



Quasar Software Categories or: How to Find Components

Johannes Siedersleben

TU Dresden, May 4, 2005

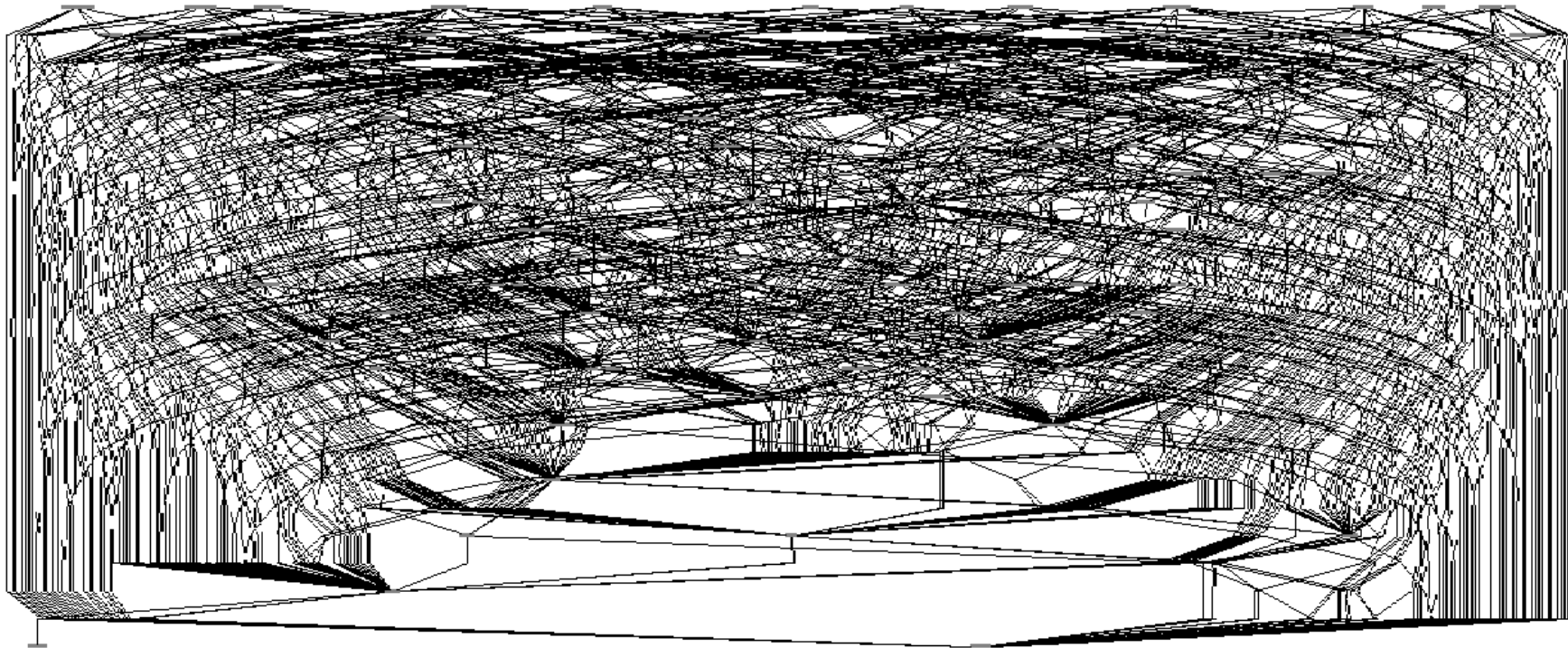
sd&m

The Zen of Quasar



- Are your lights on?
Don't shift responsibilities. Trust your team.
- Simple messages.
- Keep it simple and stupid. You are done when there is nothing left to remove, and not when there is nothing left to add.
- No design is ultimate, not even mine.
- My diagrams are more beautiful than all others, aren't they?
- Code counts.
- No broken windows.
- A fool with a tool is still a fool.
- Software design is cool.
- Software design is painful.
- Fewer nightmares.

A Nightmare: Taligent Dependency Graph



Quasar: Fewer Nightmares!



- Claim: Quasar reduces the likelihood of nightmares
 - Quasar is neither a panacea, nor a hype. It contains many old ideas (Dave Parnas!) and a few new ones.
 - Quasar means Quality-Software-Architecture
 - Quasar contains:
 1. Concepts and notions: Software categories, interfaces, components
 2. Standard architectures and standard interfaces
 3. Ready-to-use quasar conformant components such as persistence, views, authorization, rule manager.
 - More informations at <http://www.openquasar.de>
<http://www.sdm.de/de/unternehmen/fe/quasar/index.html>
- and a book (English edition planned for 2006):
Moderne Softwarearchitektur –
Umsichtig planen, robust bauen mit Quasar (dPunkt-Verlag, 2004)

Software Categories (Blood Groups)



Software can be ...

- | | |
|----|---|
| 0 | determined by nothing (container, strings)
<i>ideally reusable, of no use on its own</i> |
| A | determined by the application (customer, order, delivery)
<i>this is what the system is all about</i> |
| T | determined by at least one technical API (e.g. file system)
<i>must be</i> |
| AT | determined by the application and at least on technical API
<i>to be avoided or at least separated carefully</i> |
| R | representation software (transformation between A and T; mild kind of AT;
ideal candidate for generators) |

