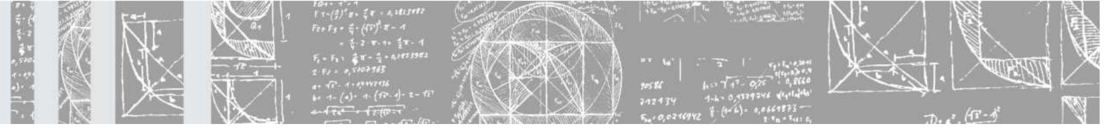
HELLO2MORROW



Golden Rules to Improve Your Architecture

Alexander v. Zitzewitz

hello2morrow Inc.

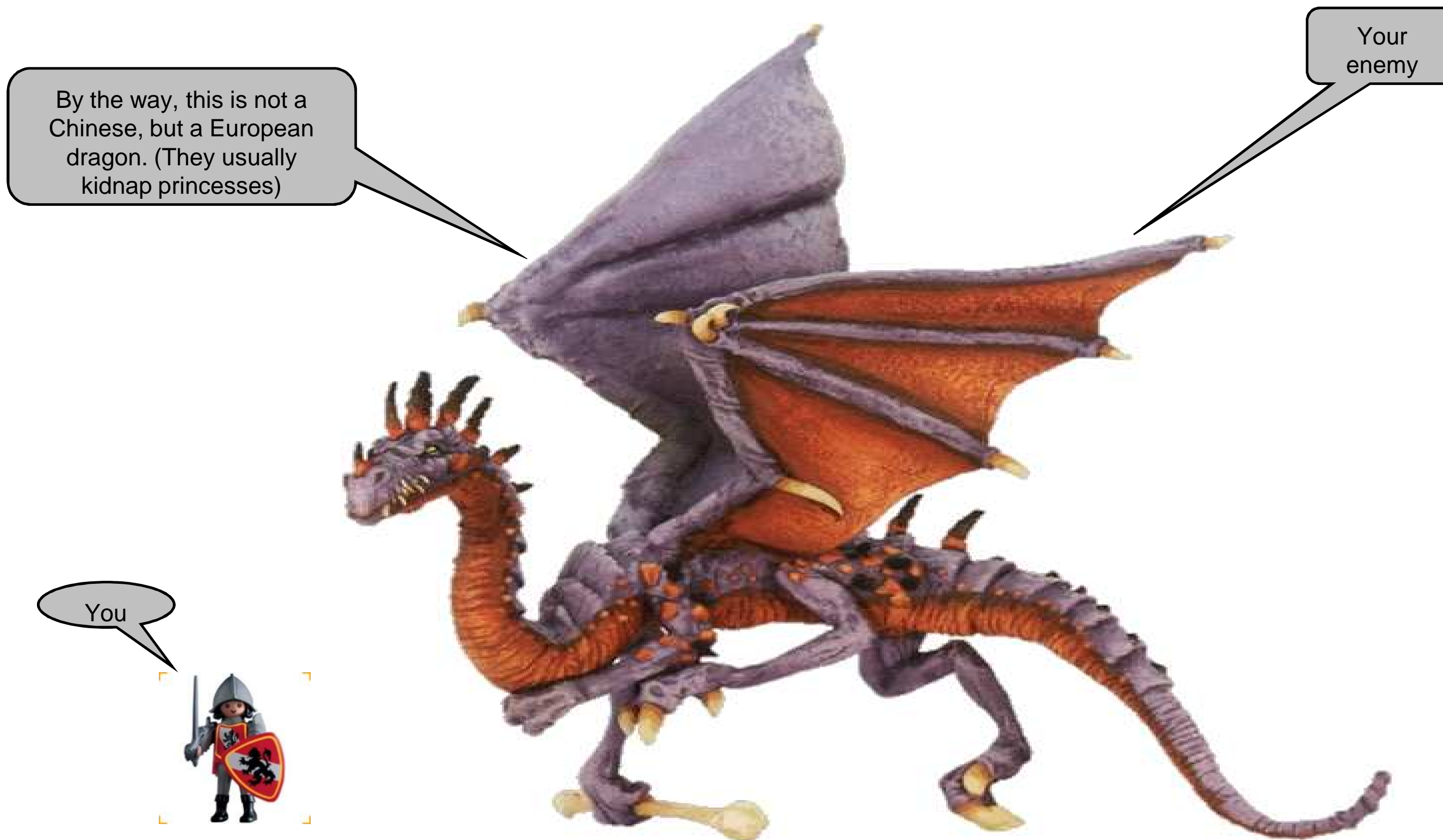
© 2005-2008, hello2morrow

Do you manage your architecture and technical quality?

- Controlling the architecture and technical quality of software is difficult
- Achieving a good structure and a high level of quality is obviously quite difficult (otherwise far more projects would be a success)

© 2005-2008, hello2morrow

Your most frightening enemy: the dragon of complexity



© 2005-2008, hello2morrow

Erosion of architecture – a fundamental law?

- Architecture erosion is quite a known problem
  - System knowledge and skills are not evenly distributed
  - Complexity grows faster than system size
  - Unwanted dependencies are created without being noticed
  - Coupling and complexity are growing quickly. When you realize it, it is often too late
  - Management considers software as a black box
  - Time pressure is always a good excuse to sacrifice structure
- Typical symptoms of an eroded architecture are a high degree of coupling and a lot of cyclic dependencies
  - Changes become increasingly difficult
  - Testing and code comprehension also become increasingly difficult
  - Deployment problems of all kind

© 2005-2008, hello2morrow

Counter measures

- Avoid package cycles by using jdepend
  - Better than nothing
  - But you cannot automatically check the structure of your code
  - Level of abstraction is far to low
- Code Reviews
  - Hopeless and inefficient, at least for controlling the architecture
  - But - do team code reviews for mutual code comprehension
- CheckStyle and FindBugs
  - Give little or no help on the structural level
  - But efficient to replace manual reviews
- Check some key metrics
  - They help you to be aware of certain problems early enough
- Micro projects
  - The smaller your sub-project, the more you loose flexibility

