



Argumentativ zur besten Designentscheidung

<for developers only...maybe>



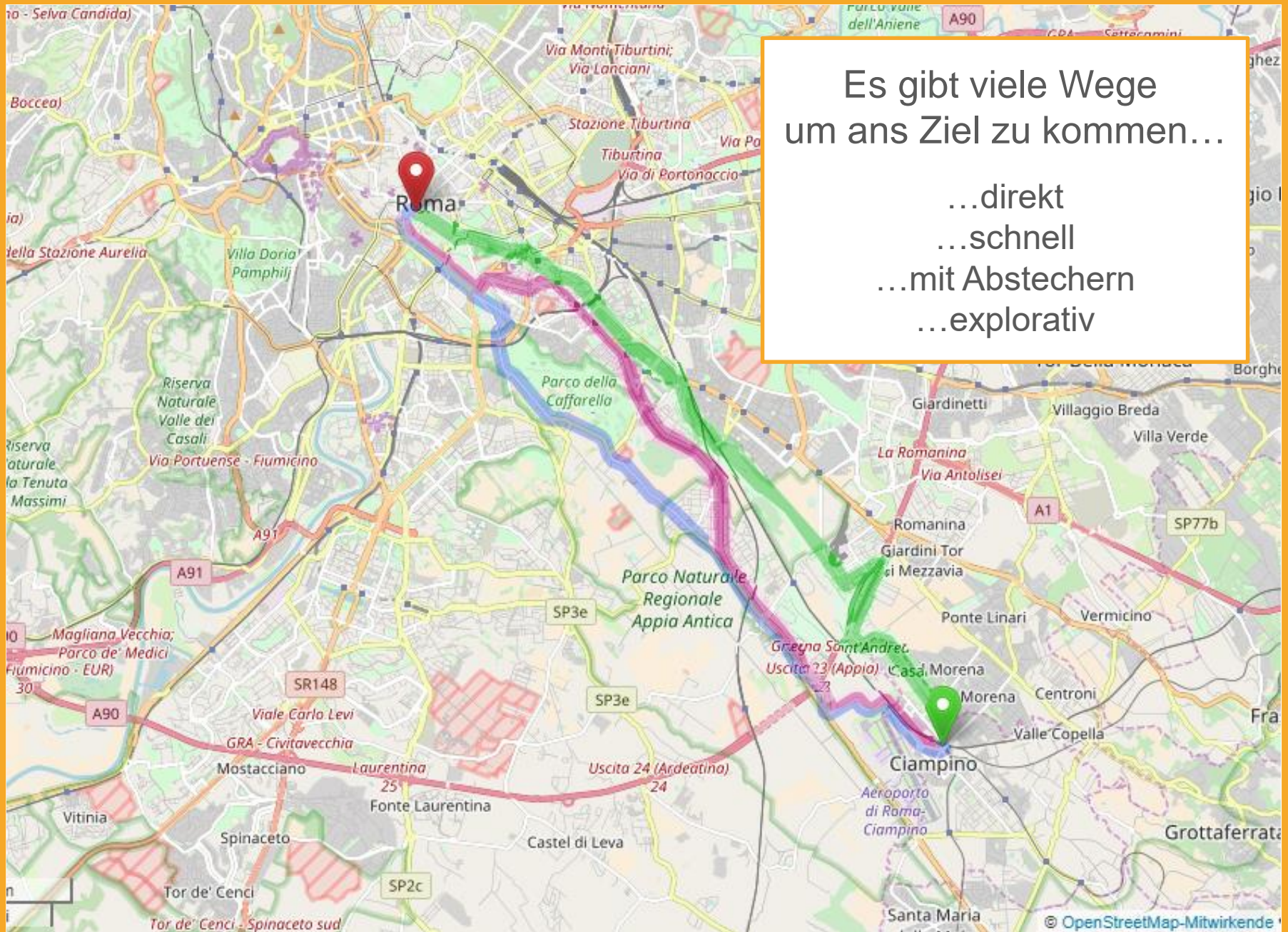
Christian Rehn:

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



Matthias Wittum:

- Head of 1&1 Source Center
- www.design-types.net



Es gibt viele Wege
um ans Ziel zu kommen...

- ...direkt
- ...schnell
- ...mit Abstechern
- ...explorativ

Szenario



vs.