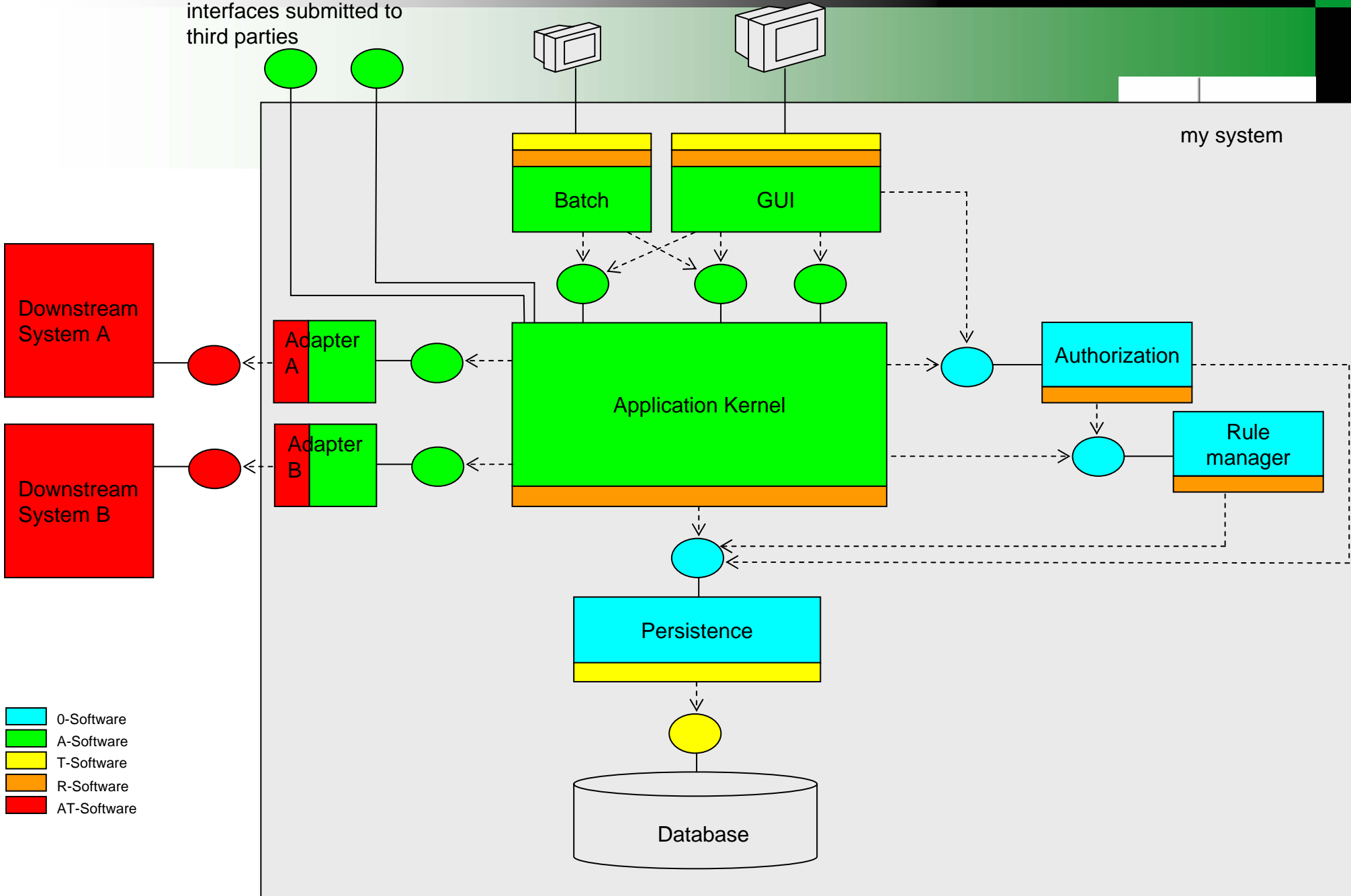
category calculus

$$A + 0 = A$$

$$T + 0 = T$$

$$A + T = AT$$

interfaces submitted to third parties



Interfaces are Abstractions

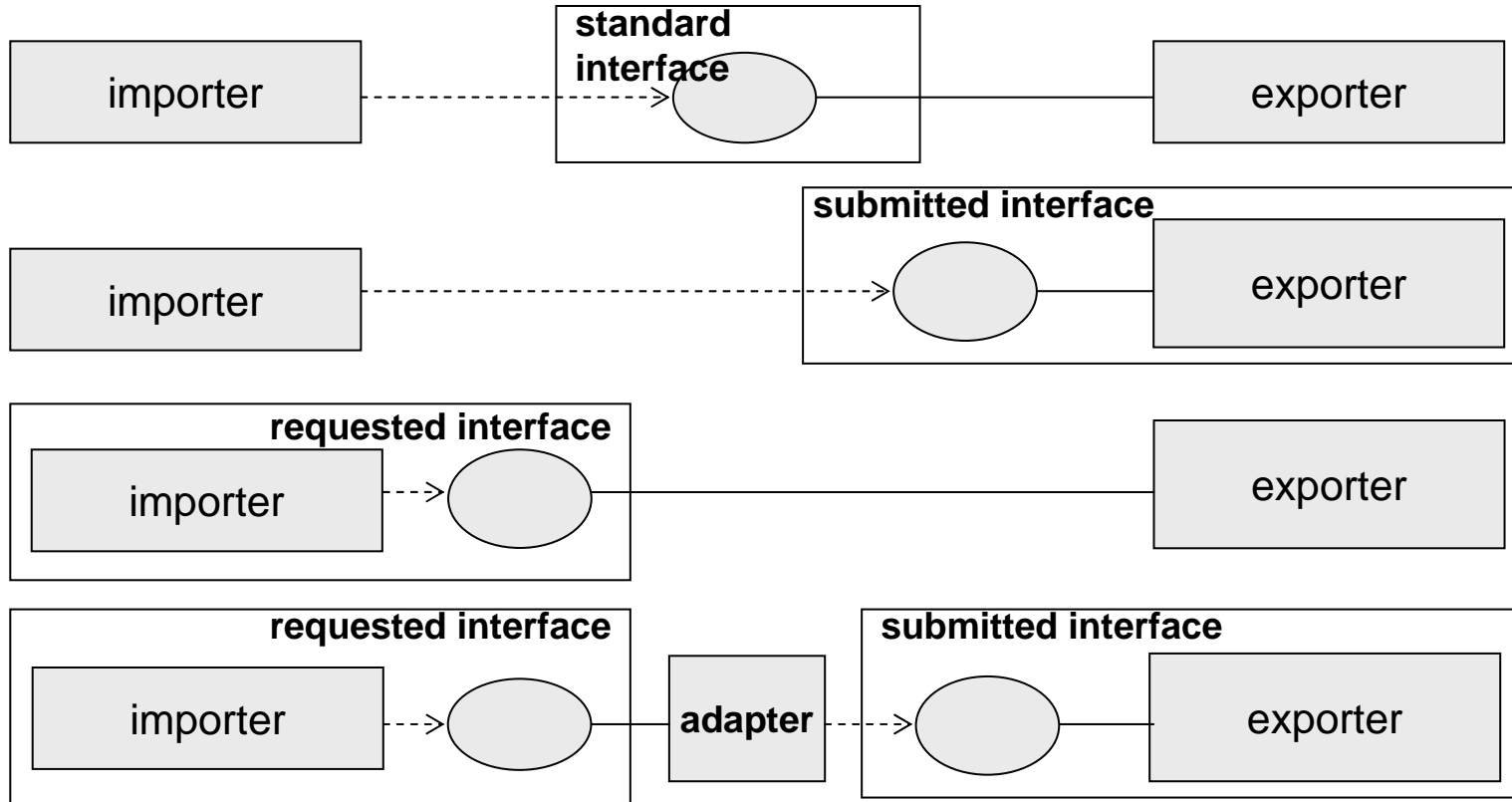


- Car: Steering wheel, accelerator, brake pedal, clutch pedal
- Authorization: *boolean mayPerform(user, action)*
