

DESIGN TYPES & DESIGN CARDS

ARGUMENTATIV ZUR BESTEN DESIGN-ENTSCHEIDUNG



Christian Rehn:

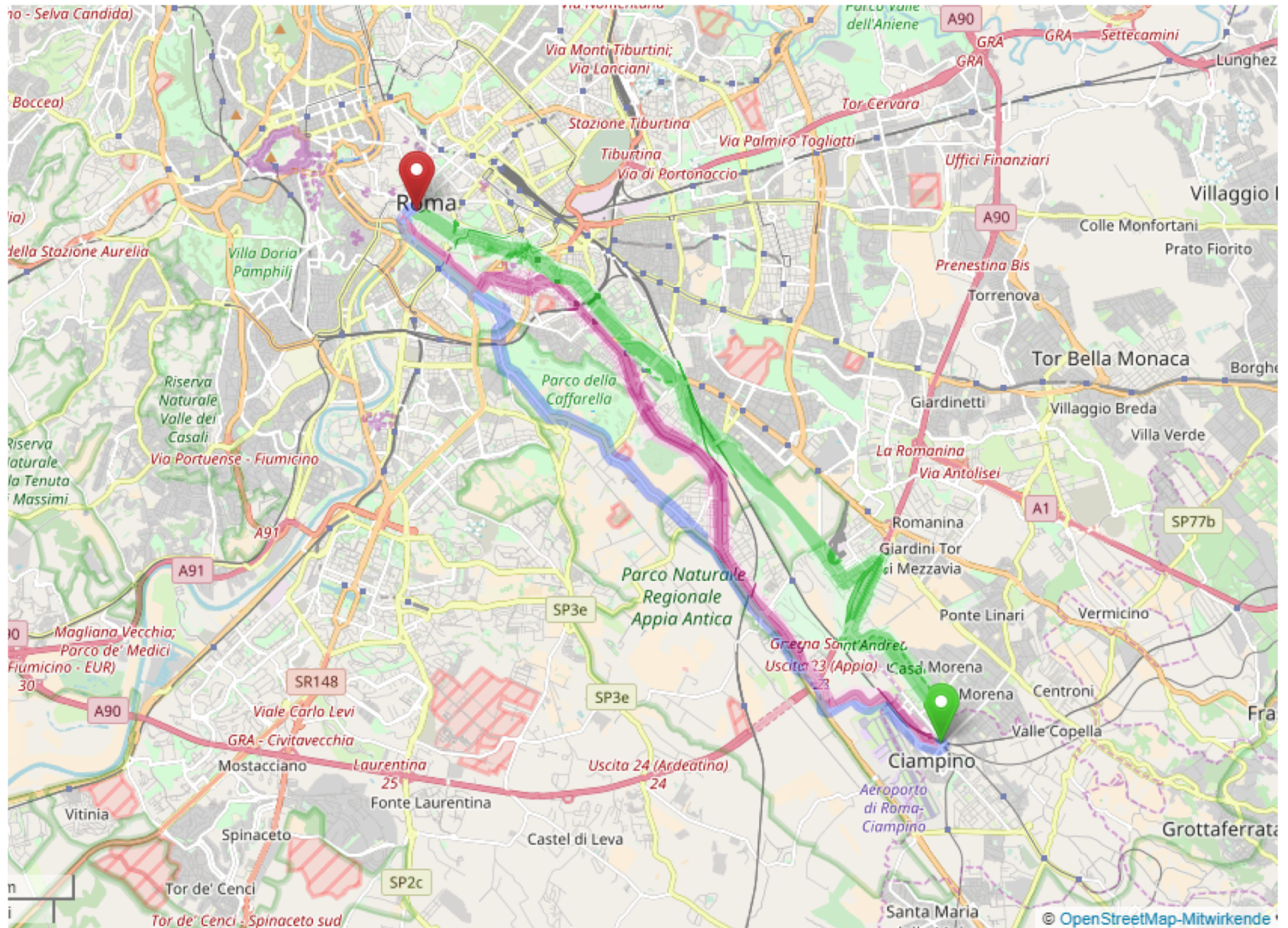
- Software Engineer
- www.principles-wiki.net
- www.design-types.net



Matthias Wittum:

- Head of Source Center
- www.design-types.net

Viele Wege führen nach Rom...



Szenario



vs.

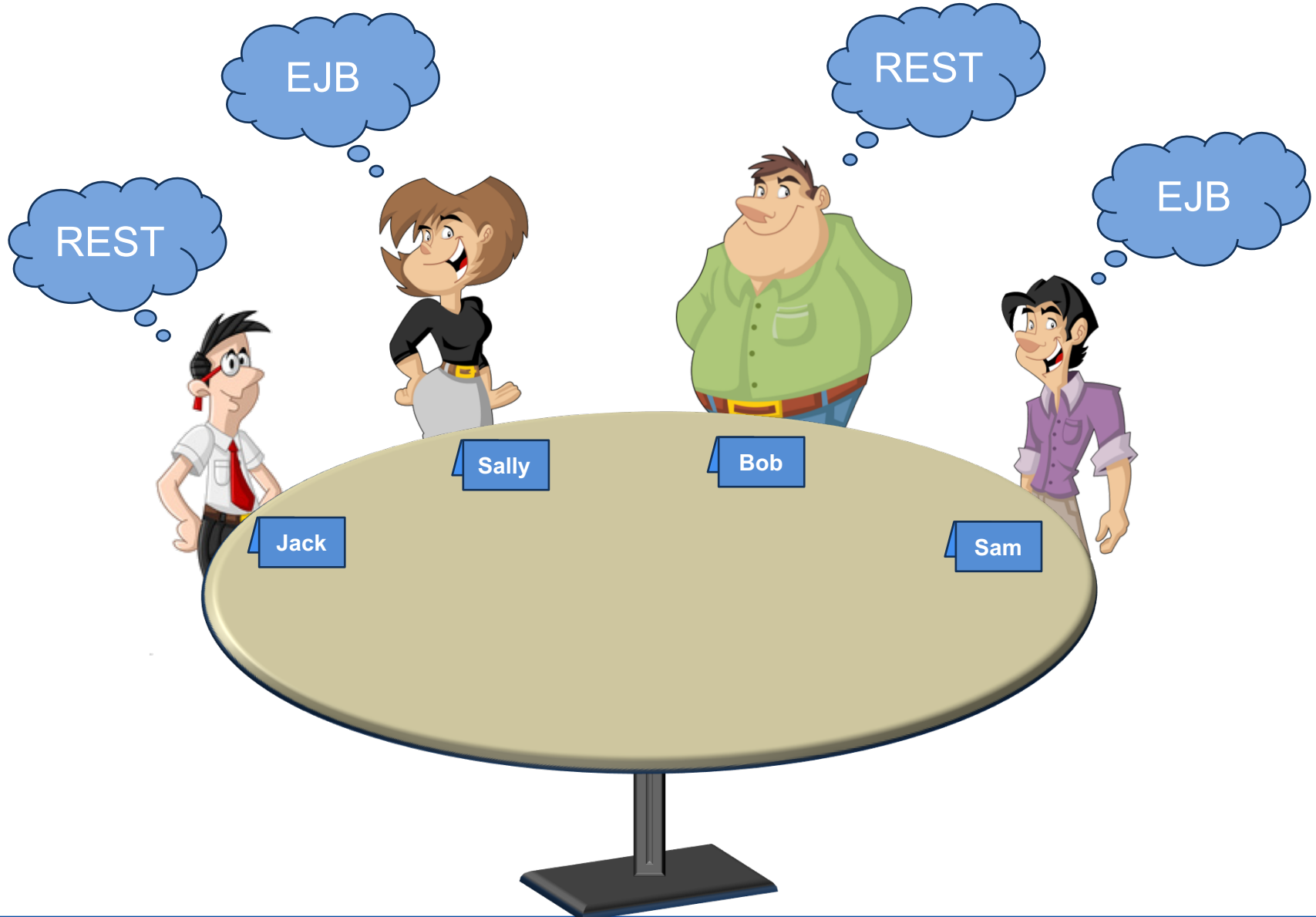
{ REST }

Ein völlig fiktives Szenario



Eine Gruppe Entwickler diskutiert, ob eine neue Schnittstelle per EJB oder REST angeboten werden soll.