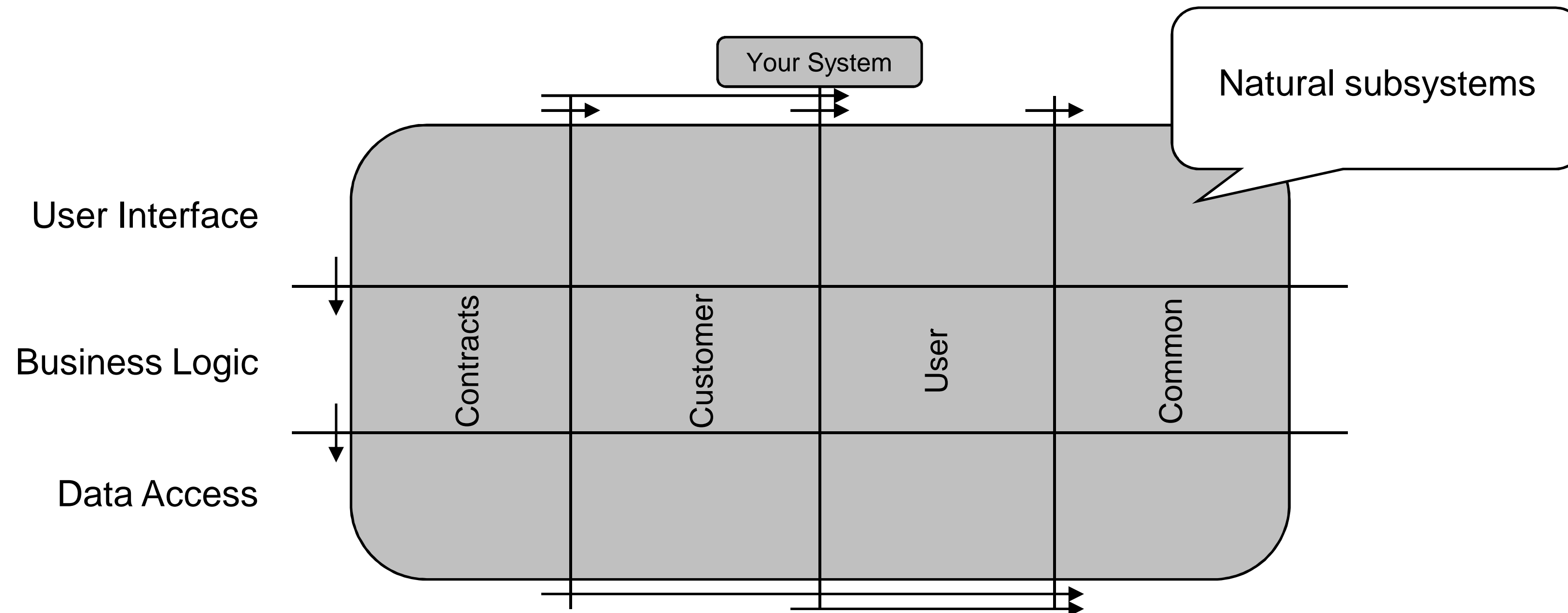
© 2005-2008, hello2morrow

What you need

- The ability to describe your architecture and some rules for technical quality on a high level of abstraction and then automatically check your code for compliance
- A small set of relevant key metrics to keep your technical quality under control
- Architecture patterns to keep the coupling low and the structure simple
  - Dependency inversion principle
  - Use the Spring Framework (dependency injection).
- A tool that allows every developer to check for himself, if his code changes would cause an architecture violation

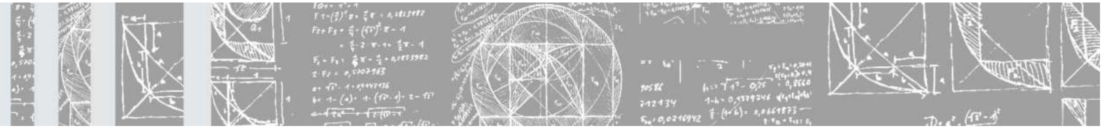
© 2005-2008, hello2morrow

Definition of a Logical Architecture



- Step 1: Cut horizontally into Layers
- Step 2: Cut vertically into vertical slices by functional aspects
- Step 3: Defines the rules of engagement

© 2005-2008, hello2morrow



Meta model: layers, vertical slices and subsystems

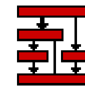
- Each subsystem belongs to exactly one layer
- A subsystem also might belong to a vertical slice
- The association between vertical slices and subsystems is typically implemented by a naming convention
- Vertical slices do not have to be present on every layer
- Technical subsystems typically are not associated with any vertical slice
- Technical systems often do not have vertical slices at all

HELLO2MORROW

© 2005-2008, hello2morrow

Mapping of physical elements to logical elements

- Each package is mapped to exactly one subsystem
- If package‘s contain types of several subsystems, virtual refactorings are helpful
- A good naming convention for package‘s can make your life very simple
  - E.g.: com.hello2morrow.project.verticalslice.layer…
- Subsystems should have interfaces
- Work incrementally
  - Start with your layering
  - Then add the vertical slices (if applicable)
  - Define subsystem interfaces
  - Fine tune the rules of engagement on the subsystem level