- CMS: *CustomerInfo getInfo(customerNumber)*

How to define interfaces:

1. Find the right level of abstraction
2. Separate concerns: CORBA, WSDL and the like are orthogonal to application interfaces such as the ones above
3. Consider exception handling as a separate concern.

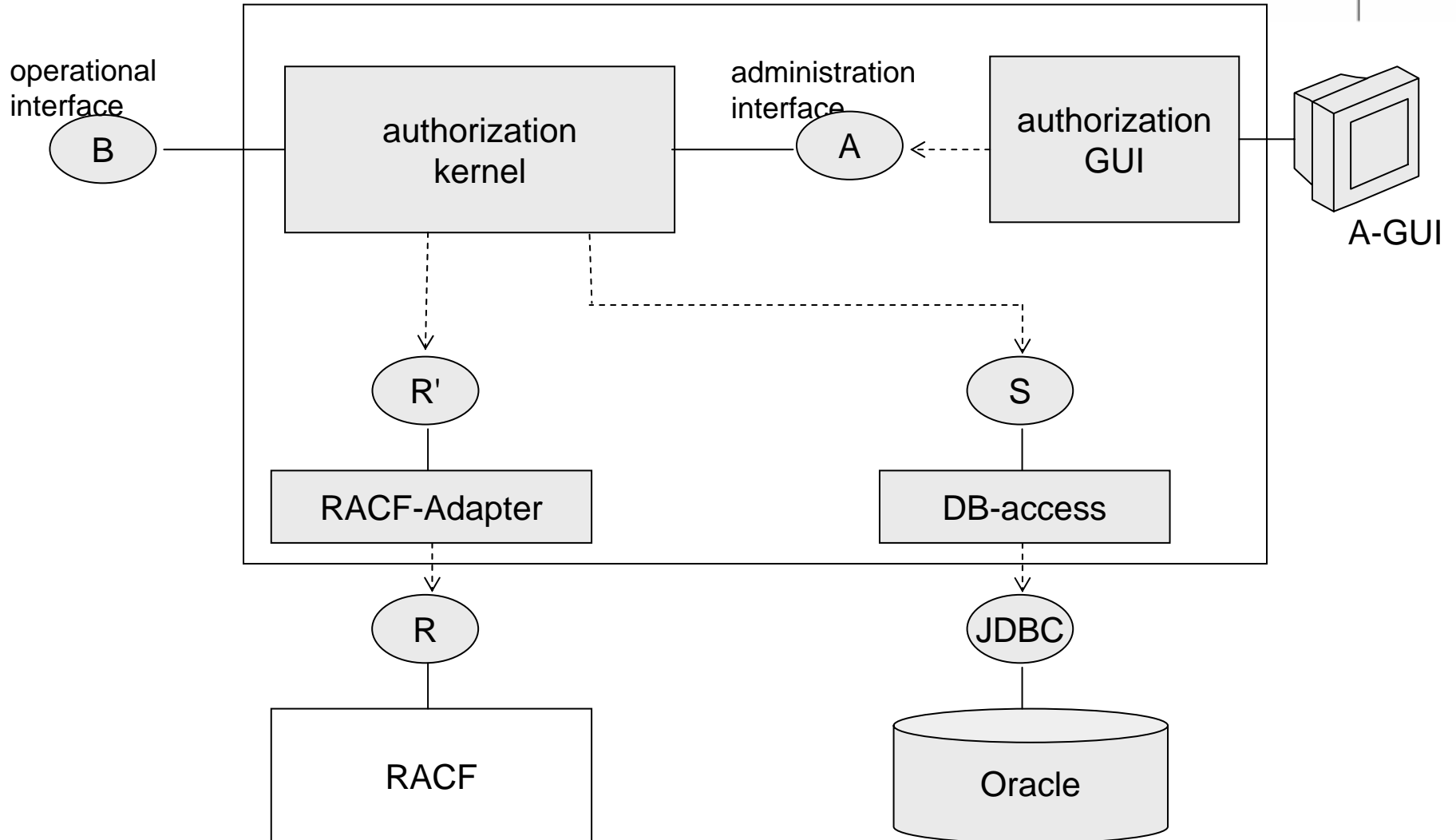
Where Do Interfaces Come From?