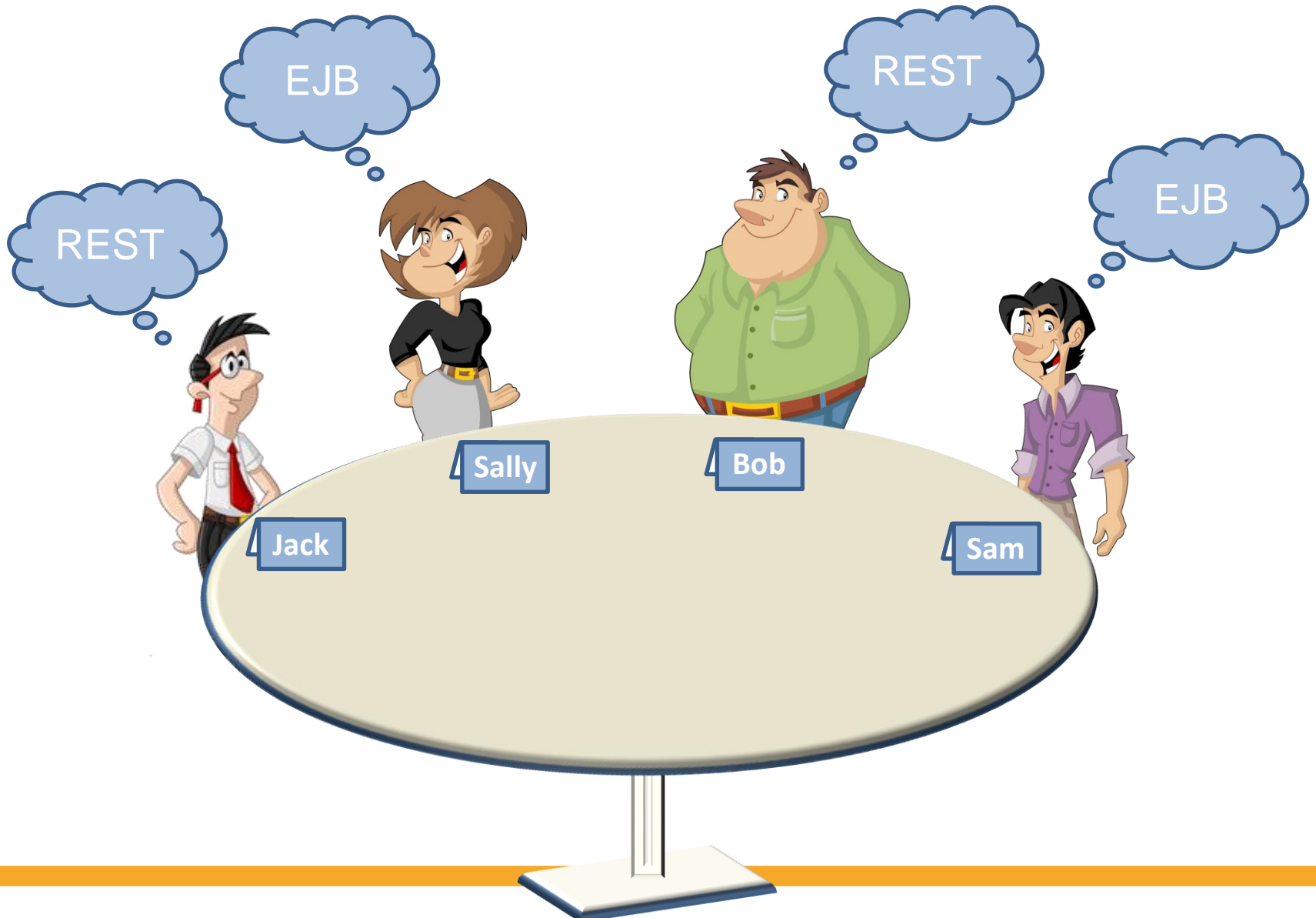
{ REST }

Ein völlig fiktives Szenario



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

Technische Diskussion: „die historische Variante“



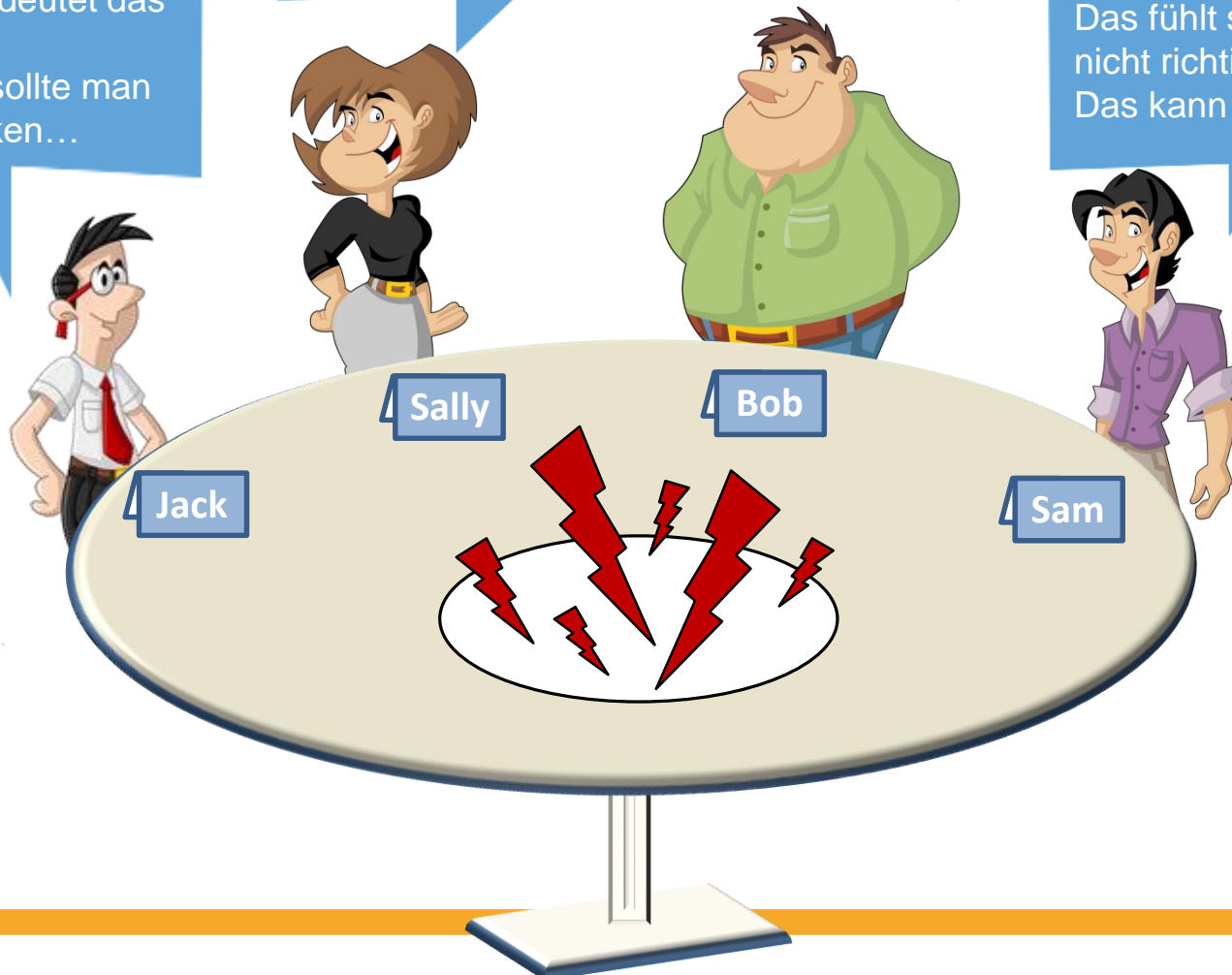
Technische Diskussion: „die historische Variante“

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..

IH/E

SDP

SoC

ML

KISS

ISP

SCP

DRY

SRP

LoD

OCP

FF

RoP

DIP

ECV

ZOI

MIMC

LC

UP

LLA

PSU

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

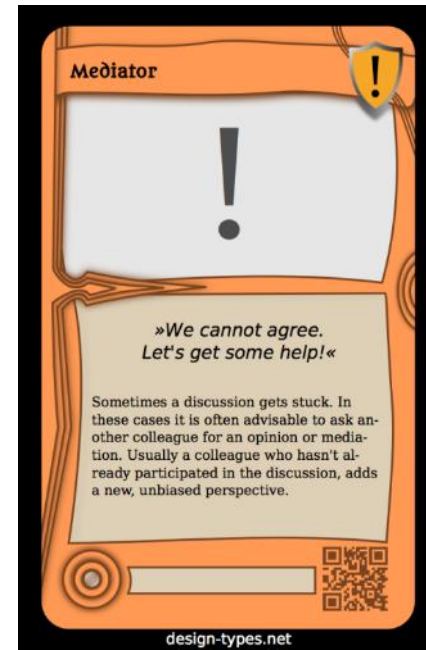
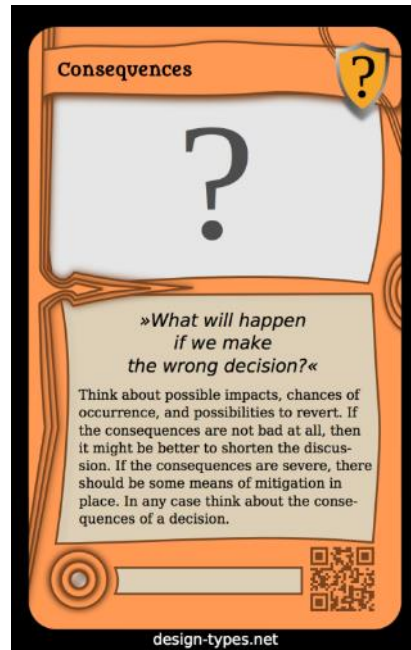
»Avoid possibilities for something to go wrong or to get misused.«