Technische Diskussion: „die historische Variante“



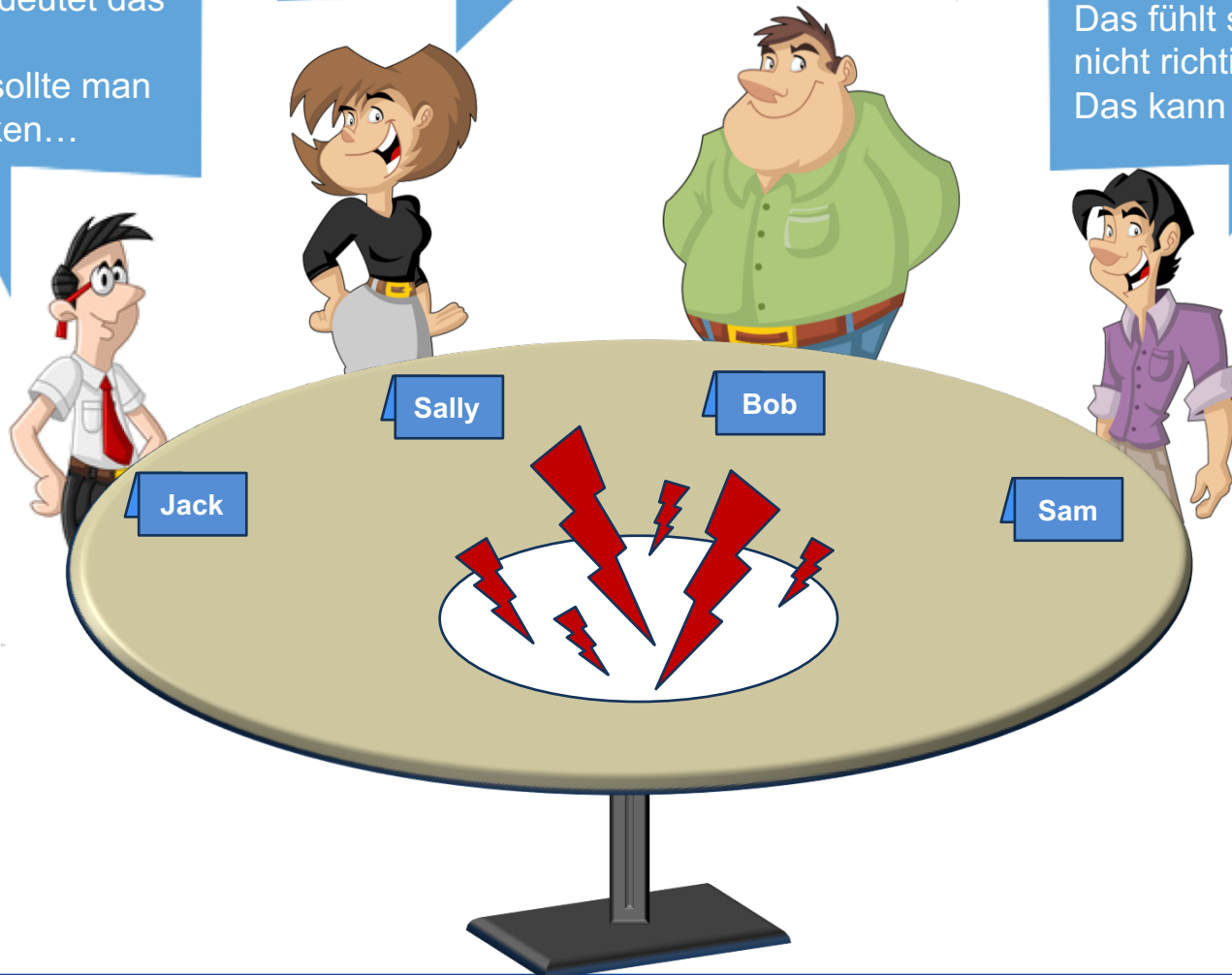
Technische Diskussion: „die historische Variante“

Aber wenn wir das so machen...
Und dann könnten wir noch...
Kombiniert bedeutet das dann...
Andererseits sollte man immer bedenken...

Wir haben schon immer EJB verwendet.
Warum jetzt REST?

Ich hab das schon oft mit REST gemacht. Das ist besser. Vertraut mir!

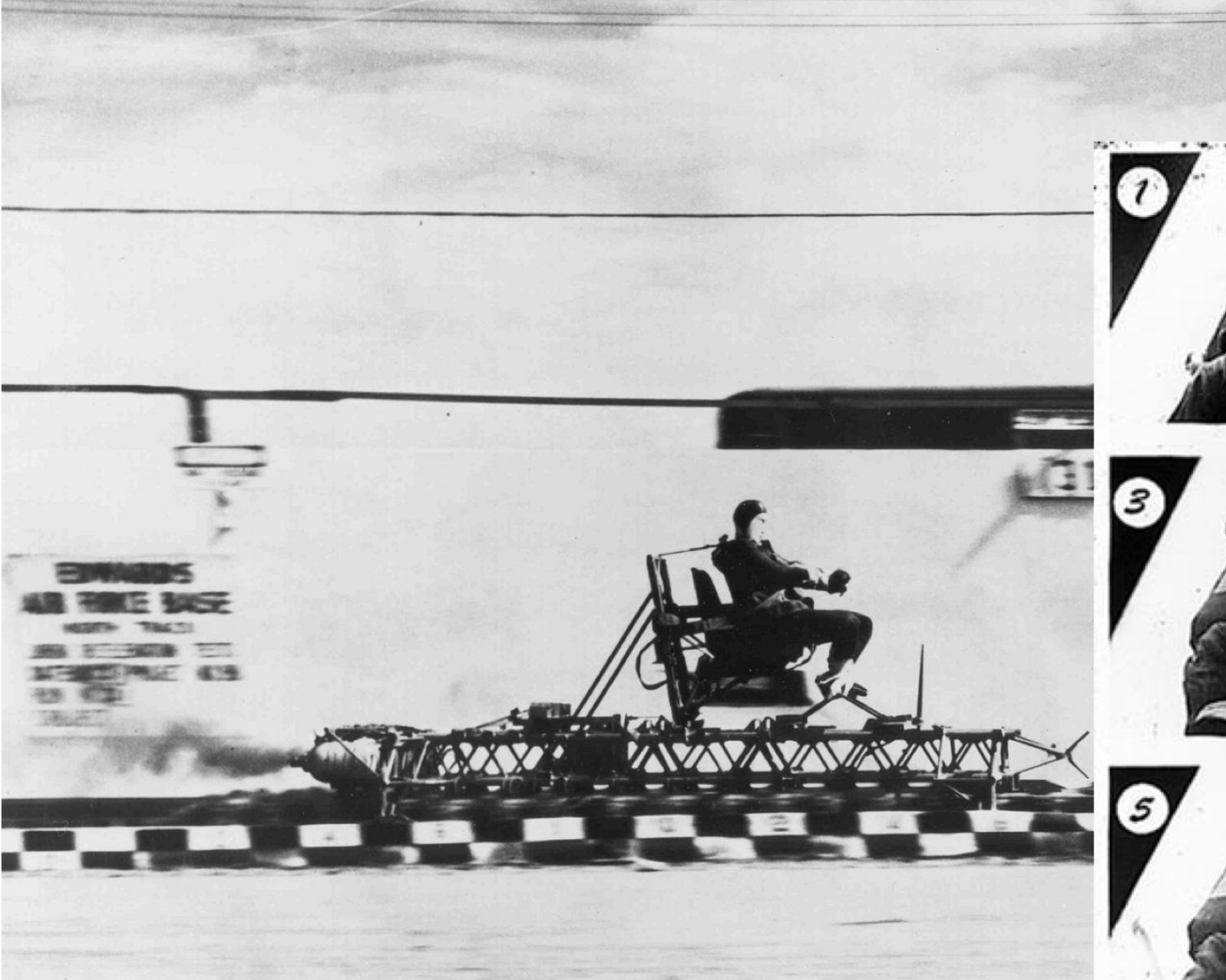
Das fühlt sich irgendwie nicht richtig an.
Das kann nicht gut sein!