© 2005-2008, hello2morrow

A good architecture is a flexible architecture

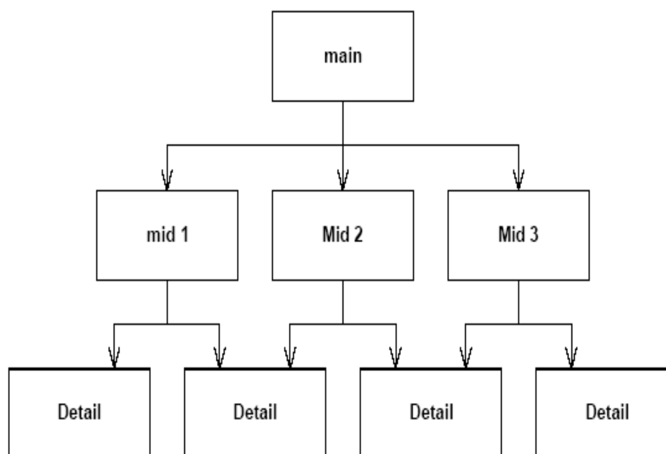
- You are always shooting on moving targets
- To gain flexibility and potential re-usability you have to minimize coupling
  - Always avoid cyclic dependencies
  - Your systems flexibility is inverse proportional to its coupling

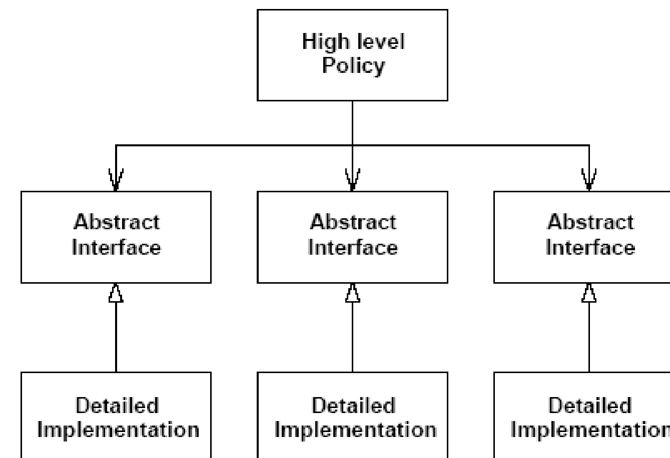
HELLO2MORROW

© 2005-2008, hello2morrow

How to keep the coupling low?

- Dependency Inversion Principle (Robert C. Martin)
  - Build on abstractions, not on implementations
  - Best pattern for a flexible architecture with low coupling
  - Have a look at dependency injection frameworks (e.g. Spring)



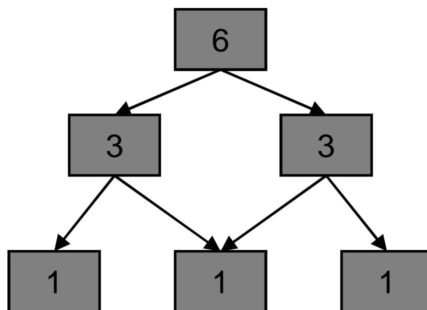


HELLO2MORROW

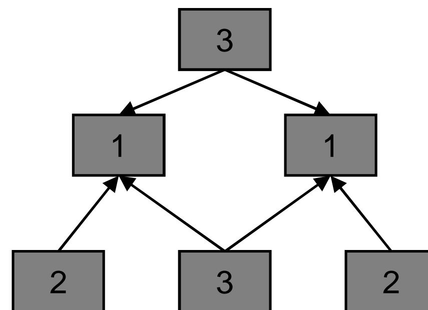
© 2005-2008, hello2morrow

How to measure coupling

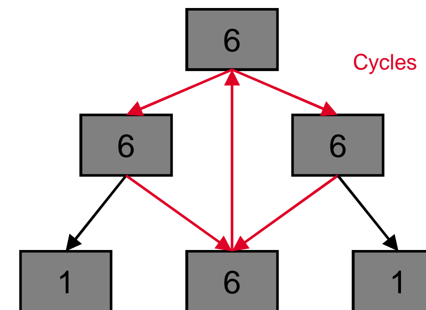
- ACD = Average Component Dependency
- Average number of direct and indirect dependencies
- rACD = ACD / number of elements
- NCCD: normalized cumulated component dependency