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

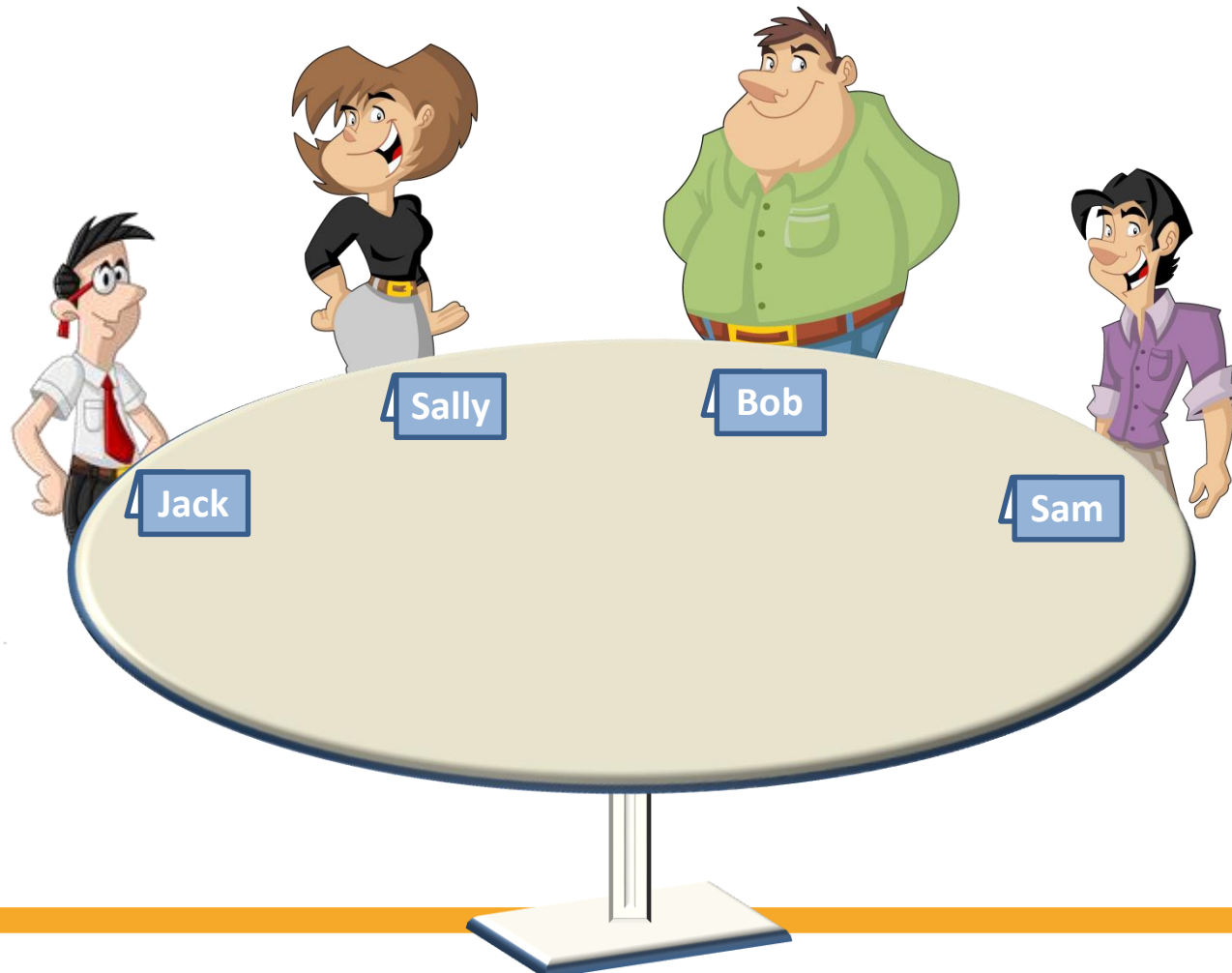
design-types.net

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

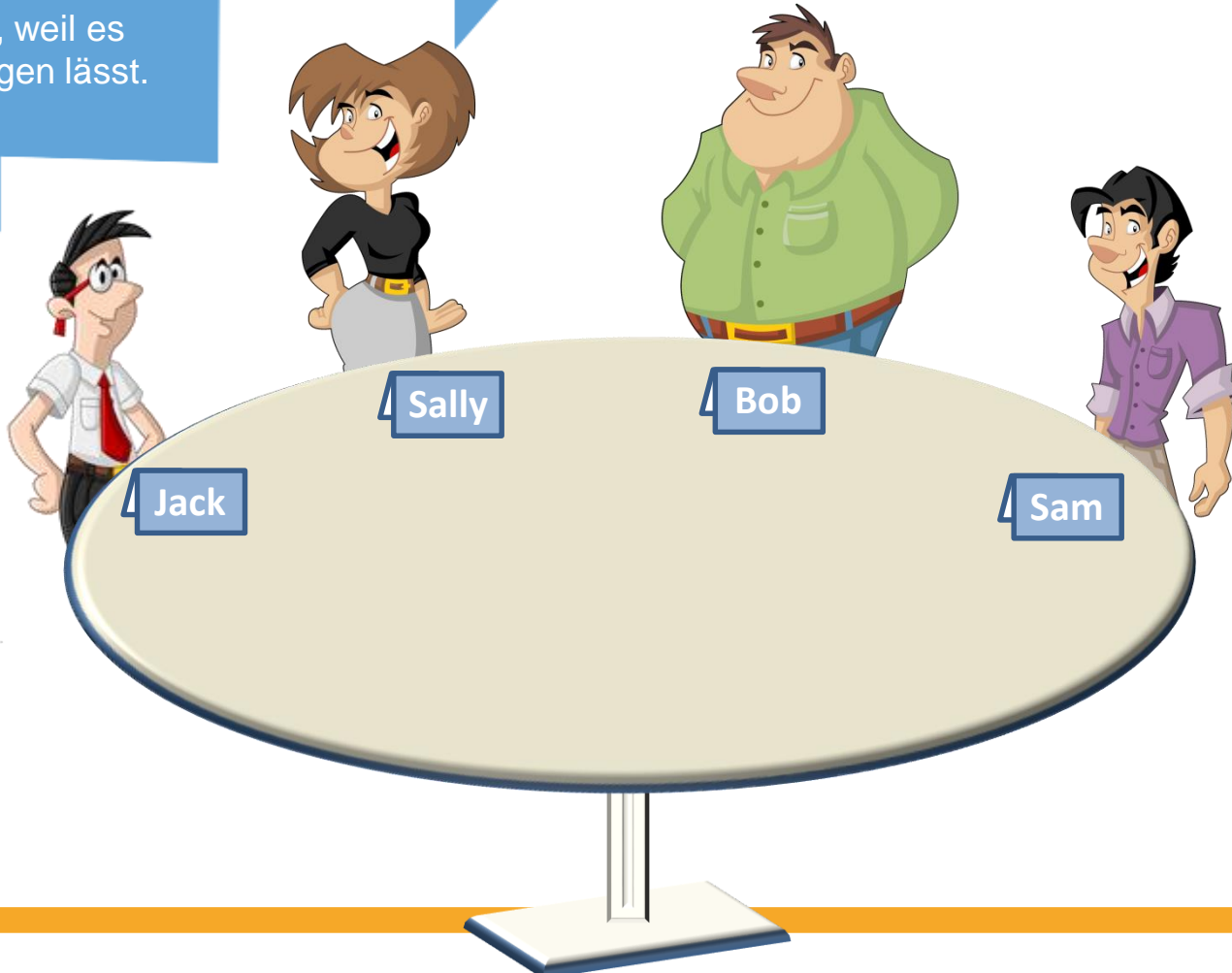
KISS

Jack

Sally

Bob

Sam

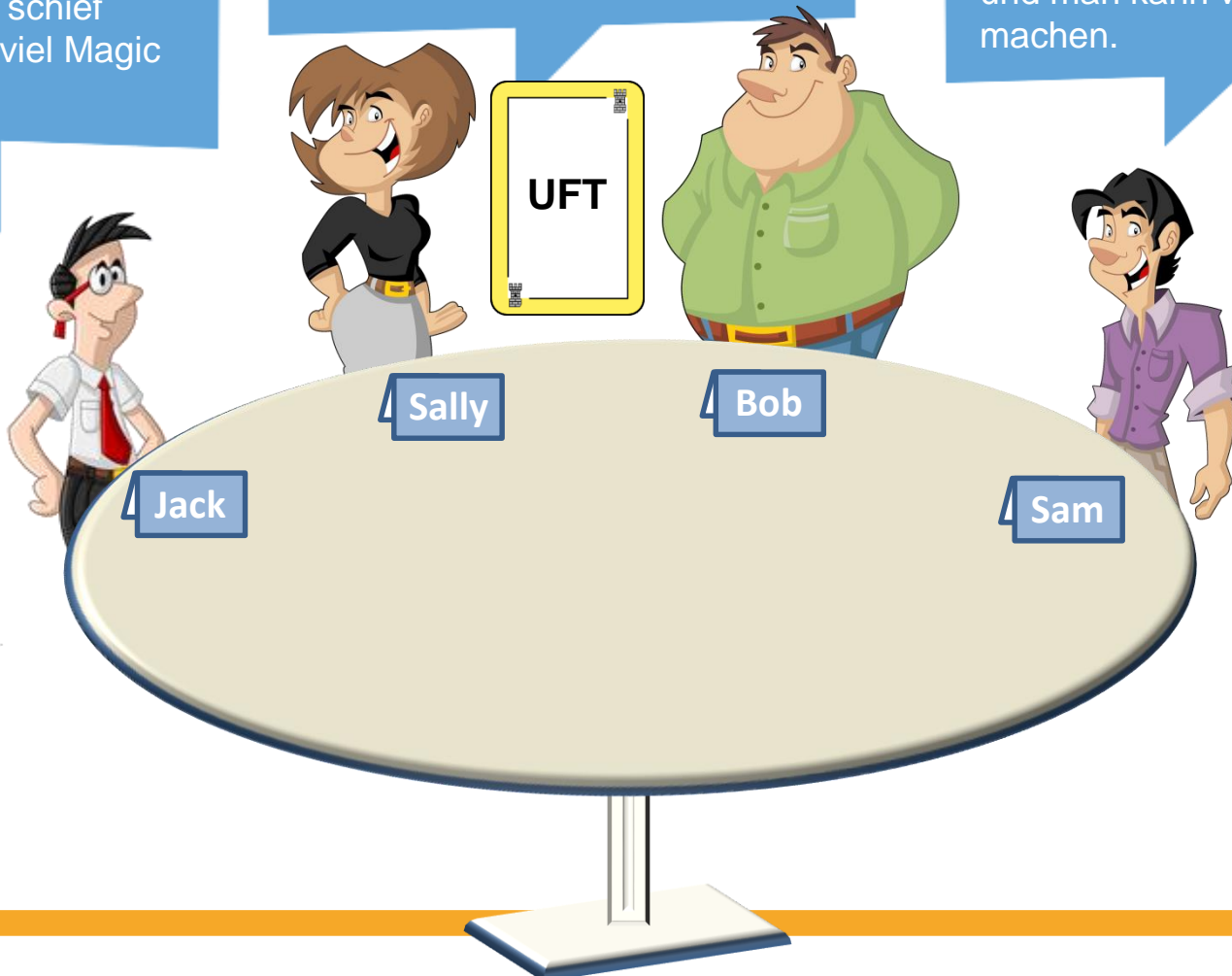
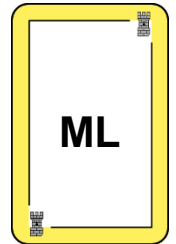
