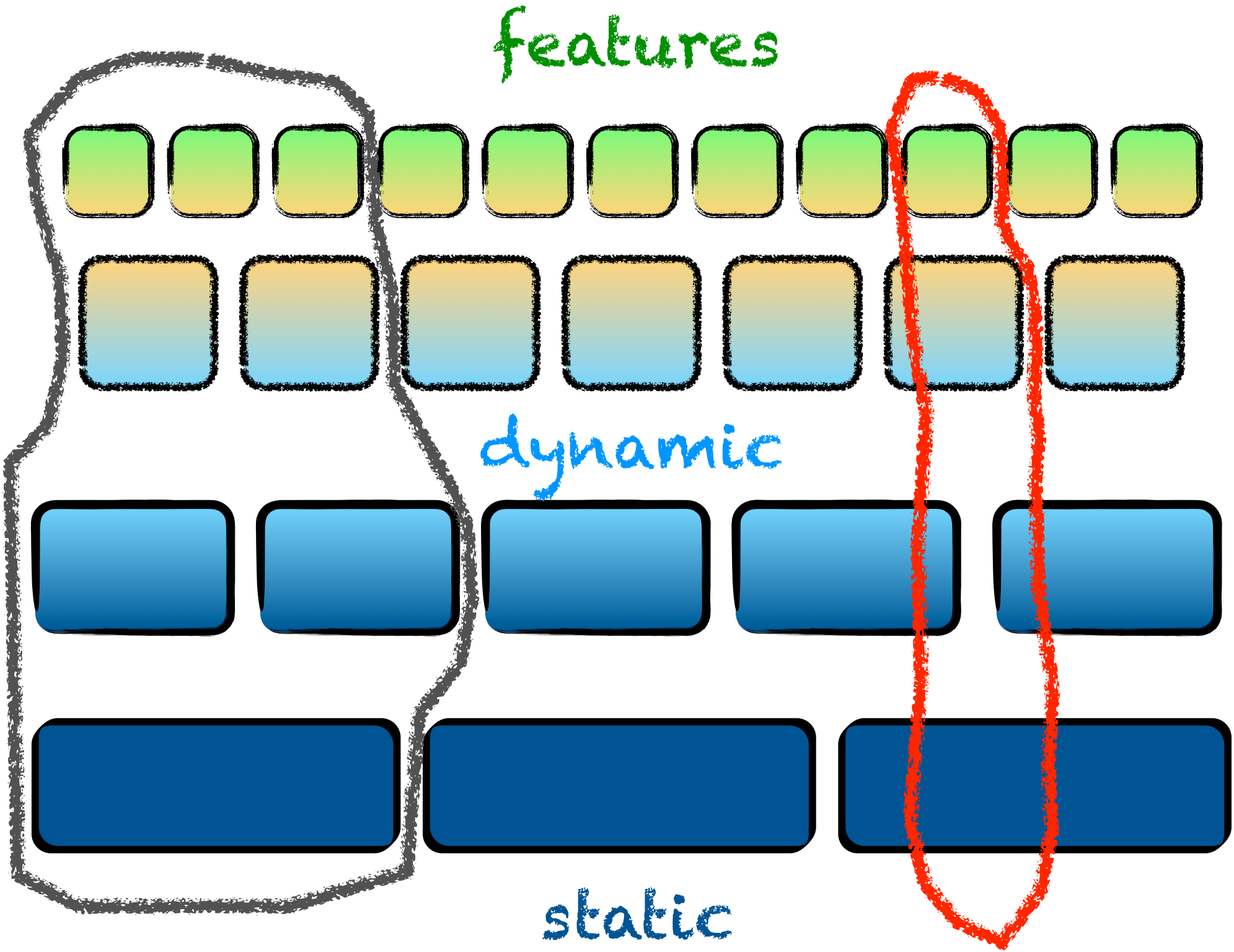
product concerns

feature

dynamic

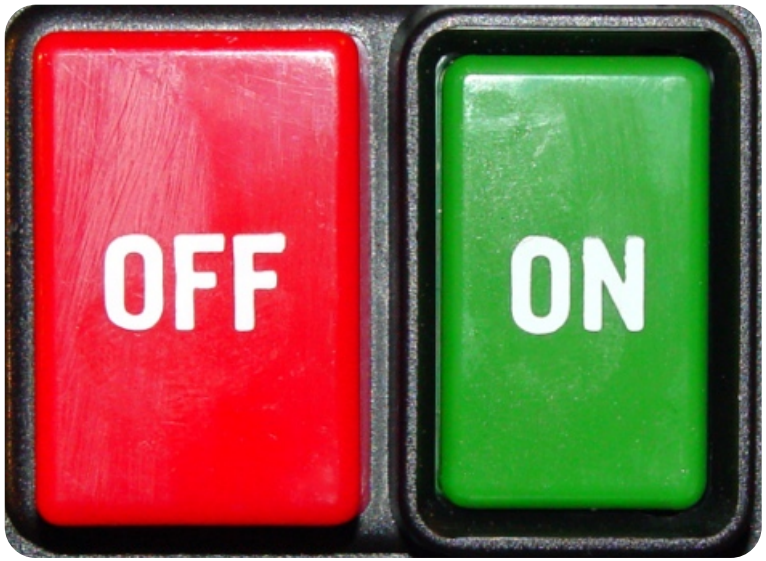