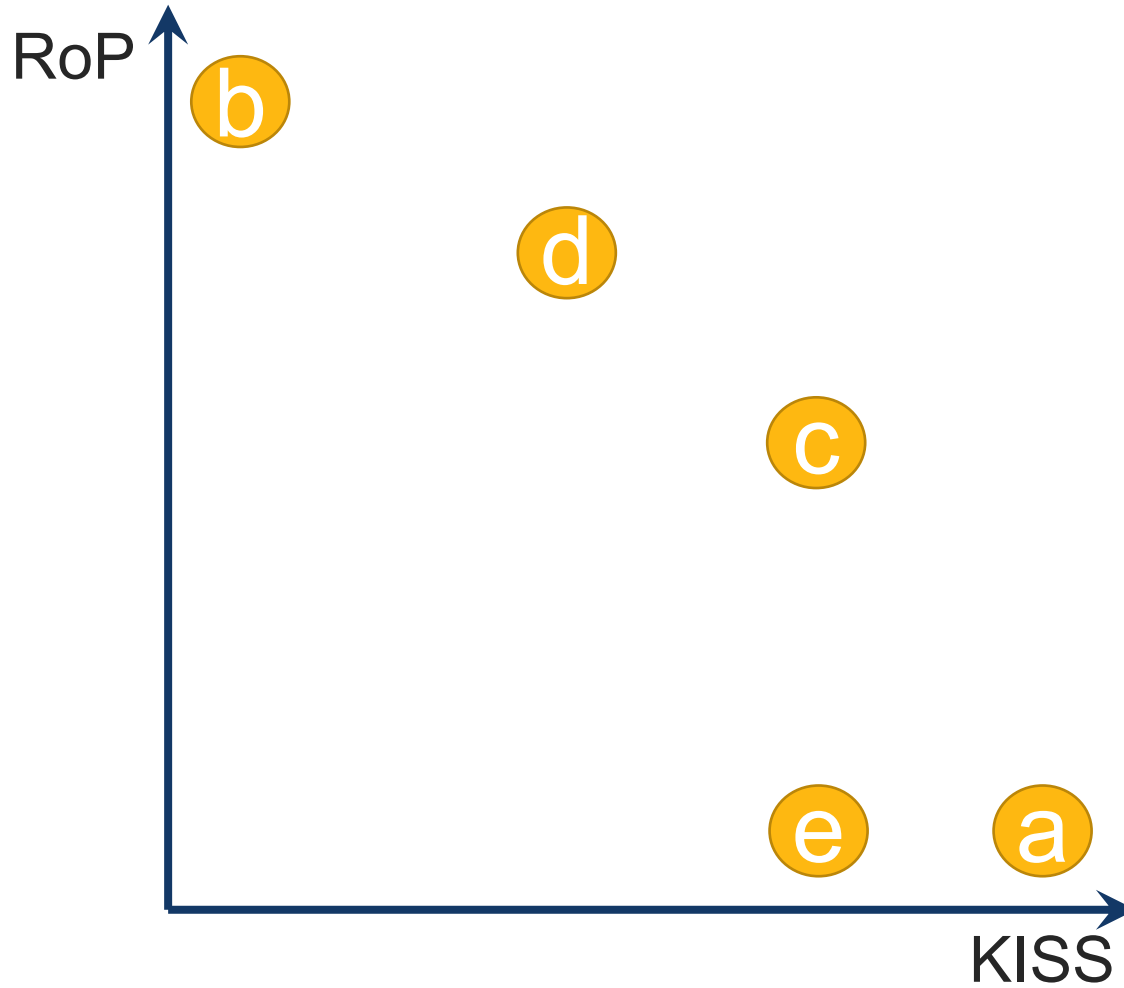
Überreden oder überzeugen?

Wir sind nicht die ersten...

John Paul Stapp und Edward A. Murphy



Konträre Prinzipien



IH/E

SDP

SoC

ML

KISS

ISP

SCP

DRY

SRP

LoD

OCP

FF

RoP

DIP

ECV

ZOI

MIMC

LC

UP

PSU

LLA

HC

EUHM

MP

TdA/IE

PLS

IAP

ADP

Design Cards – Argumentkarten


KISS: Keep It Simple Stupid

KISS

»Simple means readable, maintainable, and less error-prone. Overengineering is harmful.«

Complex code typically contains more bugs and it has to be maintained (maybe even by other people). To others it may seem obscure which can lead to frustration and bad code quality. Striving for simplicity means to avoid inheritance, low-level optimization, complex algorithms, fancy (language) features, configurability, etc.

↑RoP, ↑CF, ↓NFR, ↓MP



design-types.net


LC: Low Coupling

LC

»Tight coupling creates ripple-effects and makes the code less maintainable.«

If you decouple, you don't need to know internal details about other parts of the system. Furthermore it makes you independent from changes in those other parts and maybe even supports reuse. So better use additional layers, indirection, dependency injection, observers, messaging, etc.

↓KISS, ↑PoQ, ↑FP, ↓HC



design-types.net

CF: Customer Focus

CF

»This is not what the customer pays us for!«

If something is not requested, there has to be a very good reason to do it. Anything in addition costs additional time (also for removing or maintaining it). It creates additional risk of more bugs and makes you responsible for it. Continuously remember what was requested e.g. by looking into the requirements or asking the customer.

↓PoQ, ↑EaO, ↑YAGNI



design-types.net

ML: Murphy's Law

ML

»Avoid possibilities for something to go wrong or to get misused.«

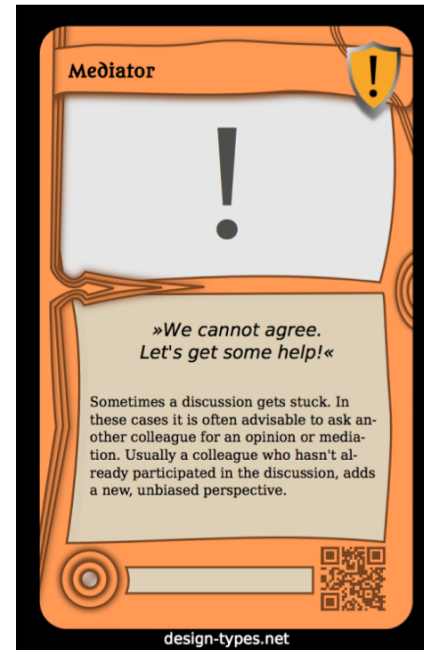
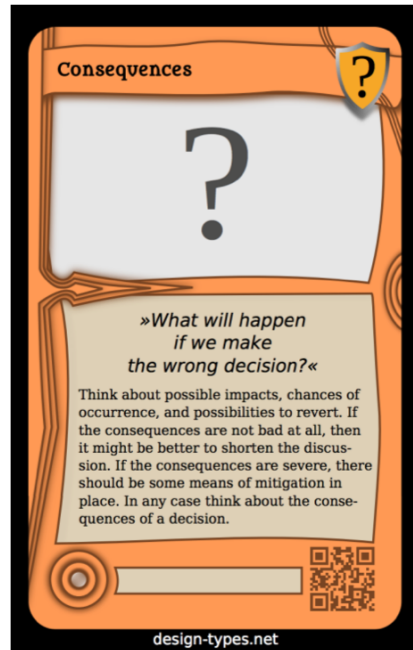
If there is a possibility for something to be used in the wrong way (like supplying parameters in the wrong order), it will eventually happen. So better avoid possible future bugs by using defensive programming, final, immutability, a common naming scheme, avoiding duplication and complexity.