Don't get confused:

submitted/requested
 vs.
imported/exported

Authorization by Interfaces



Roles: Who knows what?

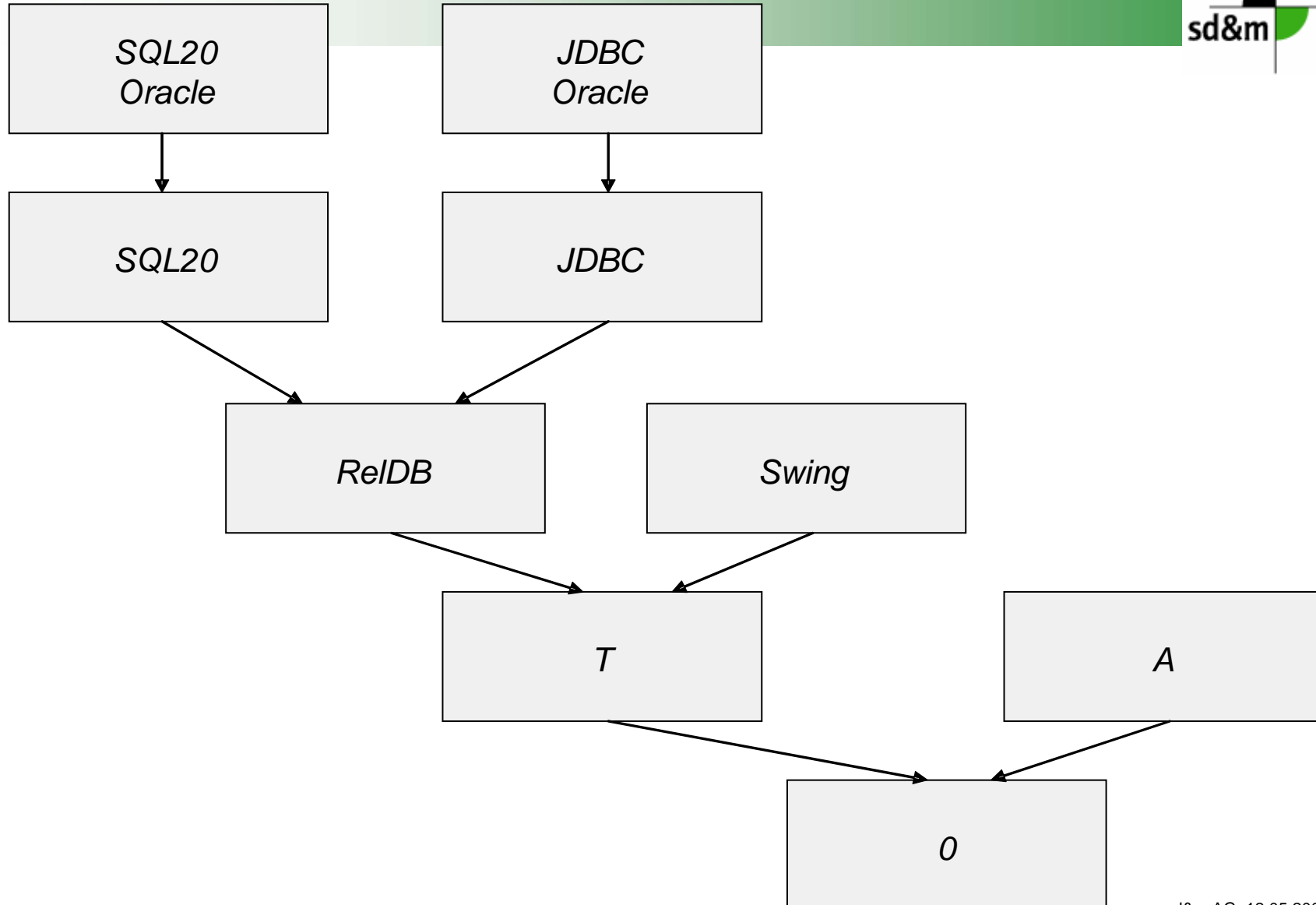


- Application programmers (many): **B**
- Administrators (few, depends on the organization): **A-GUI**
- Authorization experts (2): **A, B, R', S**
- RACF expert (1): **R, R'**
- DB expert (1): **S, JDBC**
- GUI programmer (1): **A, A-GUI**