technical architecture

static

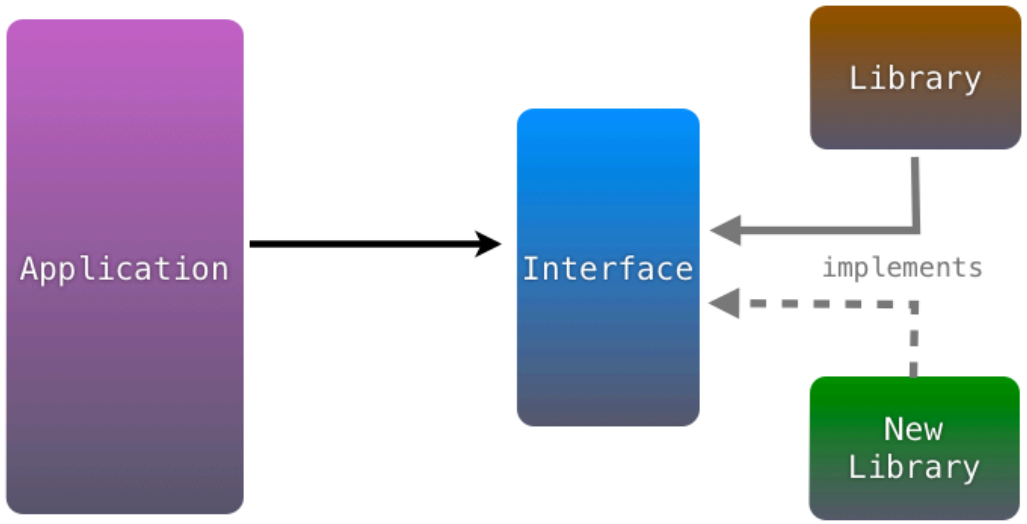

features
dynamic
static

what if you didn't guess correctly ?