CCD = 15
ACD = 15/6 = 2,5



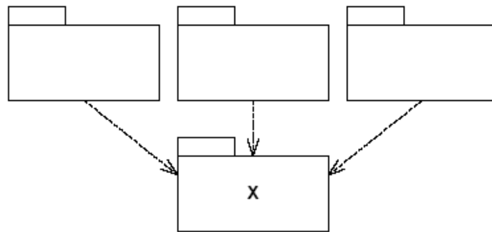
Dependency Inversion
ACD = 12/6 = 2



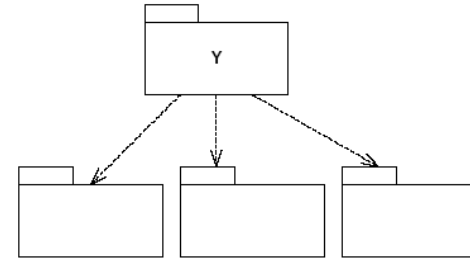
ACD = 26/6 = 4,33

© 2005-2008, hello2morrow

Architecture metrics of Robert C. Martin



X is „stable“



Y is „instable“

D_i = Number of incoming dependencies

D_o = Number of outgoing dependencies

Instability $I = D_o / (D_i+D_o)$

Build on abstractions, not on implementations

HELLO2MORROW

 © 2005-2008, hello2morrow

Abstractness (Robert C. Martin)

N_c = Total number of types in a type container

N_a = Number of abstract classes and interfaces in a type container

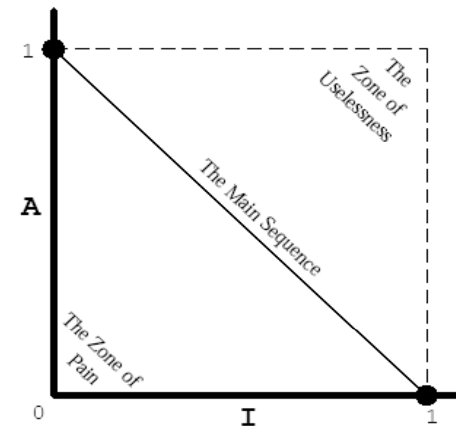
Abstractness A = N_a/N_c

© 2005-2008, hello2morrow

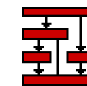
Metric „distance“ (Robert C. Martin)

$D = A + I - 1$

Value range [-1 .. +1]



- Negative values are in the „Zone of pain“
- Positive values belong to the „Zone of uselessness“
- Good values are close to zero (e.g. -0,25 to +0,25)
- „Distance“ is quite context sensitive



© 2005-2008, hello2morrow

Cyclic dependencies are evil

- "Guideline: No Cycles between Packages. If a group of packages have cyclic dependency then they may need to be treated as one larger package in terms of a release unit. This is undesirable because releasing larger packages (or package aggregates) increases the likelihood of affecting something." [AUP]
- "The dependencies between packages must not form cycles." [ASD]
- "Cyclic physical dependencies among components inhibit understanding, testing and reuse. Every directed a-cyclic graph can be assigned unique level numbers; a graph with cycles cannot. A physical dependency graph that can be assigned unique level numbers is said to be levelizable. In most real-world situations, large designs must be levelizable if they are to be tested effectively. Independent testing reduces part of the risk associated with software integration " [LSD]