Rule of thumb

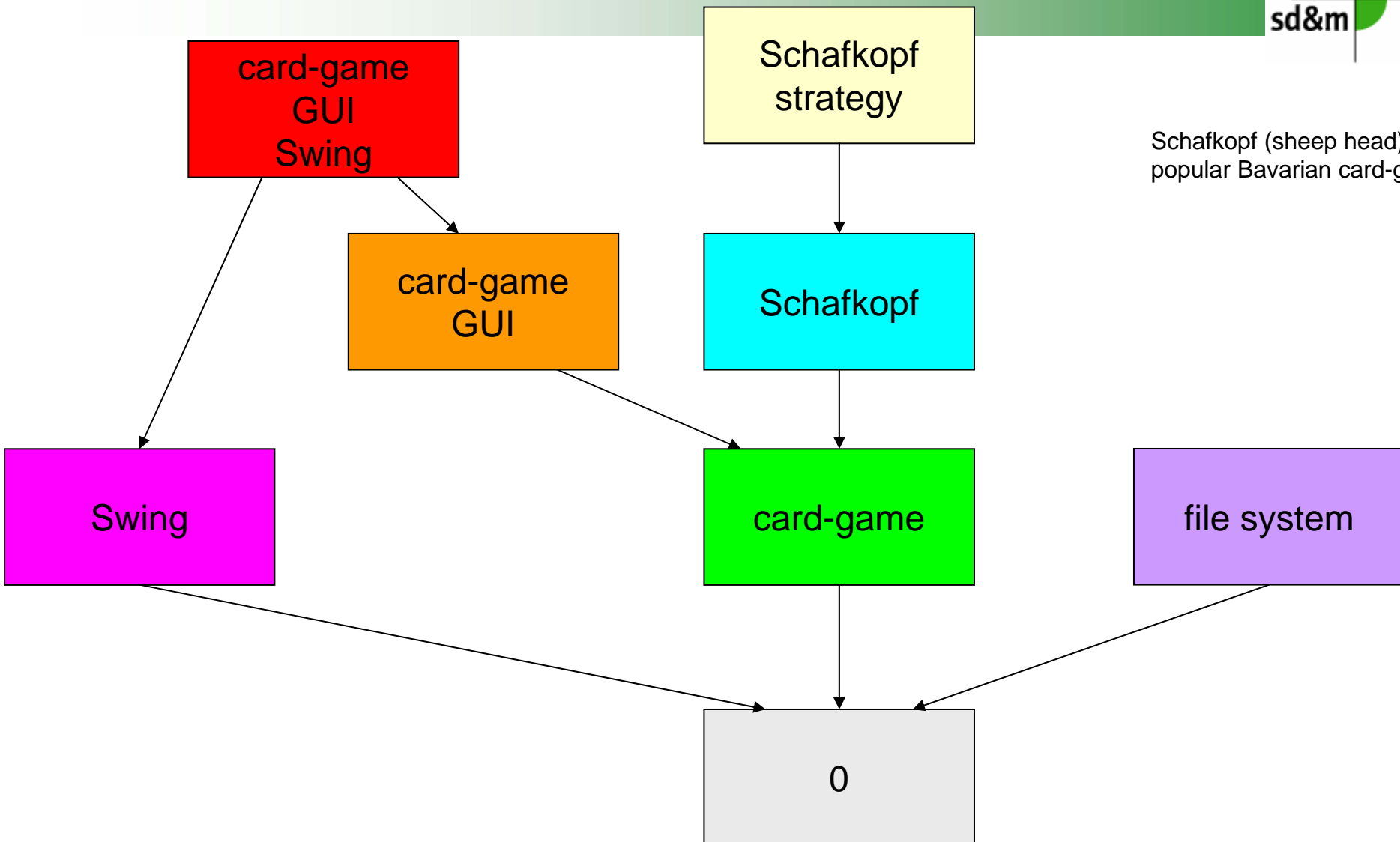
for each interface s: $\text{complexity}(s) * \text{nbrOfUsers}(s) = \text{const}$

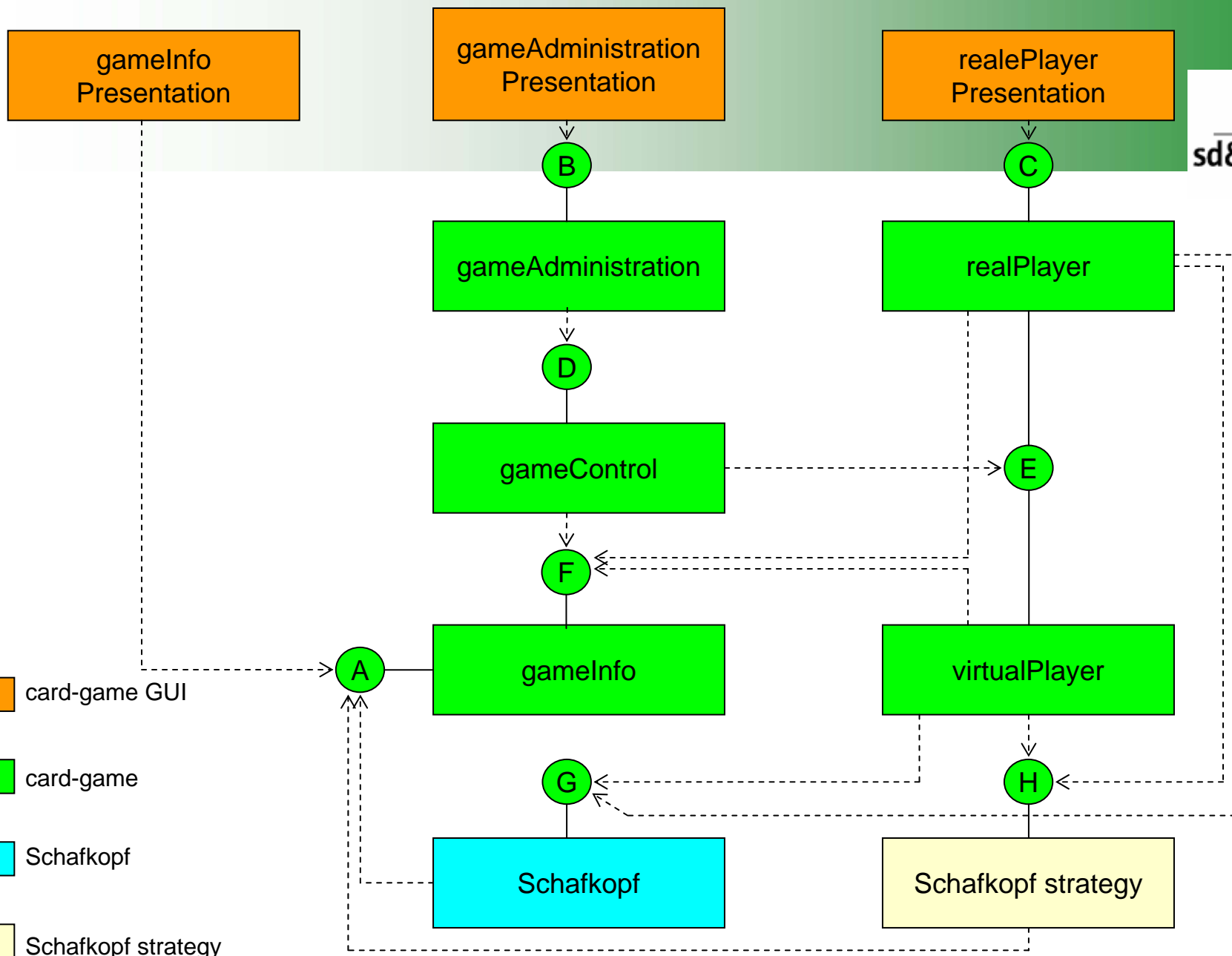
Categories: More Than A and T (1)