feature toggles



branch by abstraction



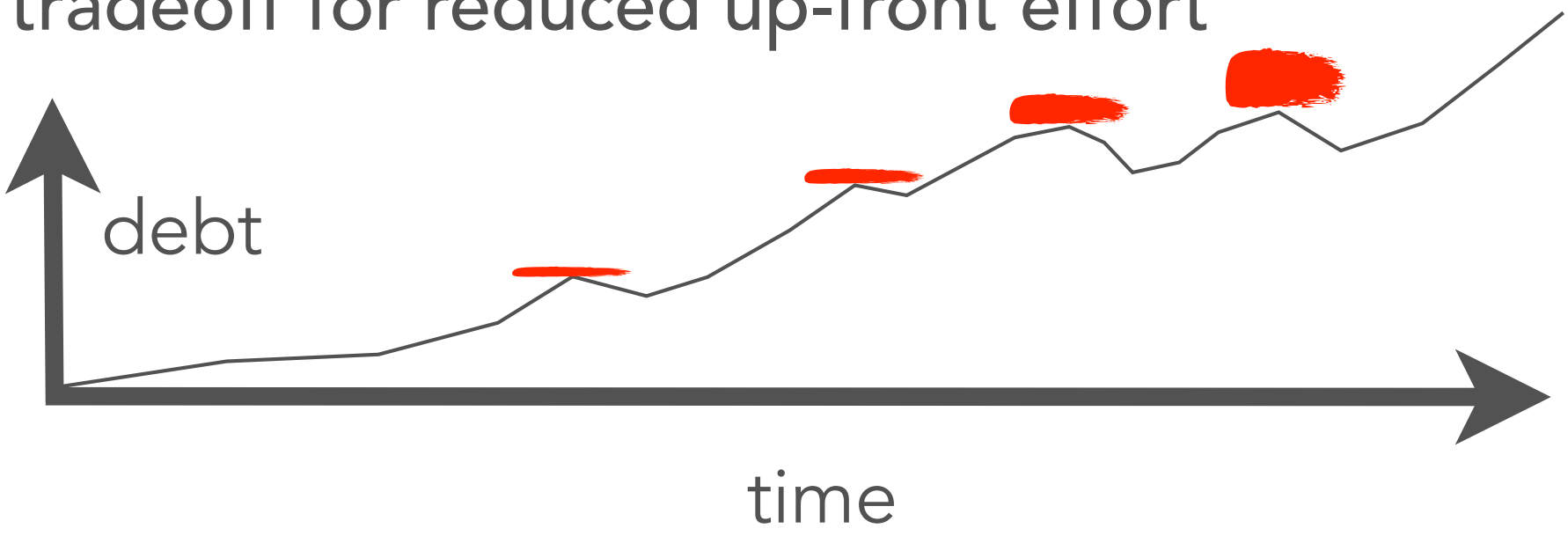
brownfield

technical debt & effort

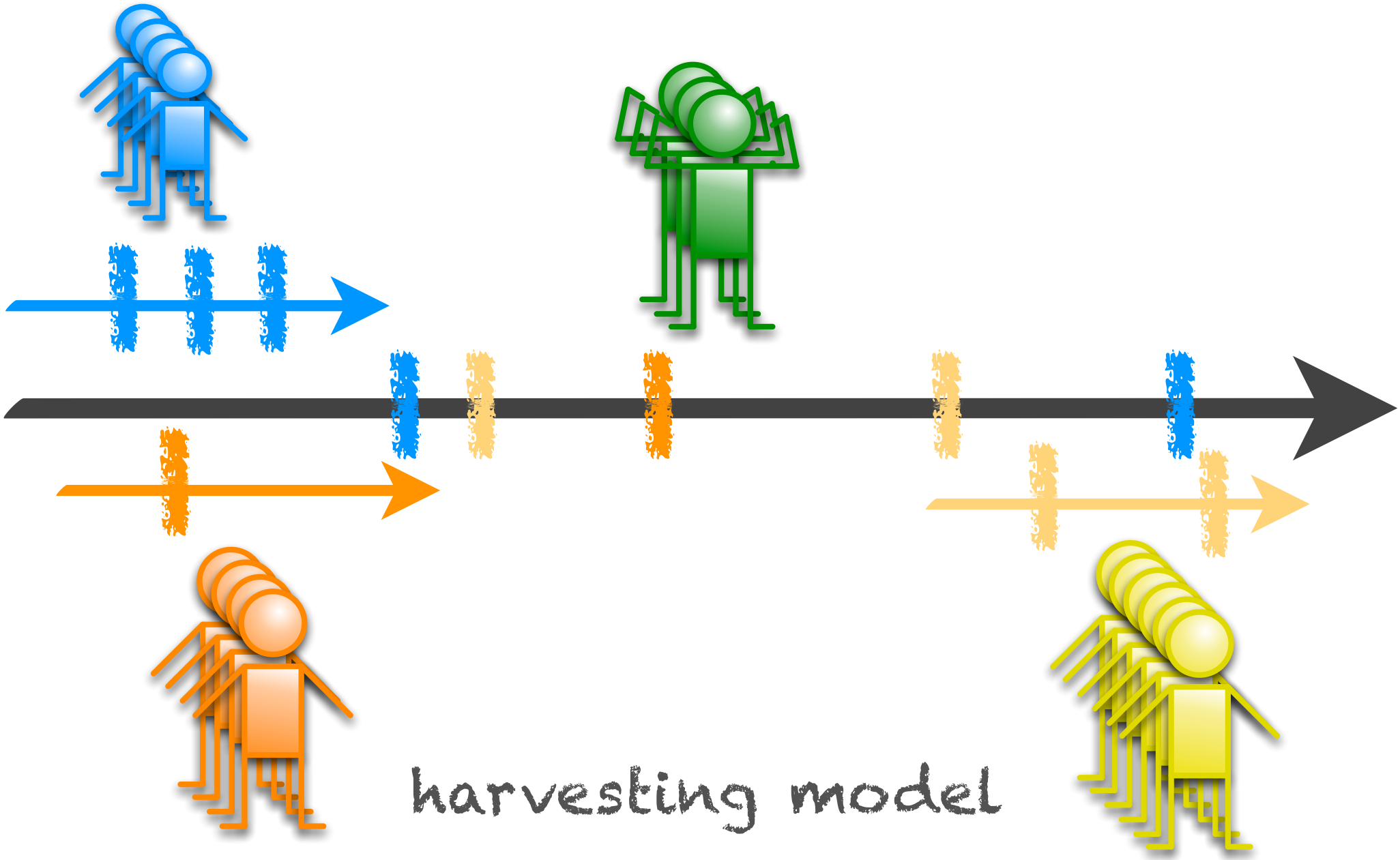
refactoring and restructuring exercises require increasing effort for the same result

plan escalating effort towards remedial architecture & design

tradeoff for reduced up-front effort



“we don’t have time…”