↓KISS, ↑DRY, ↑UP, ↓FP



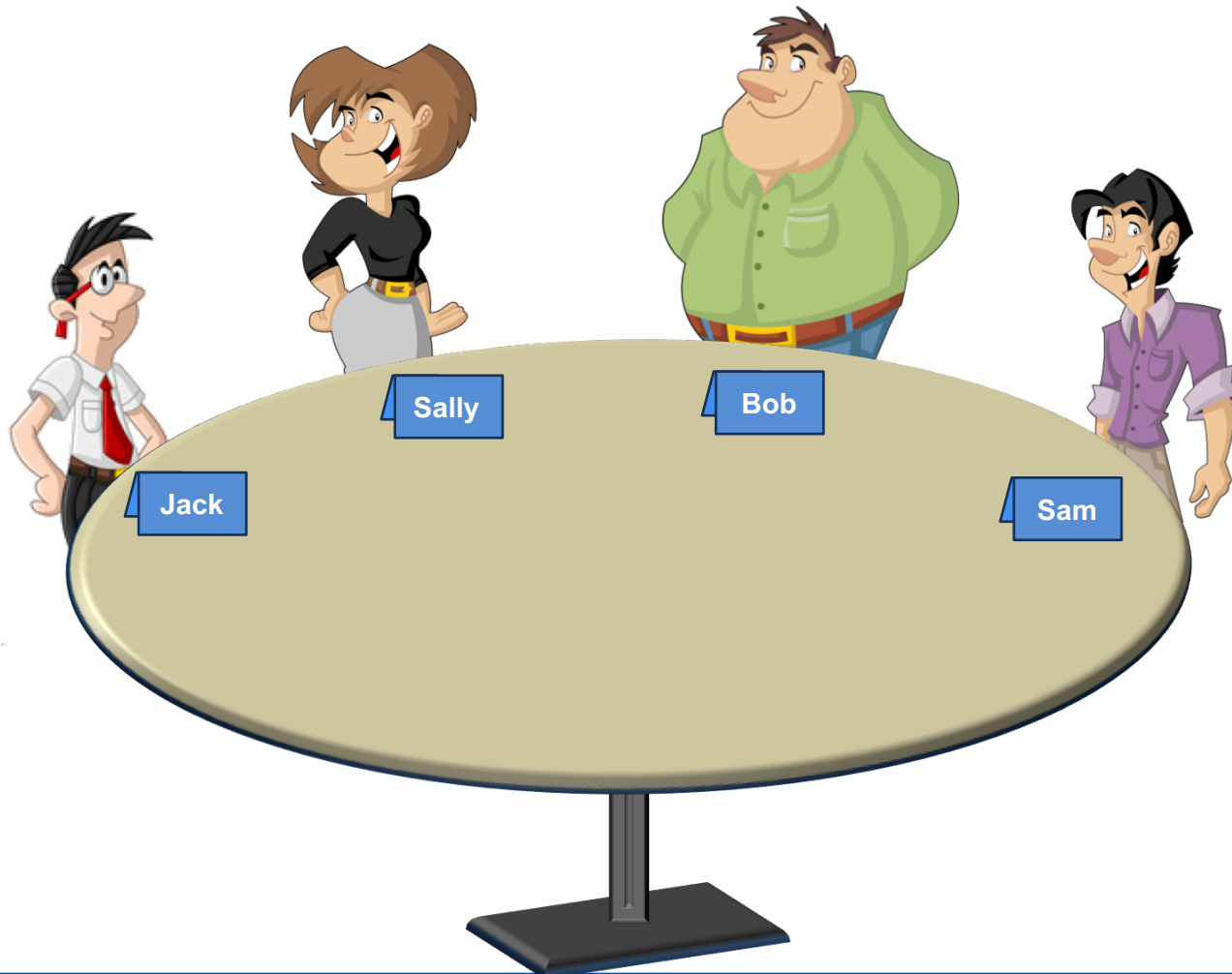
design-types.net

Design Cards – Moderationskarten



Technische Diskussion: „kartengestützte Argumentation“

Ziel: Einsatz von nachvollziehbaren Argumenten



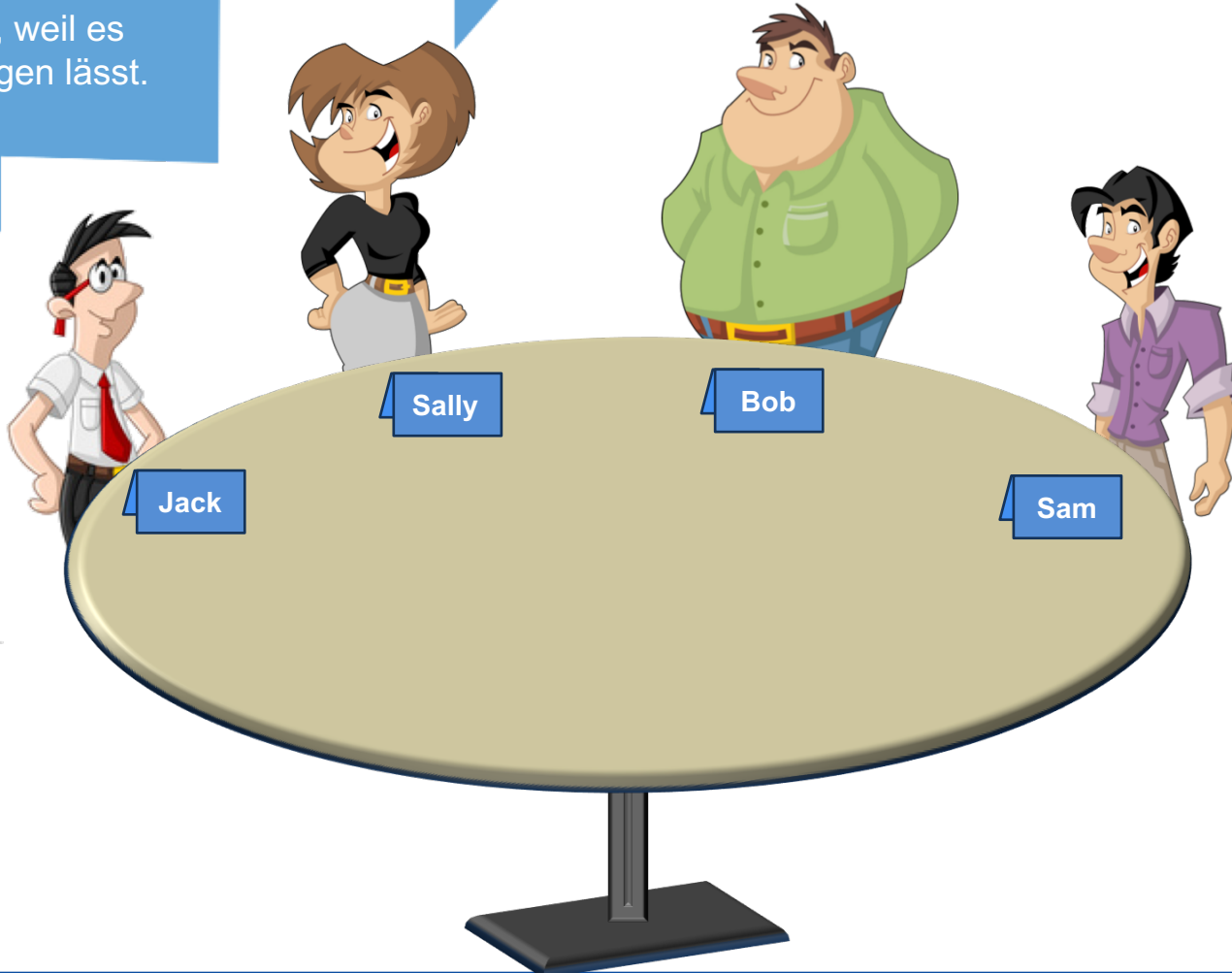
Technische Diskussion: „kartengestützte Argumentation“

REST ist einfach, weil es sich leicht debuggen lässt.

Bei EJB muss ich nur eine Annotation setzen. Das ist viel einfacher als bei REST.

KISS

KISS



Jack

Sally

Bob

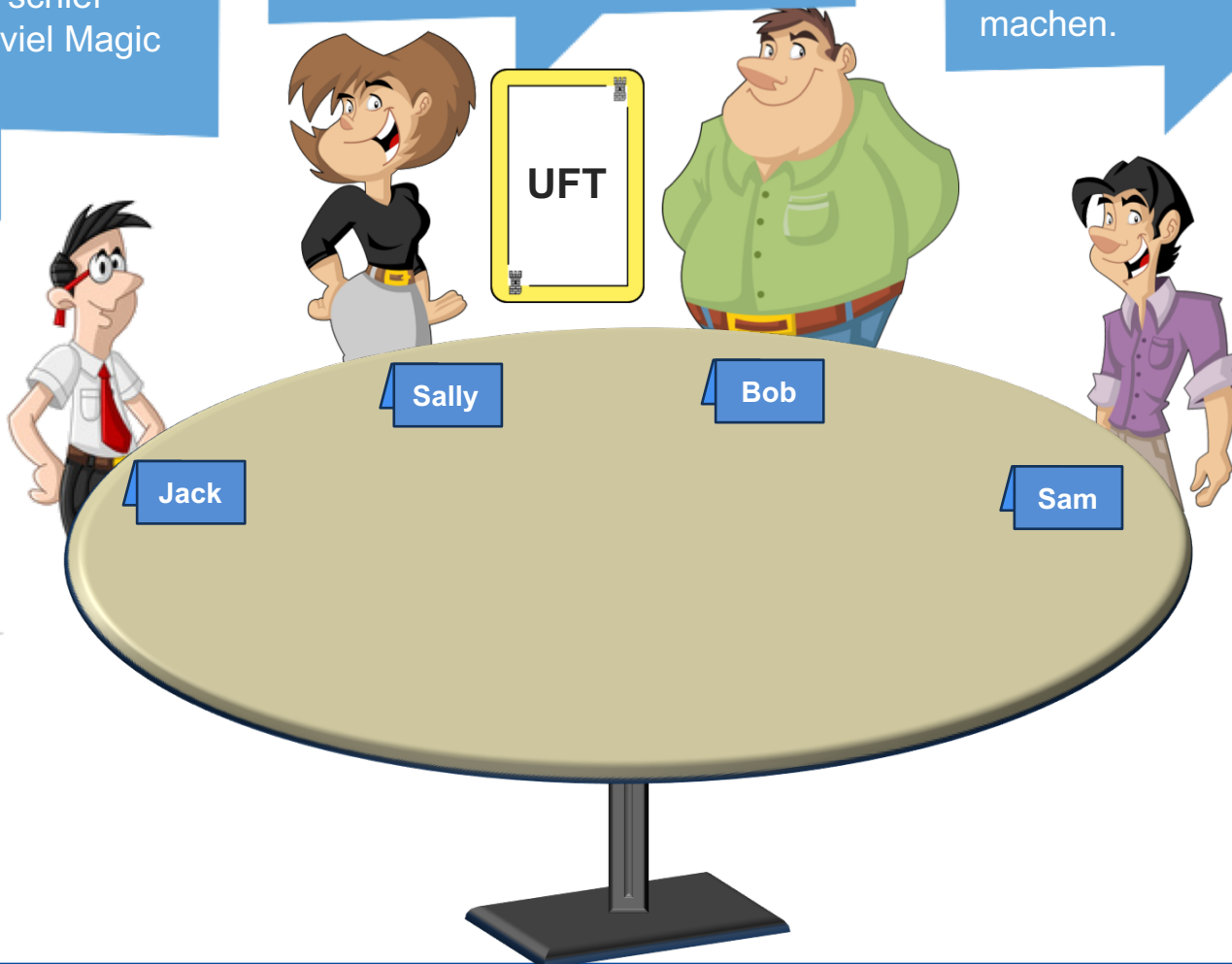
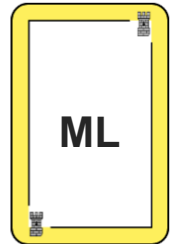
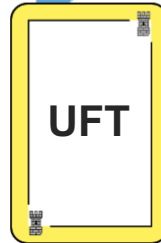
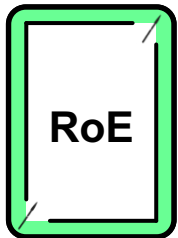
Sam

Technische Diskussion: „kartengestützte Argumentation“

Die Kommunikation ist expliziter. Einfach HTTP. Da kann weniger schief gehen, als wenn viel Magic da wäre.