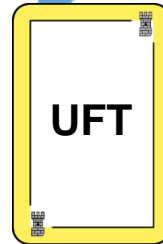
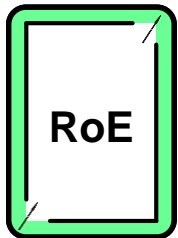


Technische Diskussion: „kartengestützte Argumentation“

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

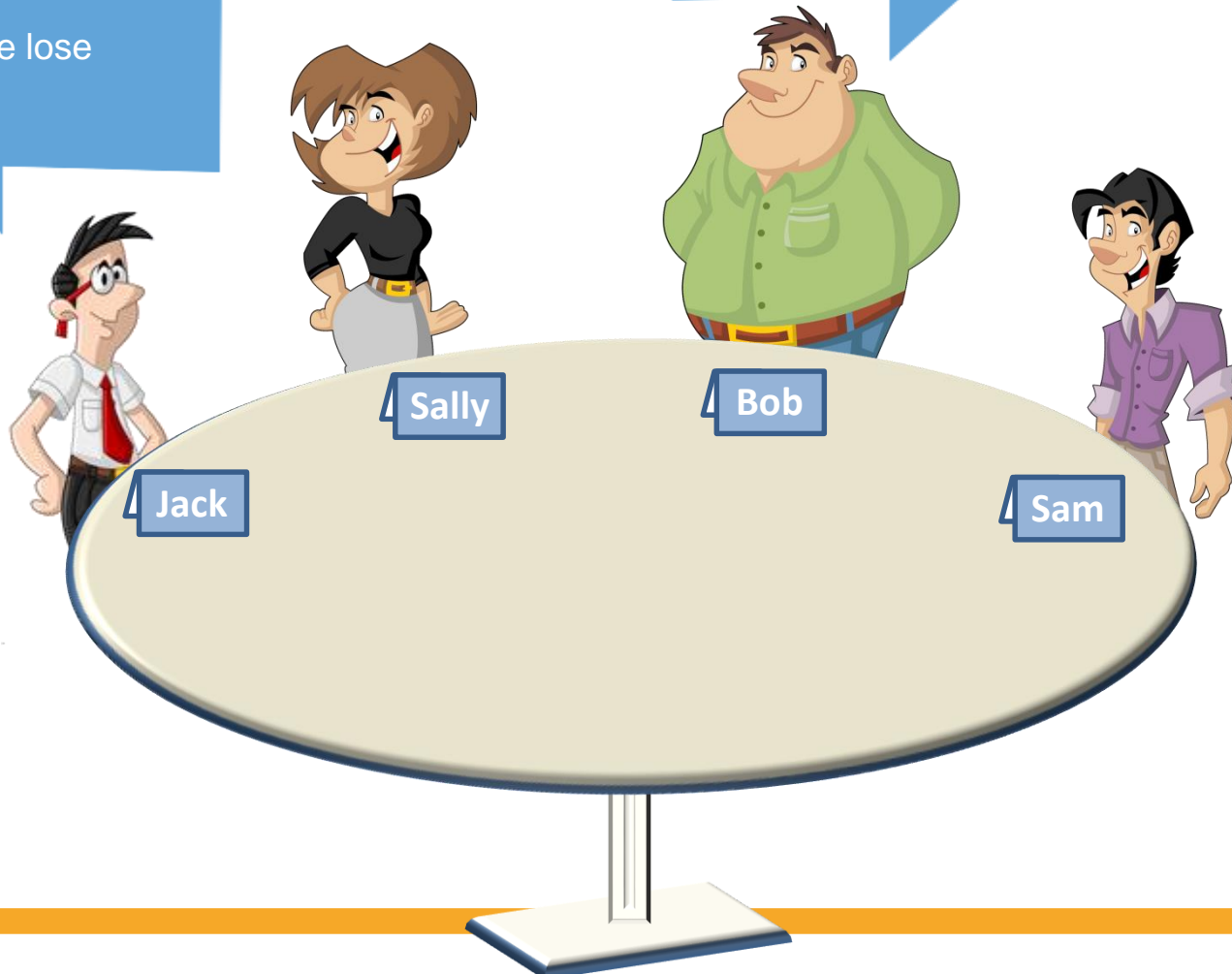
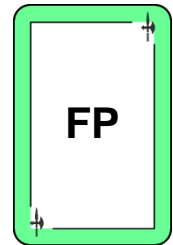
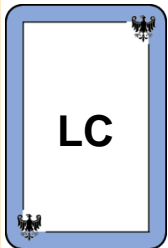
Bob