Categories: More Than A and T

Schafkopf (sheep head) =
popular Bavarian card-game





- card-game GUI
- card-game
- Schafkopf
- Schafkopf strategy

A Closer Look at Categories



- Category \approx knowledge area.
- Hypothesis: Categories are easy to find, interfaces and components are not. Categories are a guide for finding interfaces and components.
- Any component and any interface belong to exactly one category.
- The architect defines categories as a non-cyclic directed graph.
- The predecessor/successor relation defines " \leq " on the set of categories
"a \leq b" means:
 - a-software is prerequisite for b-software.
 - b-software may depend on a-software, but not vice-versa
 - b-software may see a-software but not vice-versa
- Variability: Changes in any given category *a* only affect components of category *b* where $a \leq b$. The higher the category, the more likely are changes.
- **Thus:** Categories guide the design process, improve documentation and guarantee variability within the defined categories..

Communication Between Non-Related Categories



How do components of different categories communicate while preserving their category?

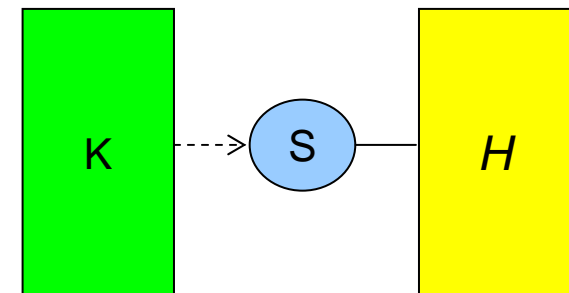
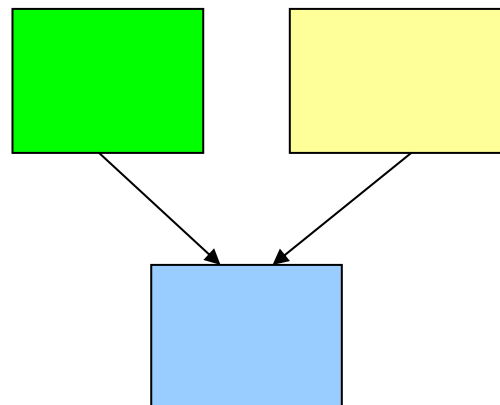
It is intuitive that:

A-software may use A-, 0 und R-Software, but not T
(it would be AT otherwise)

T-software may call T-, 0 und R-Software, but not A
(it would be AT otherwise)