Wir sollten nicht ohne Grund eine neue Technologie einführen, wo uns die Erfahrung fehlt.

Weniger schief gehen? REST ist erstmal anders und man kann viel falsch machen.



Jack

Sally

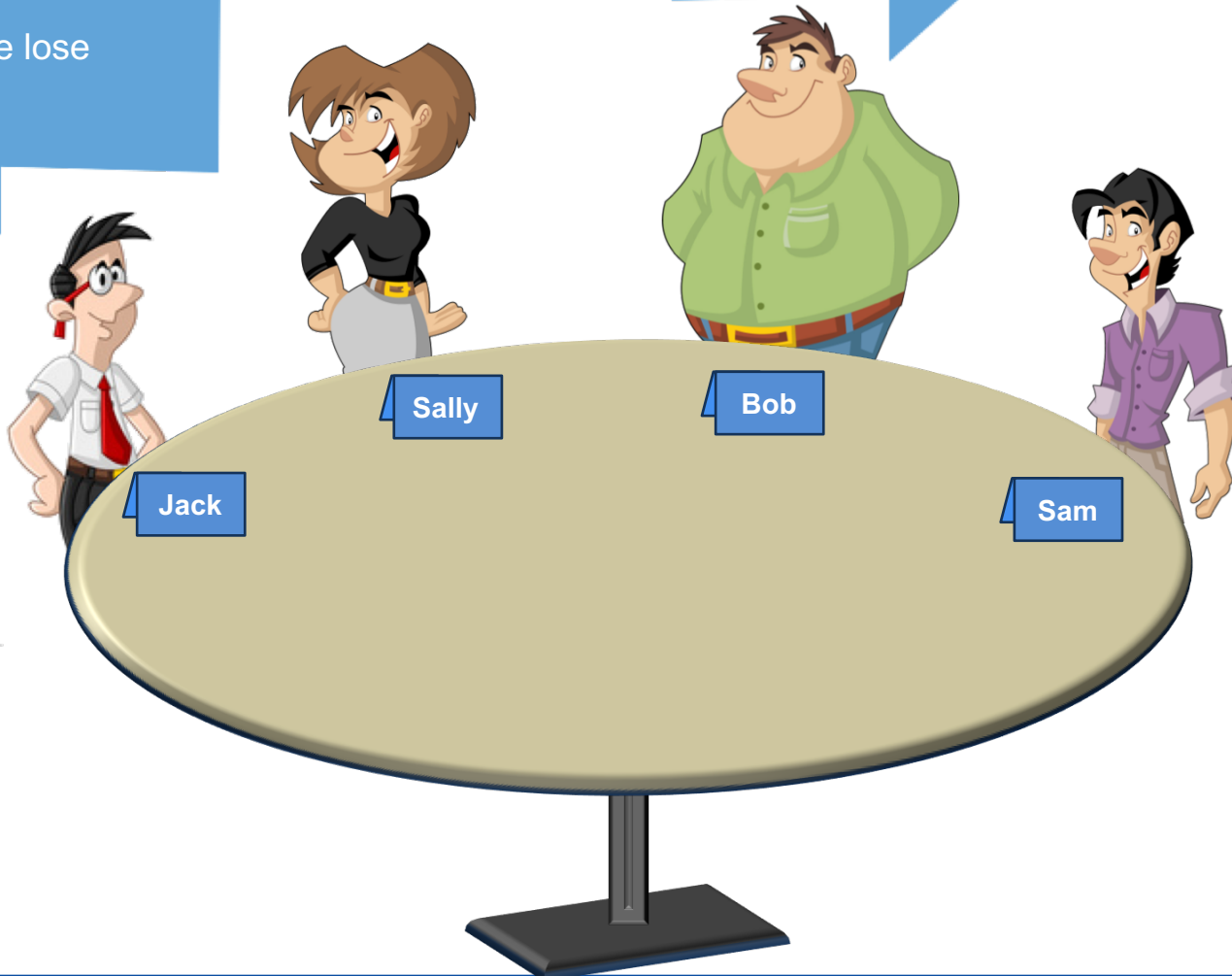
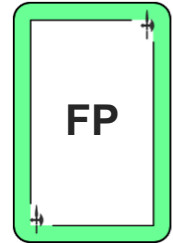
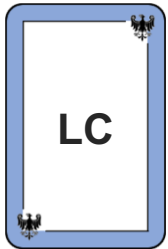
Bob

Sam

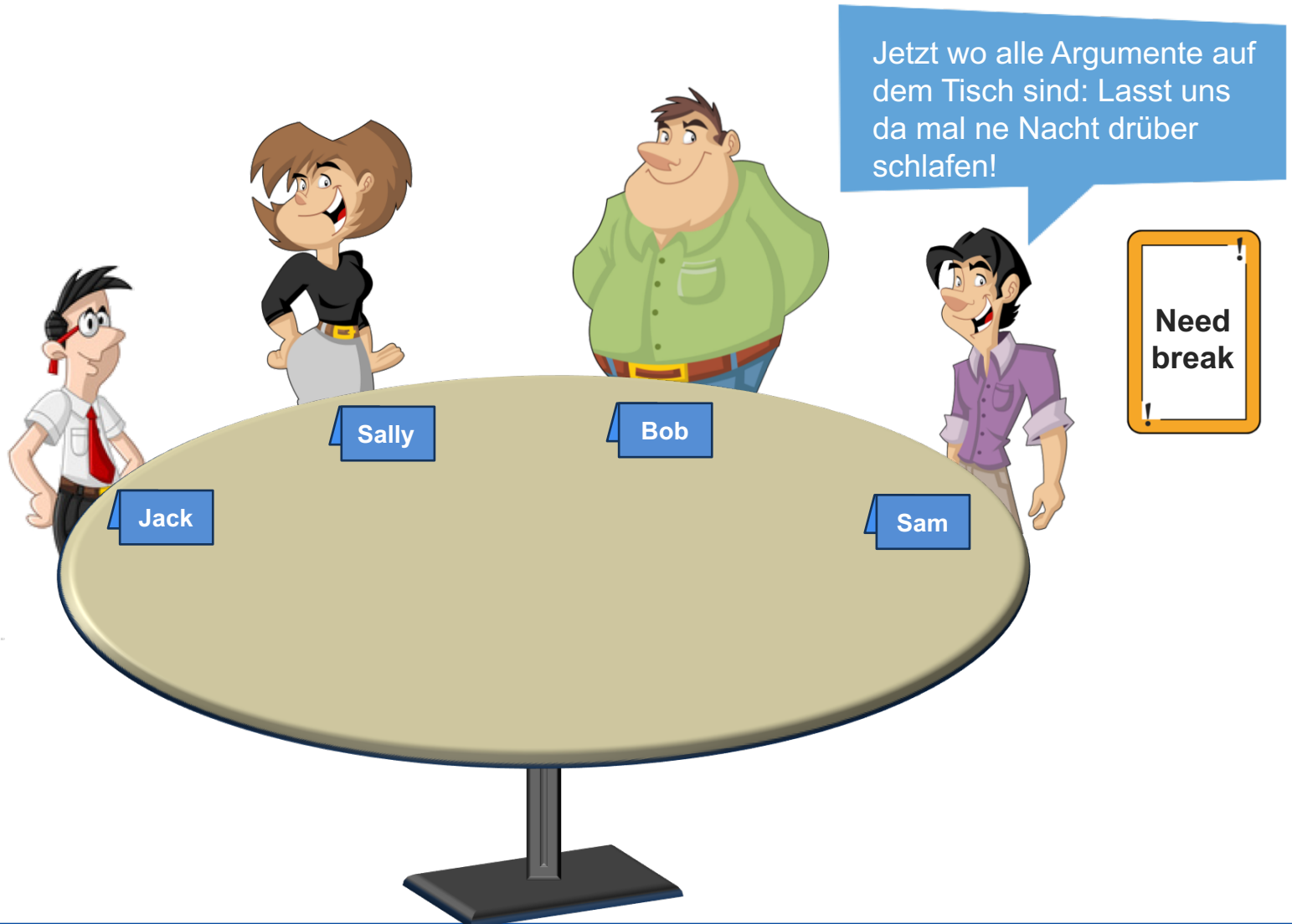
Technische Diskussion: „kartengestützte Argumentation“

Ein Grund ist die lose Kopplung.

Genau. Mit REST können wir unsere API viel flexibler versionieren.