Sam

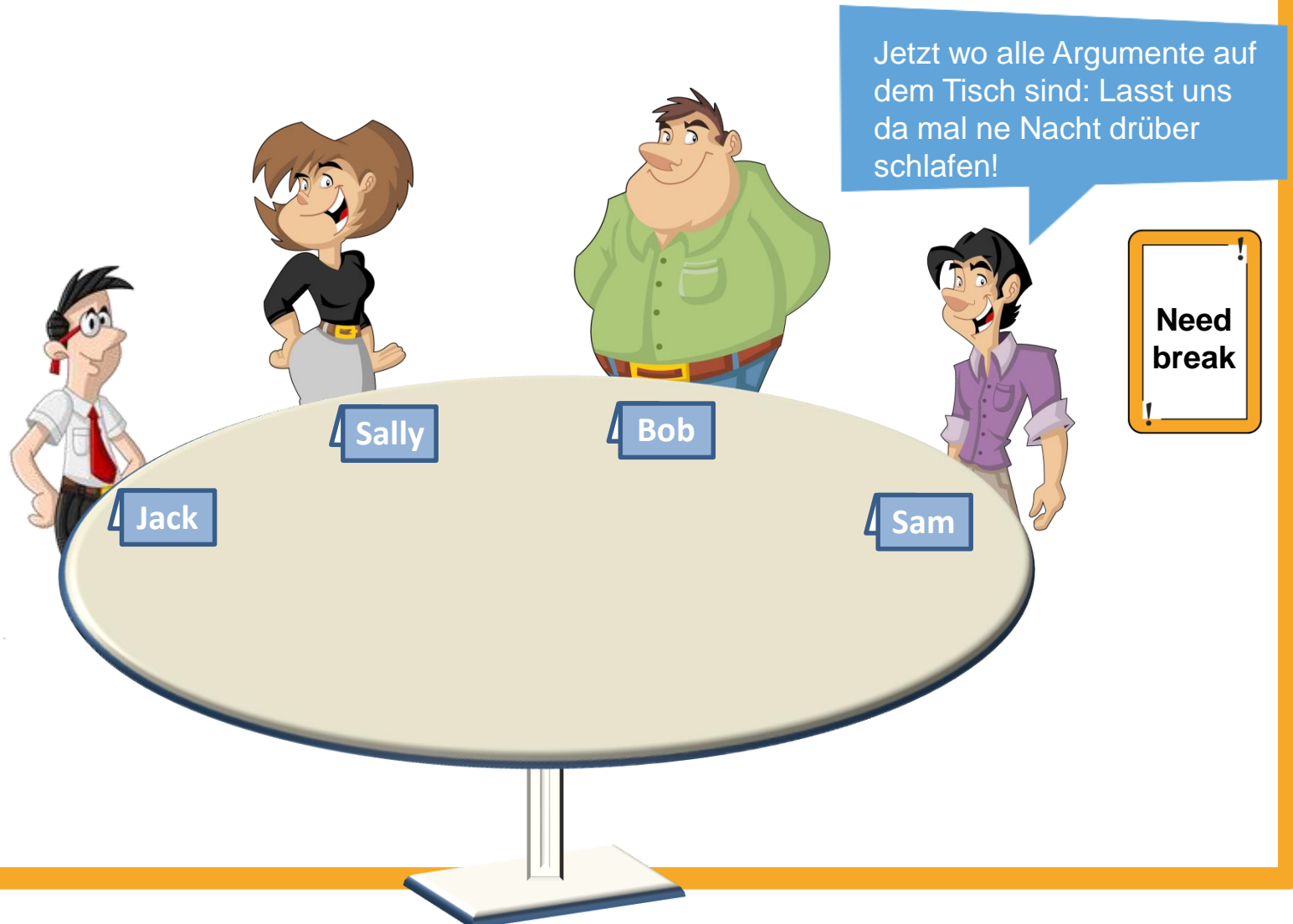
Technische Diskussion: „kartengestützte Argumentation“

Ein Grund ist die lose Kopplung.

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

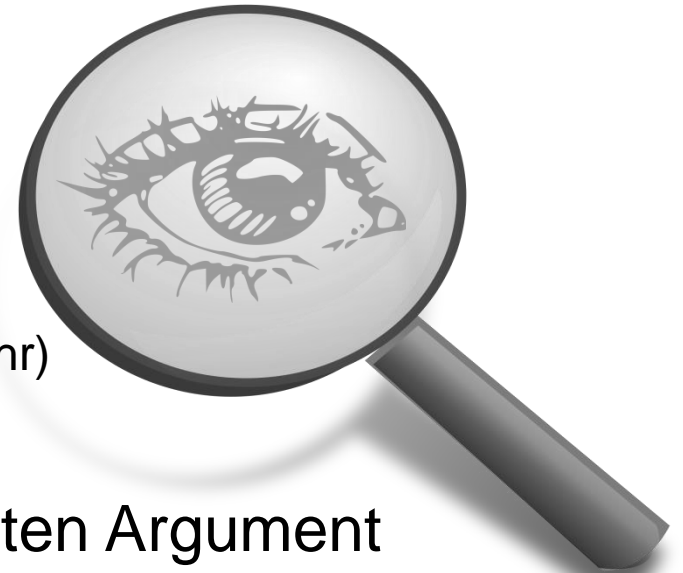


Technische Diskussion: „kartengestützte Argumentation“

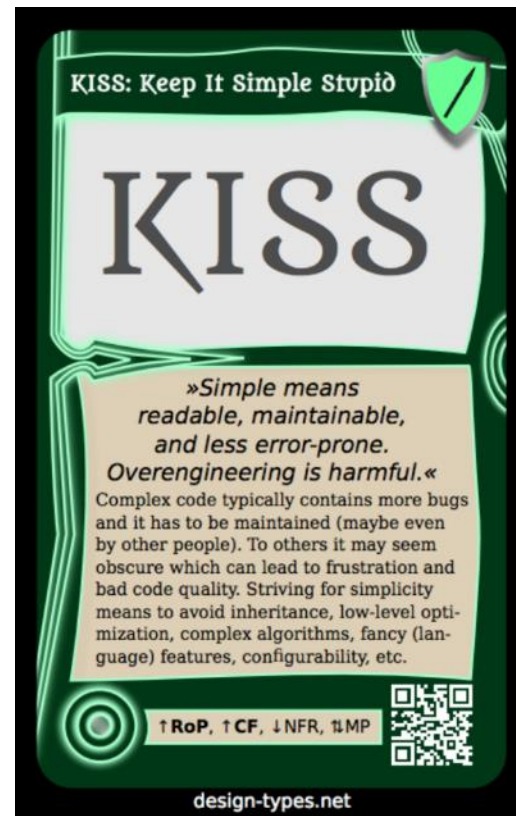


Erste Beobachtung

- Die Entwickler argumentieren klarer und nachvollziehbarer (Jack überrennt seine Kollegen nicht mehr)
- Ein Argument leitet zum nächsten Argument oder Gegenargument über