HELLO2MORROW

Example : Cyclic Dependency

Presentation

Main

AlarmHandler

AlarmToConsole

AlarmToFile

Model

AlarmClock

HELLO2MORROW

Breaking the Cycle

Presentation

Main

<<abstract>>
AlarmHandler

AlarmToConsole

AlarmToFile

<<bottom>>
Model

AlarmClock

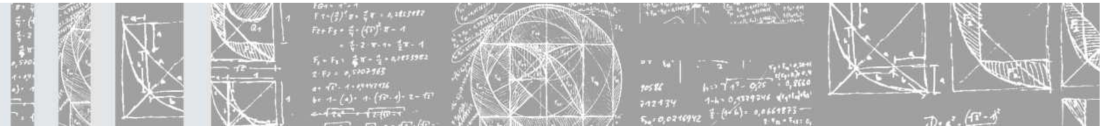
<<interface>>
IAlarmHandler

HELLO2MORROW

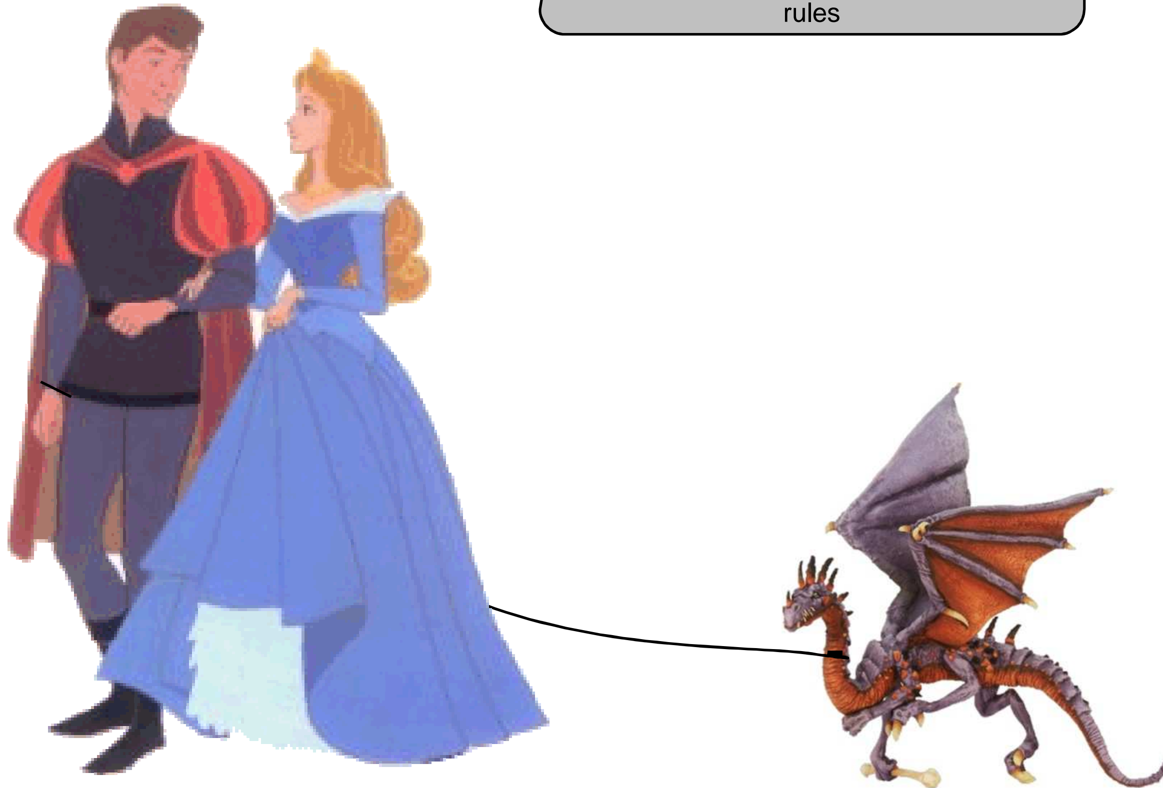
Six golden rules for a successful project

- Rule 1:
  Define a cycle free logical architecture down to the level of subsystems and a strict and consistent package naming convention
- Rule 2:
  Do not allow cyclic dependencies between different packages
- Rule 3:
  Keep the relative ACD low (< 7% for 500 compilation units, NCCD < 6)
- Rule 4:
  Limit the size of Java files (700 LOC is a reasonable value)
- Rule 5:
  Limit the cyclomatic complexity of methods (e.g. 15)
- Rule 6:
  Limit the size of a Java package (e.g. less than 50 types)