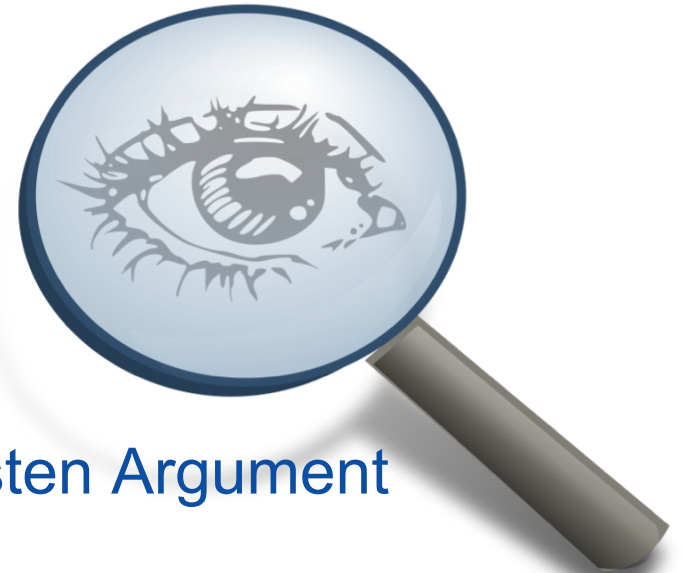


Technische Diskussion: „kartengestützte Argumentation“



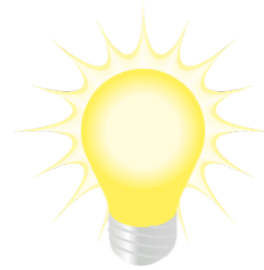
Erste Beobachtung

- Die Entwickler argumentieren klarer und nachvollziehbarer
- Ein Argument leitet zum nächsten Argument oder Gegenargument über

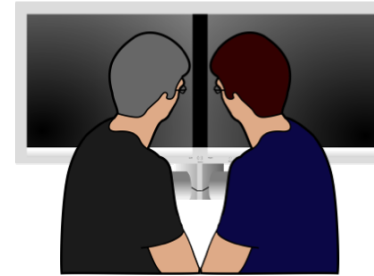


Einsatzgebiete

- Konzeption



- Pair Programming



- Code Reviews



- Gamification



- Lerneffekt

Die Karten im Detail





**Wir haben noch gar keinen Konsens
gefunden!**

Die Dimensionen unserer Entwickler-Typologie

Simple

vs.

Powerful

Abstract

vs.

Concrete

Pragmatic

vs.

Idealistic

Robust

vs.

Technologic

Die konkreten Ausprägungen

Simple means:

- keeping it simple for better understandability
- omit unnecessary things (lower risk; less bugs)
- reduce complexity by splitting it up
- prefer explicit solutions instead of implicit knowledge
- etc.

Powerful means:

- powerful and generalized solutions
- Flexibility and Extensibility by foresighted design
- config. solutions instead of frequent code changes
- mastering complexity
- etc.

Abstract means:

- think in concepts and abstractions
- focus on the big picture and interaction of components
- know about the consequences of a change
- focus on real world models
- etc.

Concrete means:

- think and act in code
- transferring ideas into components immediately
- optimizing algorithms for better performance
- understanding systems by reading the code
- etc.

Pragmatic means:

- fulfill requirements asap
- use only things that guarantee a value
- omit unnecessary things
- bring others down to earth
- etc.

Idealistic means:

- make things right – not only 80%
- consider all aspects not only functional ones
- everything has its right place
- do not misuse existing concepts
- etc.

Robust means:

- protect applications against risks and potential bugs
- use standards for an obvious structure
- avoid magic and complexity
- use proven solutions which stand the test of time
- etc.

Technologic means:

- using new, modern and more productive technologies
- evolve with technology for being more competitive
- broaden their personal horizon
- etc.

S

C


I

R

Ausprobieren und lernen > www.design-types.net


Design Type Test Yourself Play Around Learn More principles-wiki.net

How Do You Design Software?




Learn more about the idea of principles and design types.

[Learn More](#)



Do you want to know how you design software? Fill out the questionnaire.

[Test Yourself](#)



Ever wondered why you always argue with your colleagues?

[Assess Your Colleagues](#)



Simple
Abstract
Idealistic
Technologic



Simple
Abstract
Pragmatic
Robust

Ein Beispiel-Ergebnis

► SAPR: The Construction Manager



Description

The Construction Manager loves to work like on a construction site. There is a plan and everybody works hand in hand to reach the aimed goal. He focuses on working solutions that are built on proven technologies. This ensures that the result will stand the test of time. The most matching motto is: Getting things done. He rather implements by himself than choosing the wrong and maybe unstable framework. He knows very well about his abilities and has reservations about foreign technologies that did not proof their maturity over a certain period of time. He also focuses more on the interaction of particular modules instead of having too many sophisticated and complex constructs in his design. He prefers simple craftsmanship which tells him not to finish before a certain level of robustness has been shown by manual or automated tests.

Your designs are

Stable and reasonably planned without unnecessary complexity

Programming is

Like managing a construction site. Something has to be built.

Principles you prefer

KISS, MIMC, RoE

Principles you reject

GP, PSU, TdA/IE

Strengths

- Fast in development
- Code and architecture

Suggestions

- Keep your design simple
- Don't get lost in details
- Keep your design continuous
- Don't forget about speed.



Your Design Type: The Construction Manager (SAPR)

Simple

This means you prefer simple, straightforward solutions

Abstract

This means you always have the big picture in mind

Pragmatic

This means you like getting things done fast

Robust

This means you strive for stable and robust software

Your designs are:

Stable and reasonably planned without unnecessary complexity

Programming is to you:

Like managing a construction site. Something has to be built.