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:

- to keep code simple for better understandability
- to omit unnecessary things (lower risk; fewer bugs)
- to reduce complexity
- to prefer explicit solutions instead of implicit ones
- etc.

Powerful means:

- to build powerful and generalized solutions
- to have flexibility/extensibility by foresighted design
- to have configurable solutions
- to master complexity
- etc.

Abstract means:

- to think in concepts and abstractions
- to focus on the big picture and component interactions
- to know all potential consequences of a change
- to build models of the real world
- etc.

Concrete means:

- to think in code or simultaneously transfer ideas into code immediately
- to optimize algorithms for better performance
- to understand systems by reading the code
- etc.

Pragmatic means:

- to fulfill requirements asap
- to focus on customer needs to guarantee a value
- to omit unnecessary things
- to bring others down to earth
- etc.

Idealistic means:

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

Robust means:

- to protect applications against risks and potential bugs
- to define and adhere to standards
- to avoid too much magic and complexity to reduce risks
- to use proven solutions which stood the test of time
- etc.

Technologic means:

- to use new, modern and more productive technologies and to get rid of legacy
- to evolve with technology to be more competitive
- to broaden your personal horizon
- etc.

S

C

I

R

Selbst ausprobieren: www.design-types.net

Design-Types.net Design Types ▾ Design Matrix Design Cards About

principles-wiki.net

How Do You Design Software?



Examine proposed technical solutions from all perspectives.

Design Matrix



Learn why software design is individual and often leads to discussions with colleagues.

Design Types



Improve technical discussions by using proven arguments.

Design Cards

Test yourself & Assess colleagues

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.

-  **Simple** : This means you prefer straight-forward solutions
-  **Abstract** : This means you always have the big picture in mind
-  **Pragmatic** : This means you like things done fast
-  **Robust** : This means you strive for stable and robust software

Principles you probably like

KISS, MIMC, RoE, LC, HC

Principles you rather disregard

GP, PSU, TdA/IE

Strengths


- Fast in delivering stable and working solutions.
- Code and design are normally easy to understand.


Suggestions


- Keep your technical knowledge up to avoid building too much on your own existing library could do.
- Don't get left behind by evolving technologies.
- Keep your design flexible and extendable. Be prepared for continuous requirement changes.
- Don't forget the trade-offs you make when increasing development speed.




Your Design Type: The Construction Manager (SAPR)

 **Simple**
This means you prefer simple, straight-forward 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

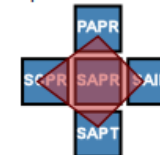
Programming is to you:

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

Dimension overlap

Simple	Powerful
Abstract	Concrete
Pragmatic	Idealistic
Robust	Technological

Type overlap



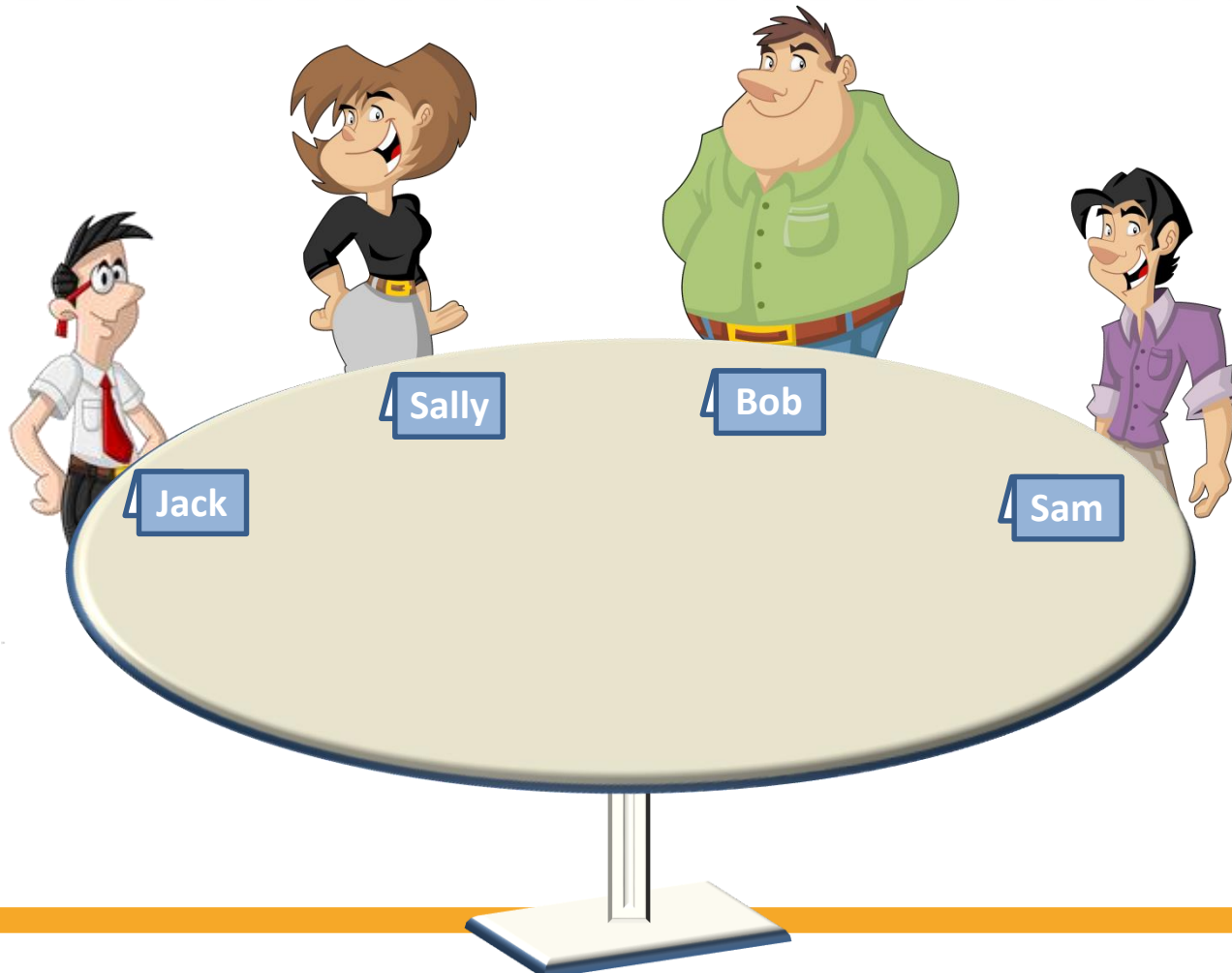
design-types.net



Like it? Print it!