© 2005-2008, hello2morrow



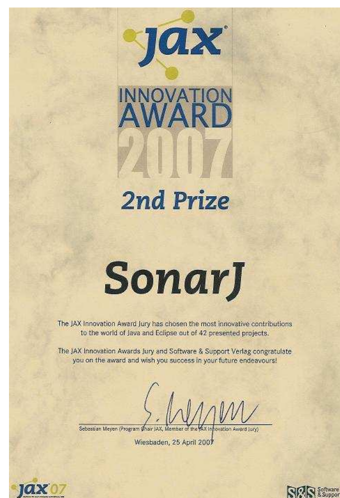
You have won the heart of the princess after your were able to calm the dragon of complexity by following the five golden rules



© 2005-2008, hello2morrow

Awards and nominations

- Second prize of Jax innovation award in April 2007
- Nomination for European ICT prize 2007
- Awarded as most exciting innovation on Systems 2005


jax

INNOVATION AWARD 2007

2nd Prize

SonarJ

The JAX Innovation Award Jury has chosen the most innovative contributions to the world of Java and Eclipse out of 42 presented projects.

The JAX Innovation Awards Jury and Software & Support Verlag congratulate you on the award and wish you success in your future endeavours!

Sebastian Meyen (Program Chair JAX, Member of the JAX Innovation Award Jury)

Wiesbaden, 25 April 2007

jax 07

S&S Software & Support Verlag

Innovation Area Award

SYSTEMS

24-28 Oktober 2005

Innovation Area Award SYSTEMS 2005

URKUNDE

für

die spannendste Innovation

der Firma

hello2morrow GmbH

SYSTEMS, 25. Oktober 2005

Die Jury:
Andreas Dohmen, Cisco Systems, **Hans Joachim Heins**, Consellex, **Andreas Itzrodt**, Ernst & Young, **Andreas Hedler**, Comdrade AG, **Werner Knoblich**, Red Hat, **Dr. Martina Maier**, eXXcellent solutions, **Klaus C. Plönzke**, Pluralis AG, **Tom Schwaller**, IBM, **Alfons Stärk**, Microsoft, **Eric Voss**, Sun Microsystems, **Prof. Dr. Heinz Thielmann**, Frauenhofer Institut SIT

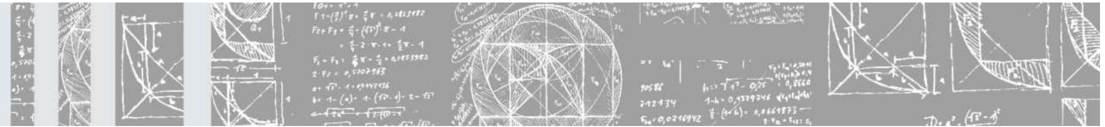
IT-ADVENTURE

© 2005-2008, hello2morrow

Some of our Customers and Partners

- Accenture
- BearingPoint
- Electronic Arts
- Commerzbank
- PlanetHome
- Teambank
- T-Systems
- IBM Global Services
- eBay Motors
- Société Générale AM
- Gemalto
- Farm Credit Canada
- University of Michigan
- Sanofi-Aventis
- Austrian National Bank
- French Army
- Credit Suisse
- Daimler
- Austrian Environment Agency
- Unilog
- Géndarmerie Nationale (France)
- Volkswagen
- Austrian Government
- Siemens
- Business Objects
- ImmobilienScout 24

© 2005-2008, hello2morrow



Your questions…

- Download free architecture rules white paper from:

  www.hello2morrow.com

- My email address:

  a.zitzewitz@hello2morrow.com

HELLO2MORROW

© 2005-2008, hello2morrow