Dimension overlap

Simple	Powerful
Abstract	Concrete
Pragmatic	Idealistic
Robust	Technologic

Type overlap



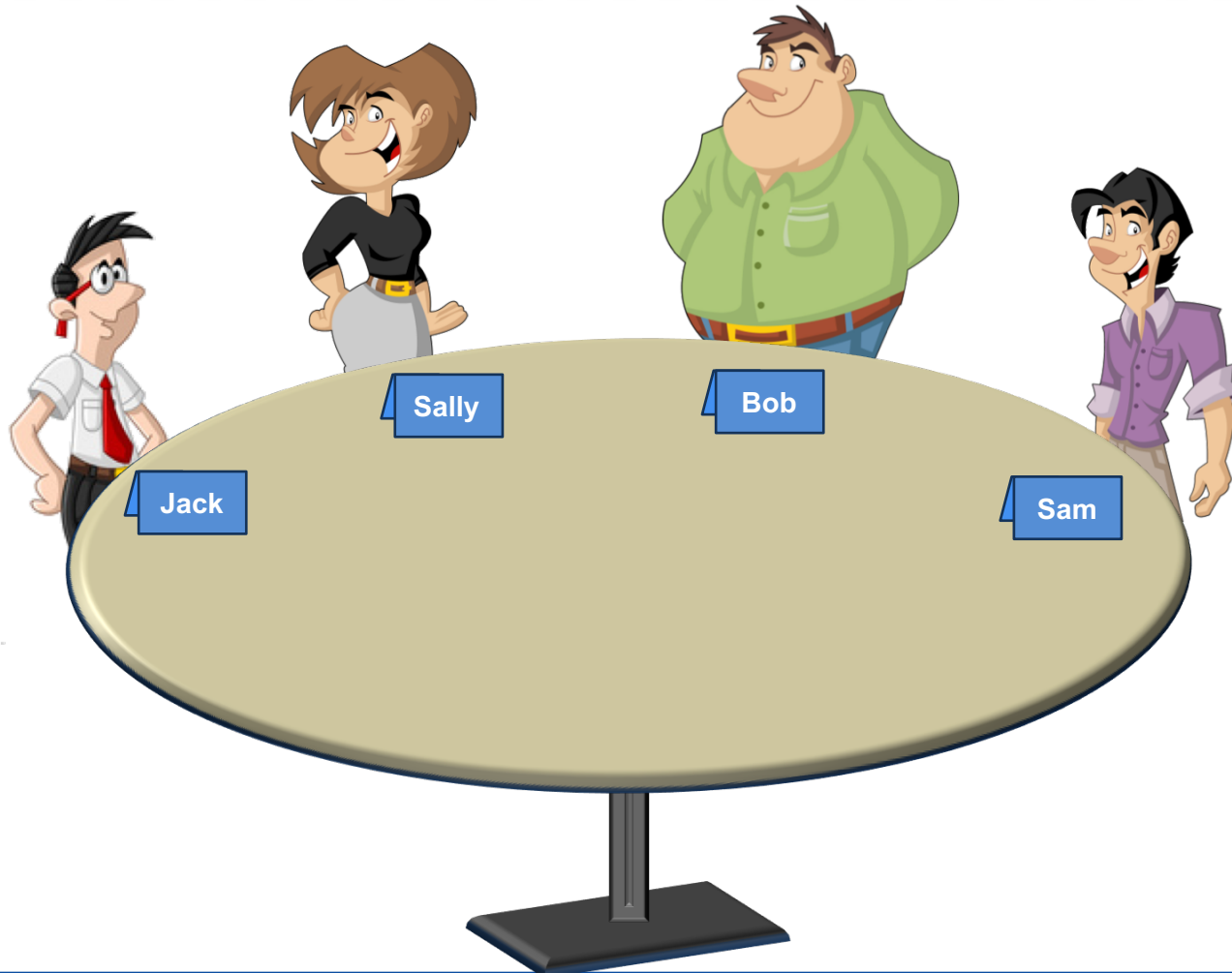
design-types.net



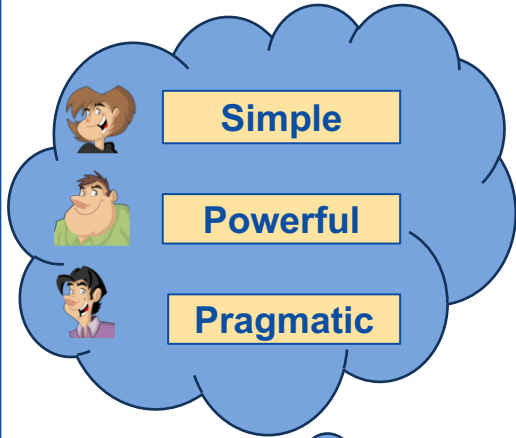
Like it? Print it!

Technische Diskussion: „Diskutieren mit Kontext“

Der nächste Tag...



Technische Diskussion: „Diskutie



Ja, aber das ist doch viel zu kompliziert! In einem halben Jahr will und muss ich das auch noch verstehen.

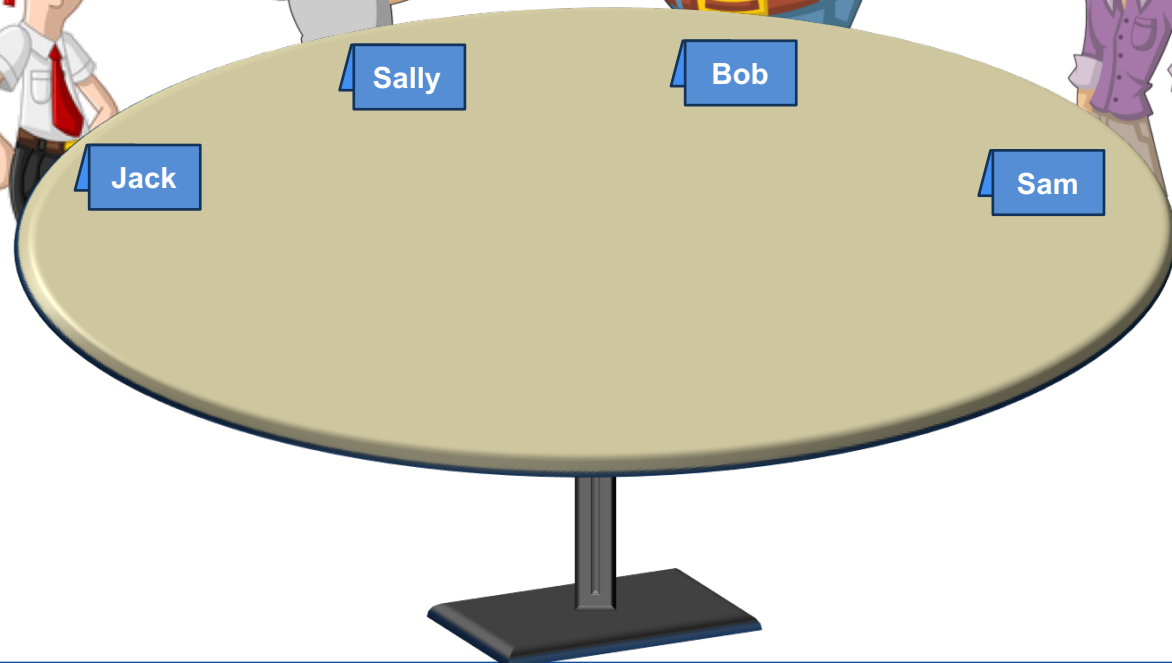
Simple

In einem halben Jahr kommt bestimmt eine neue Anforderung und wir müssen die Schnittstelle erweitern...

Powerful

Lasst uns bitte die Deadline nicht vergessen. Wir sollten uns nicht verzetteln!

Pragmatic



Technische Diskussion: „Diskutieren mit Kontext“

Als **Abstract** ist mir eine lose Koppelung wichtig.

Simple

Powerful

Pragmatic

Sally

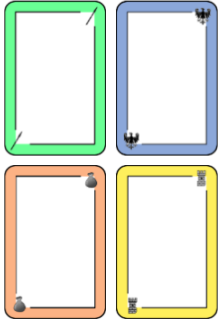
Bob

Sam

Mit REST können wir auf die bereits bestehende Web-Infrastruktur aufsetzen und müssen z.B. Caching nicht neu implementieren. Das macht uns schneller.



Was nehmen wir mit?



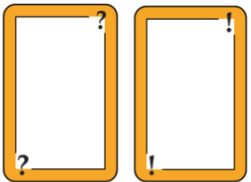
- Gute und nachvollziehbare Argumente

- Design Cards



- Gegenseitiges Verständnis für unterschiedliche Positionen

- Design Types



- Um Blockaden oder Patt-Situation auflösen zu können, benötigt man Exitstrategien

- Moderationskarten

Gibt es noch mehr?

Design Matrix

Description of design challenge	Result of design decision
Name: <input type="text"/>	Date: <input type="text"/> <input type="checkbox"/> Approved <input type="checkbox"/> Rejected
Topic overview & Solution details: <input style="width: 100%; height: 100%;" type="text"/>	Decided by: <input type="text"/>
If useful, link relevant documents	Stakeholder: <input type="text"/>
	Summary: <input style="width: 100%; height: 100%;" type="text"/>

Simple	<ul style="list-style-type: none"> <input type="checkbox"/> Is the solution easy to understand (even in the future)? Is there a solution that is easier? <input type="checkbox"/> Does it avoid „clever“ magic and overly generic approaches? <input type="checkbox"/> Is the solution explicit so there is less room for misinterpretation or for ugly surprises? <input type="checkbox"/> Are there unrequested features we can omit? 	notes	notes	<ul style="list-style-type: none"> <input type="checkbox"/> Is the solution foresighted enough? <input type="checkbox"/> Does it take non-functional requirements into account? <input type="checkbox"/> Is the solution generic and reusable? <input type="checkbox"/> Which parts will change in the near future? Which ones continuously? What should stay stable, what flexible? 	Powerful
--------	---	-------	-------	--	----------

Ziel: Systematische und strukturierte Herangehensweise

Abstract	<ul style="list-style-type: none"> <input type="checkbox"/> on its own? <input type="checkbox"/> Are modules cohesive and is coupling low? 	notes	notes	<ul style="list-style-type: none"> <input type="checkbox"/> the same breath? <input type="checkbox"/> Can the solution grow naturally over time? (e.g. allow further changes/refactorings) 	Concrete
----------	--	-------	-------	--	----------

Pragmatic	<ul style="list-style-type: none"> <input type="checkbox"/> Does the solution provide value early on? <input type="checkbox"/> Does the solution really address the customer's goals/use cases? <input type="checkbox"/> Does the solution really fit to the timeline? <input type="checkbox"/> Can we use already existing Code (snippets, libraries, services)? 	notes	notes	<ul style="list-style-type: none"> <input type="checkbox"/> Is this the right solution? <input type="checkbox"/> Is it consistent with the rest of the system? <input type="checkbox"/> Is ensured that there are no workarounds or bad decisions that will produce serious problems later? 	Idealistic
-----------	---	-------	-------	--	------------

Robust	<ul style="list-style-type: none"> <input type="checkbox"/> Is the solution hard to misuse? <input type="checkbox"/> Are the chances for something to go wrong minimized? <input type="checkbox"/> Are standards used and adhered to? <input type="checkbox"/> Are used technologies/libraries stable? <input type="checkbox"/> Do all involved people have the necessary knowledge? 	notes	notes	<ul style="list-style-type: none"> <input type="checkbox"/> Is there already an existing technology or library that helps us? <input type="checkbox"/> Is the solution state-of-the-art? <input type="checkbox"/> Is the solution a technologic progress? <input type="checkbox"/> Can we get rid of legacy code? 	Technologic
--------	---	-------	-------	---	-------------

Stand der Dinge



- Design Types
 - Fertig
 - > 2300 Teilnehmer bisher



- Design Matrix
 - Fertig
 - Aktuell Feedback durch Pilotgruppen
 - Demnächst zum Download



- Design Cards
 - 26/54 Karten fertig (Basic Set)
 - Online-Karten gerade im Entstehen
 - Aktuell Feedback durch Pilotgruppen


Design Cards – Illustrationen

TP: Technological Progress 

TP

»Progress must not be ignored in a competitive environment.«

New technology is not only motivating but also comes with benefits like more features, more performance, better maintainability, and fixed bugs. Furthermore old technology won't be supported for much longer and new people don't know the old stuff anymore. Continuously challenge existing solutions by evaluating alternatives.