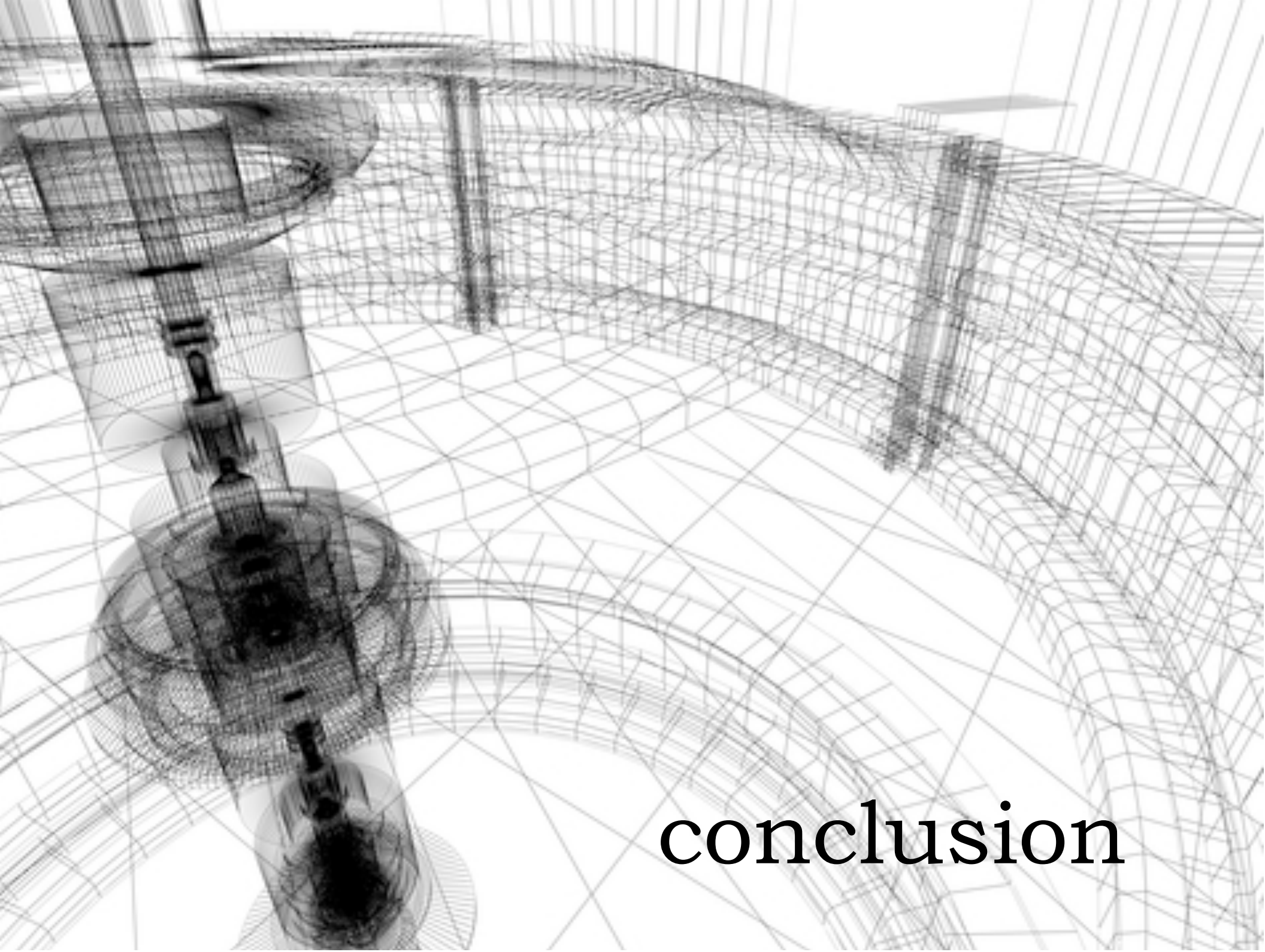
annealing

anneal, n. - heat (metal or glass) and allow to cool slowly, **in order to remove internal stresses and toughen it**

greenfield projects => emphasize malleability

brownfield projects => maximize annealing efforts

this is the price you pay for emergent design!



conclusion

architect for change

Yesterday's best practice is tomorrow's anti-pattern.

Evolutionary architecture and emergent design are inevitable because of the nature of business software.

embrace goal-oriented development

fitness functions for quality of service (non-functional requirements)

use the scientific method

learn & use metrics

holistic approach

architecture doesn’t exist in isolation

synergistic engineering practices

reduce friction

enhance collaboration

deliver continuously

?’S

Mark Richards
Independent Consultant
Hands-on Enterprise / Integration Architect
Published Author / Conference Speaker

http://www.wmrichards.com
http://www.linkedin.com/pub/mark-richards/0/121/5b9

Published Books:
Java Message Service, 2nd Edition
97 Things Every Software Architect Should Know
Java Transaction Design Strategies

Neal Ford
Director / Software Architect /
Meme Wrangler

ThoughtWorks®

2002 Summit Blvd, Level 3, Atlanta, GA 30319, USA
T: +1 40 4242 9929 Twitter: @neal4d
E: nford@thoughtworks.com W: thoughtworks.com