Technische Diskussion: „Diskutieren mit Kontext“

Der nächste Tag...



Technische Diskussion: „Diskutiert“

Simple
Powerful
Pragmatic

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

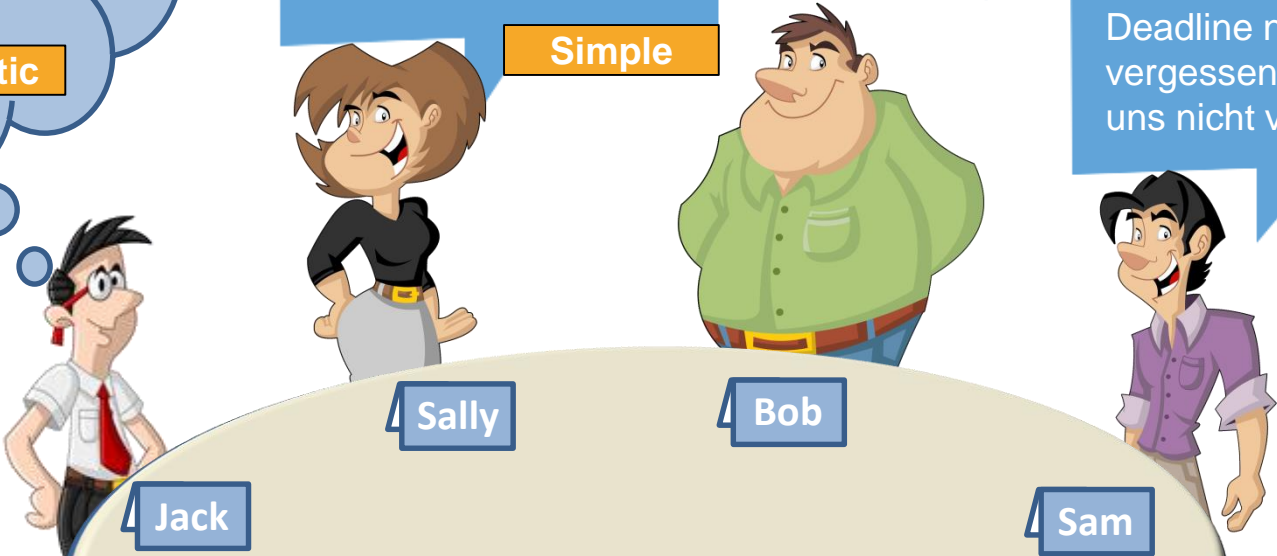
Simple

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"/> Status: <input type="checkbox"/> Approved <input type="checkbox"/> Rejected
Topic overview & Solution details:	<input type="text"/> <small>If useful, link relevant documents</small>	Decided by:	<input type="text"/>
		Stakeholder:	<input type="text"/>
		Summary:	<input 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="text"/> notes	<input type="text"/> 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? 	Powerful
	<div style="border: 2px dashed red; padding: 10px; color: red; font-weight: bold;"> <p>Ziel: Vorbereitete Design-Entscheidungen aus verschiedenen Perspektiven beleuchten.</p> </div>				
Abstract	<ul style="list-style-type: none"> <input type="checkbox"/> on its own? <input type="checkbox"/> Are modules cohesive and is coupling low? 	<input type="text"/>	<input type="text"/>	<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)? 	<input type="text"/> notes	<input type="text"/> notes	
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? 	<input type="text"/> notes	<input type="text"/> 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?

Stand der Dinge



- Design Types
 - Fertig
 - > 2600 Teilnehmer bisher

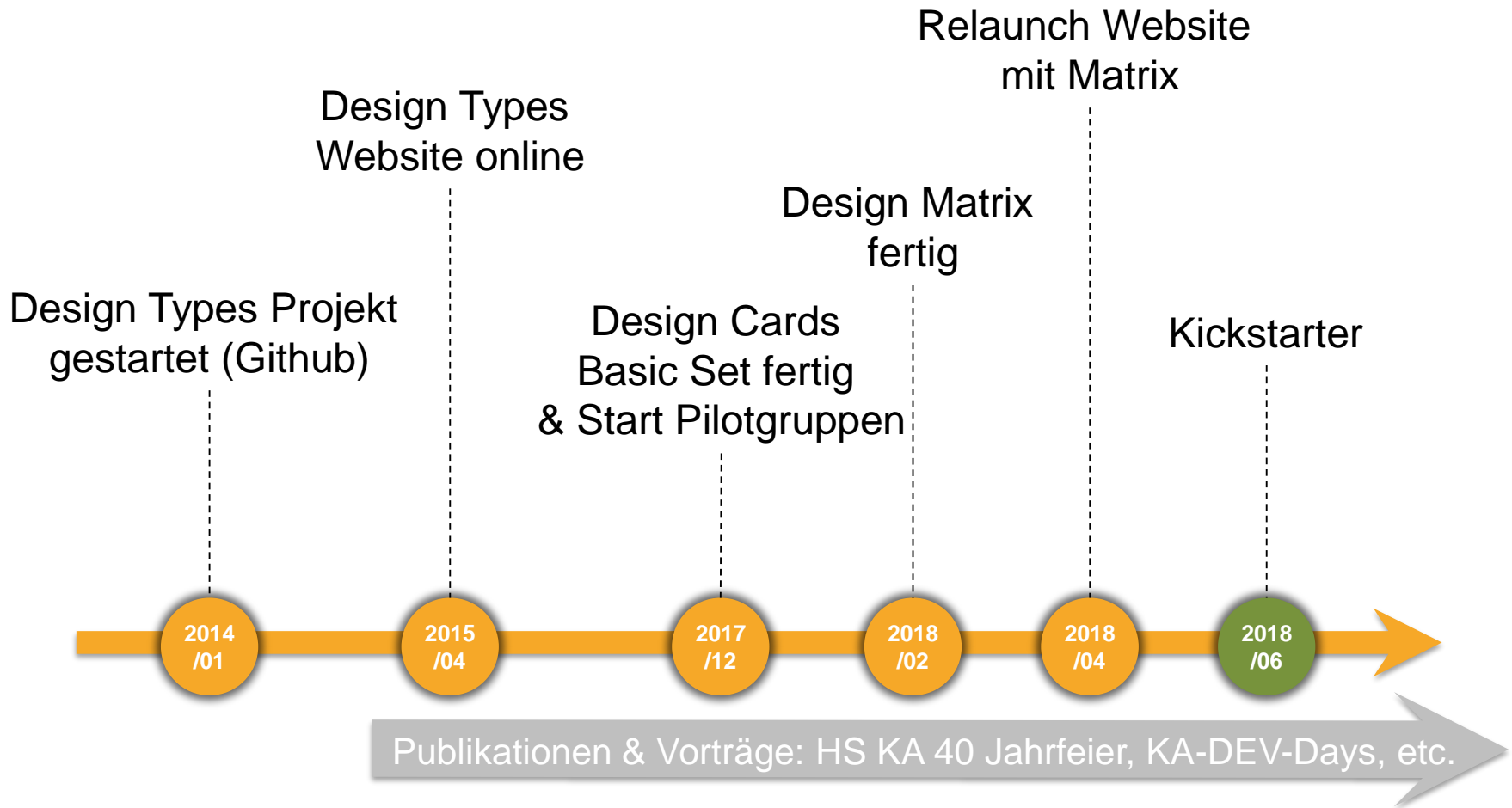


- Design Matrix
 - Fertig
 - Aktuell Feedback durch Pilotgruppen
 - Verfügbar als Download

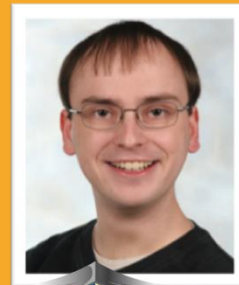


- Design Cards
 - 26/54 Karten fertig (Basic Set)
 - Online-Karten gerade im Entstehen
 - Aktuell Feedback durch Pilotgruppen

Timeline – was passiert(e)?