↑FRD, ↓UFT, ↓IR, ↑RoS 

design-types.net

TP: Technological Progress 



»Progress must not be ignored in a competitive environment.«

New technology is not only motivating but also comes with benefits like more features, more performance, better maintainability, and fixed bugs. Furthermore old technology won't be supported for much longer and new people don't know the old stuff anymore. Continuously challenge existing solutions by evaluating alternatives.

↑FRD, ↓UFT, ↓IR, ↑RoS 

design-types.net

PoQ: Principle of Quality 

PoQ

»Bad quality kills us in the long run!«

It may be faster now but we need to be fast tomorrow, too. Bad quality frustrates maintainers, makes fixing bugs harder and leads to huge efforts for changes. This often starts by being careless once. Don't let a vicious circle begin. Use metrics, adhere to the architecture, have a high test coverage, apply code reviews and continuous refactoring. Don't be lazy.

↑KISS, ↑LC, ↑EaO, ↓CF 

design-types.net

PoQ: Principle of Quality 



»Bad quality kills us in the long run!«

It may be faster now but we need to be fast tomorrow, too. Bad quality frustrates maintainers, makes fixing bugs harder and leads to huge efforts for changes. This often starts by being careless once. Don't let a vicious circle begin. Use metrics, adhere to the architecture, have a high test coverage, apply code reviews and continuous refactoring. Don't be lazy.

↑KISS, ↑LC, ↑EaO, ↓CF 

design-types.net

DRY: Don't Repeat Yourself 


DRY


»Duplication makes changing the code cumbersome and leads to bugs.«

Having a functionality more than once means to update or bugfix it at every occurrence which is more error-prone and more effort. Refactorings like method or class extraction may help as well as inheritance, polymorphism and some design patterns.

↓KISS, ↑RoP, ↑PoQ, ↑ML 


design-types.net

DRY: Don't Repeat Yourself 



»Duplication makes changing the code cumbersome and leads to bugs.«

Having a functionality more than once means to update or bugfix it at every occurrence which is more error-prone and more effort. Refactorings like method or class extraction may help as well as inheritance, polymorphism and some design patterns.

↓KISS, ↑RoP, ↑PoQ, ↑ML 

design-types.net

Danke!

Fragen?



Christian Rehn

Wer informiert bleiben möchte, noch Fragen, Anregungen o.ä. hat, kann uns gerne eine Mail schreiben:

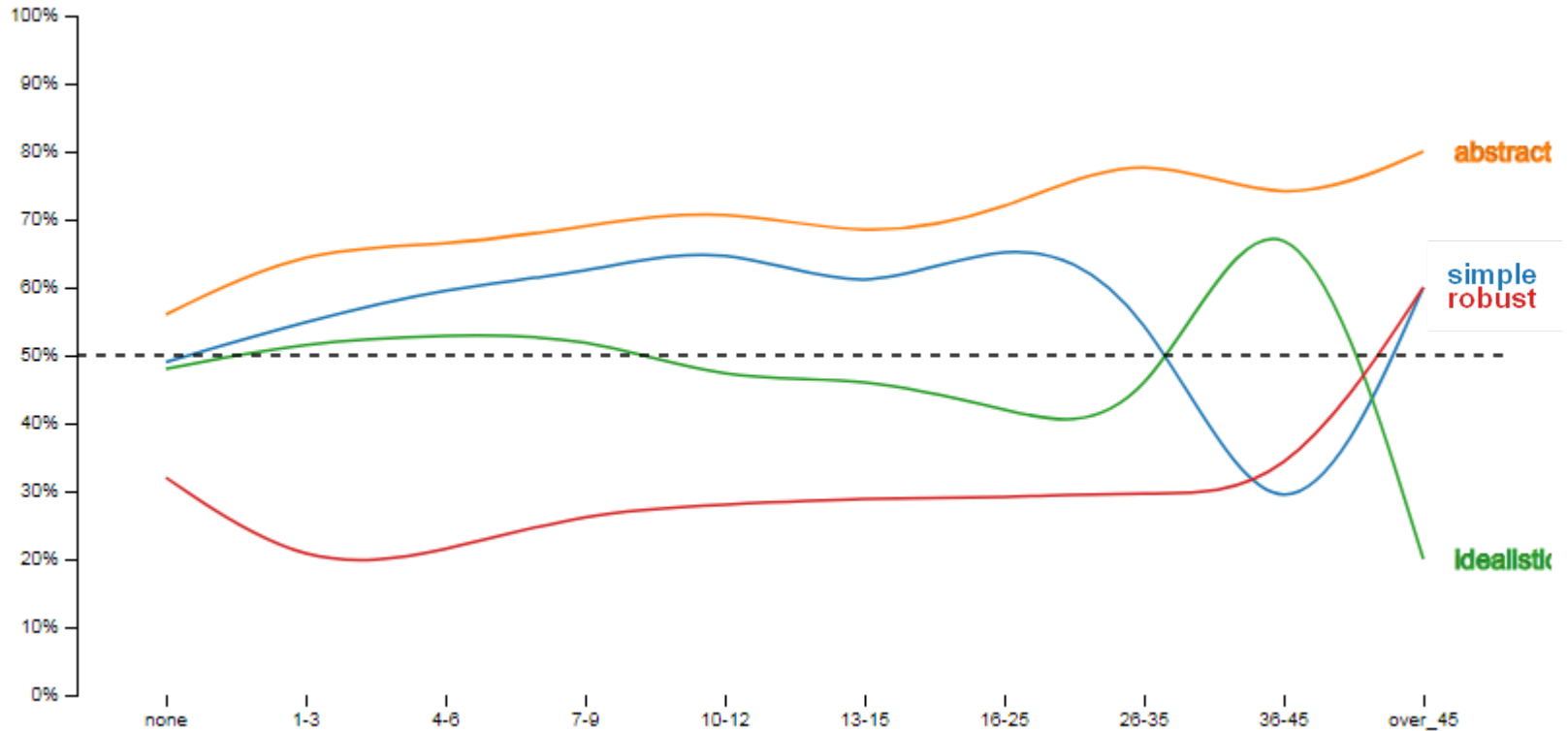
email@design-types.net



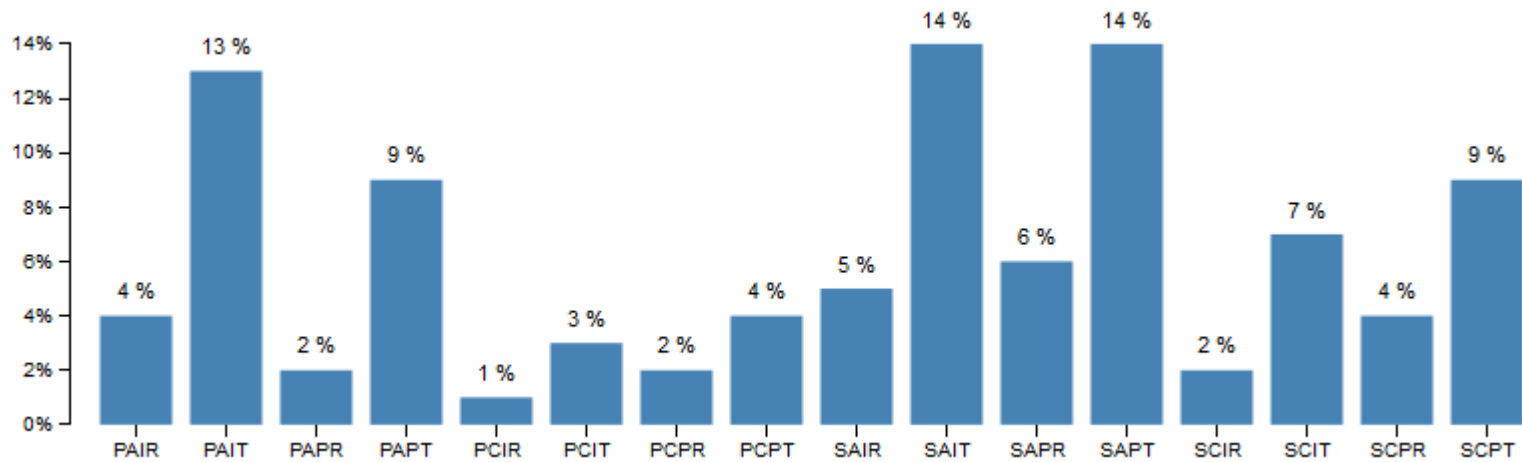
Matthias Wittum

Anhang

Statistiken – Veränderung durch Berufserfahrung



Statistiken – Verteilung der Typen



Statistiken – Häufigkeit der Antworten