Solution

Communication via
interfaces of a category
visible to either partner

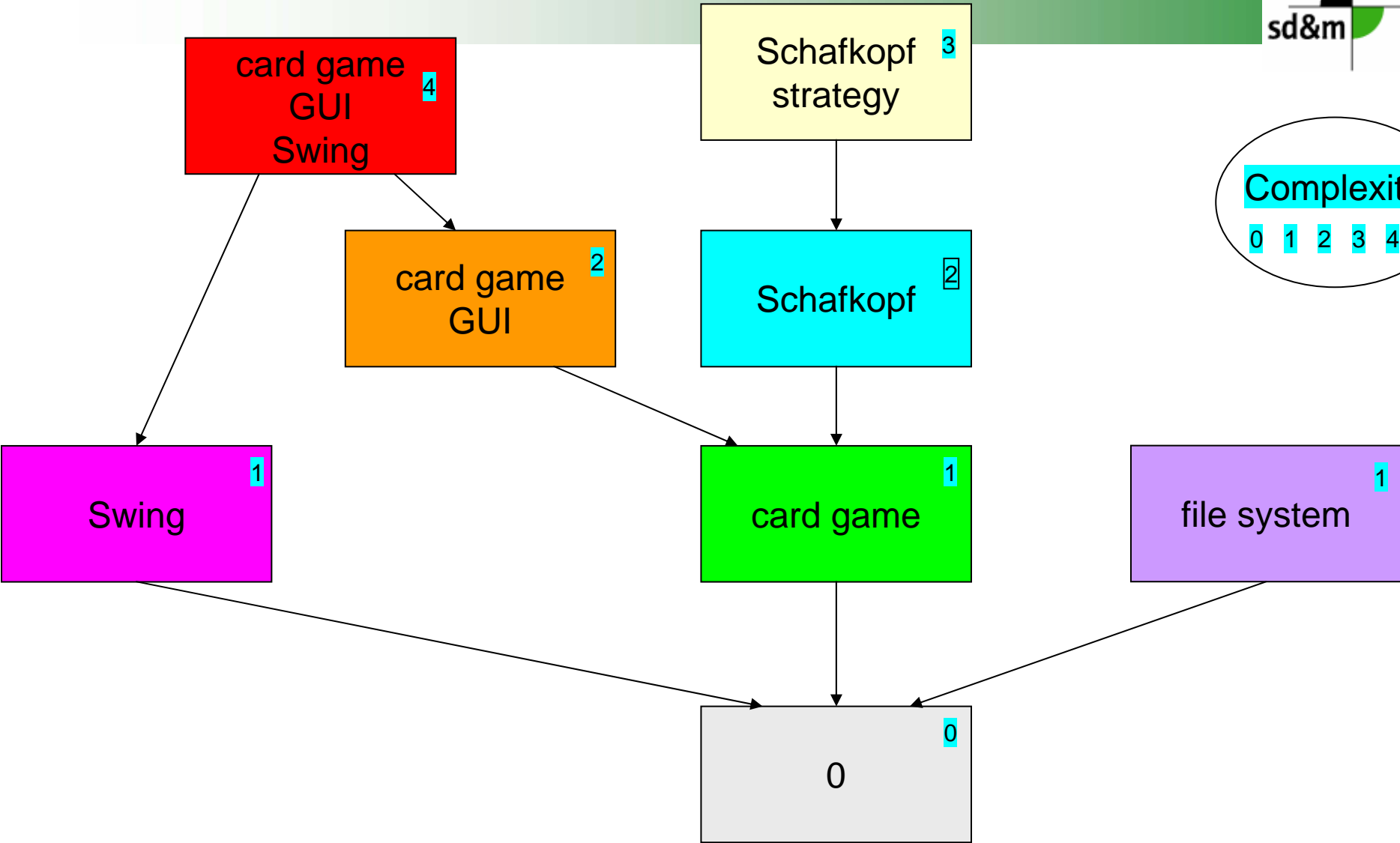
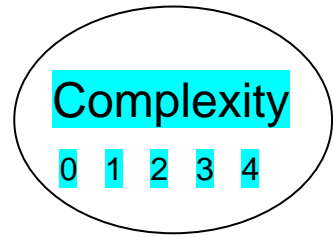


QCM: Quasar Complexity Measure Based on Categories



- Idea: Lower categories are desirable, higher or mixed categories should be minimized.
- QCM of a category = number of descendants in the category graph
- Interfaces don't count
- QCM of a composed component = weighted sum of the complexities of its composants, the weight being any measure, e.g. LOC.
- Note: QCM is no absolute measure – it is meant for comparing competing designs based on the same category graph.