Thank you for your interest...
...the „Software Design Knights“



Any
Questions?

Contact

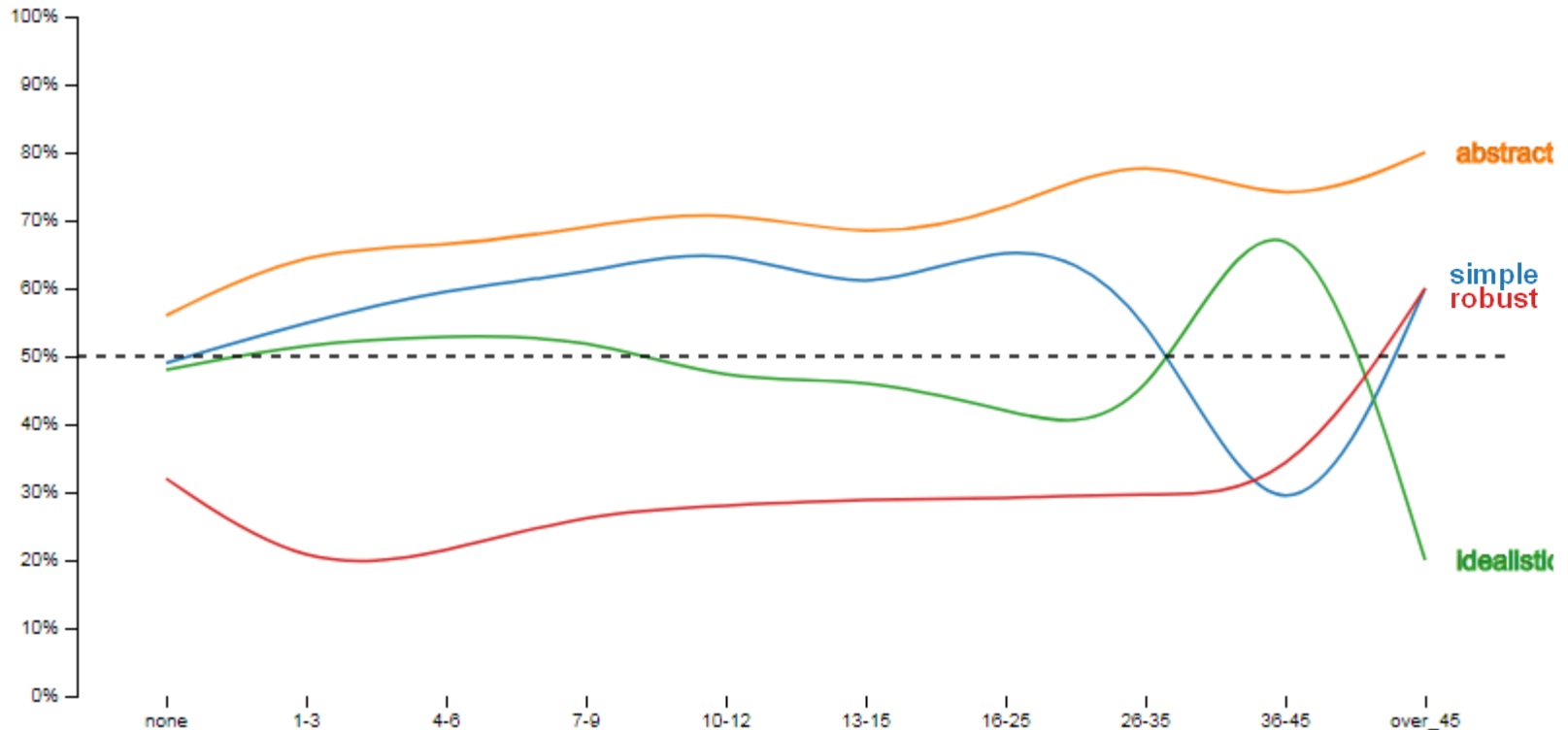
Web: www.design-types.net

Mail: email@design-types.net

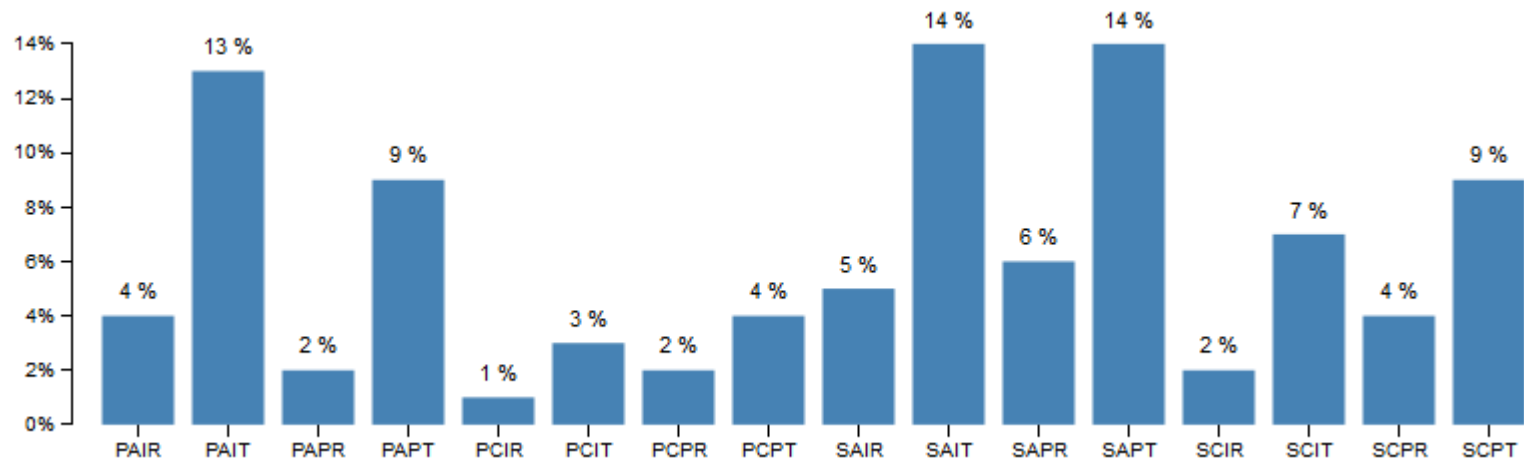
Twitter: [@SWDesignKnights](https://twitter.com/SWDesignKnights)

Anhang

Statistiken – Veränderung durch Berufserfahrung



Statistiken – Verteilung der Typen



Statistiken – Häufigkeit der Antworten