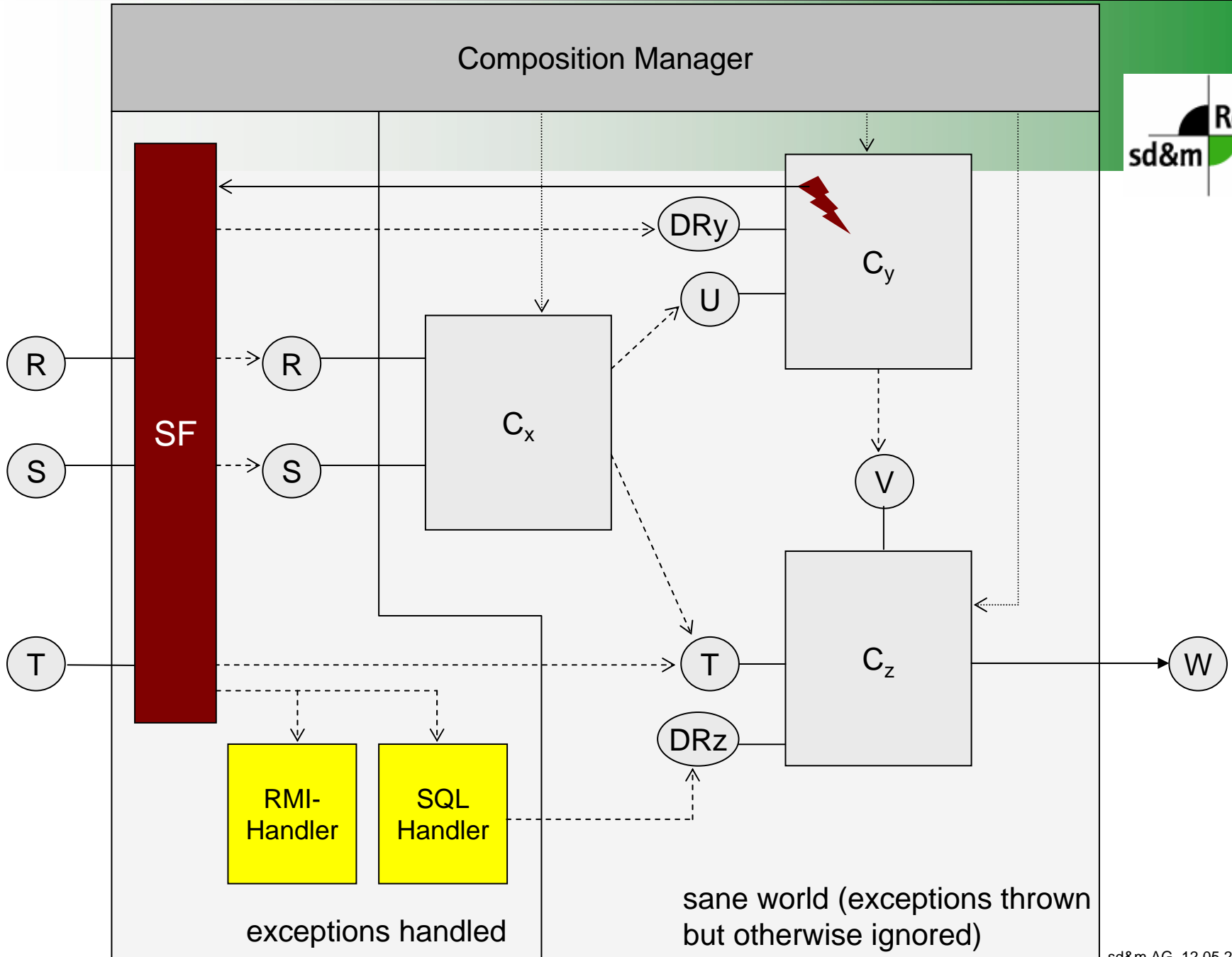
QCM Example



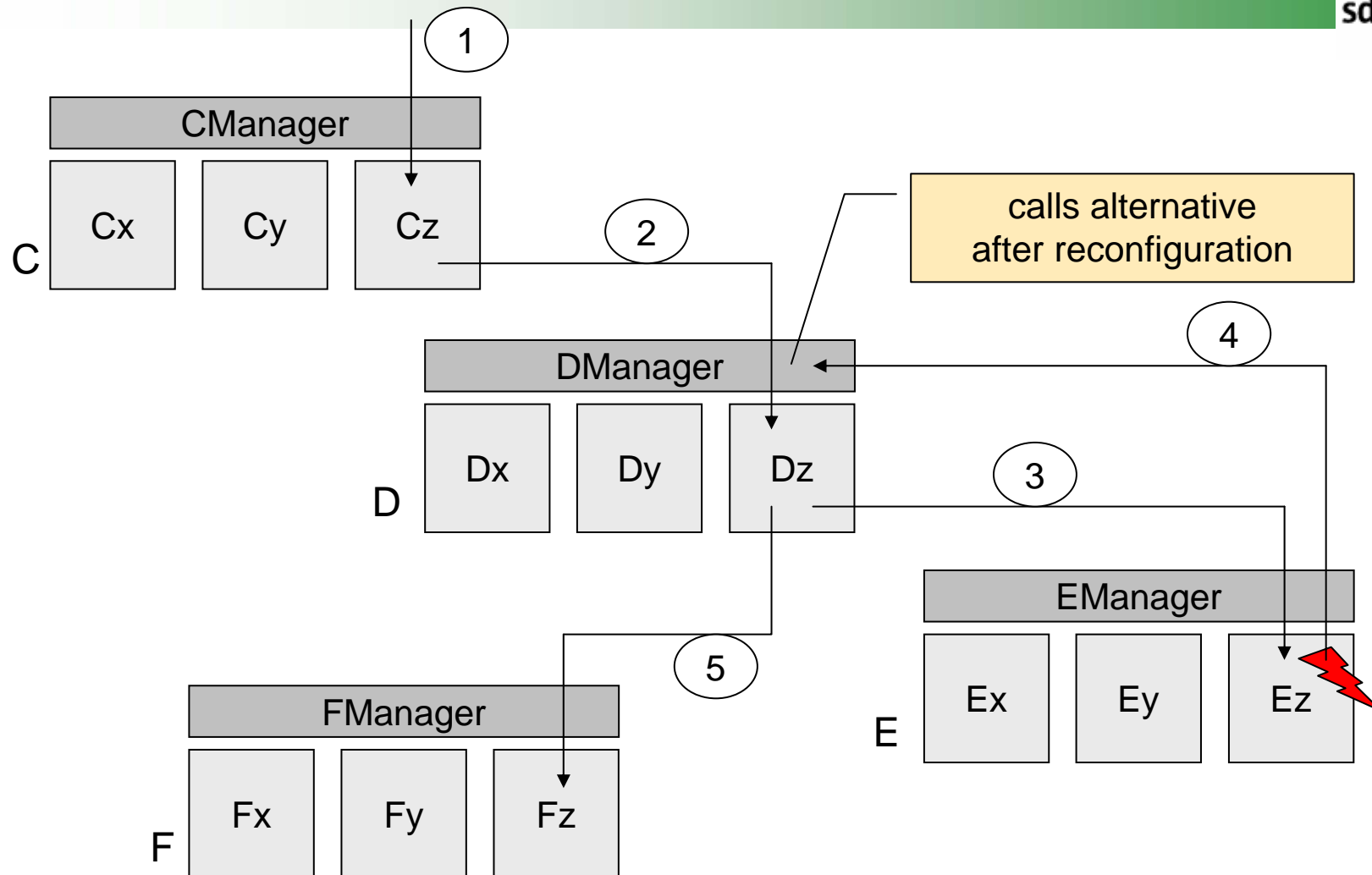
Errors and Exceptions by Quasar



- Errors are part of the interface definition, they occur all the time (Example: account overdrawn)
- Exceptions can almost never be enumerated (there are one million reasons for a car to break down). Exceptions are implementation dependent, and they almost never occur (at least in well-tested systems)
- Exceptions may be thrown at any time, at any place by anybody.
- Exceptions fly to the next security facade. Security facades are in charge of exception handling.
- Security facades delegate exception handling to diagnosis&repair experts (one per category). Components may export a dedicated D&R interfaces for use by D&R experts.
- Components are clustered to risk communities.
- Java, C#, Python or Ada exceptions are only loosely related to true exceptions (e.g. Java/InterruptedException or Python/StopIteration are not exceptional at all)



Risk Communities: A Robust Architecture



Next Steps



- 2nd edition:
 - Non-Functions
 - Specification of Interfaces
- Quasar by examples
 - case studies from different areas and domains: sd&m projects, student projects, mobile phones, Subversion
- Quasar Enterprise
 - Quasar applied to software landscapes
 - Quasar and SOA