

15 LESSONS FROM 15 YEARS AS A SOFTWARE ARCHITECT

Ingo Rammer
@ingorammer

Co-Founder & CEO at thinktecture

Just my personal experiences, not
the ultimate sage advise!

- 1994 – Web apps with CGI, Perl, Apache, ...
- 1996 – Windows Apps (VB et. al)
- 1999 – Java backends, Servlets, XML, SOAP, ...
- 2001 – .NET → consulting (mainly server side)
- 2005 – Client side becomes interesting again
- 2009 – Phones (iOS, Android, BB, WP7, ...)
- 2010 – HTML5 for Applications

Last ten years: consultant
to software architects

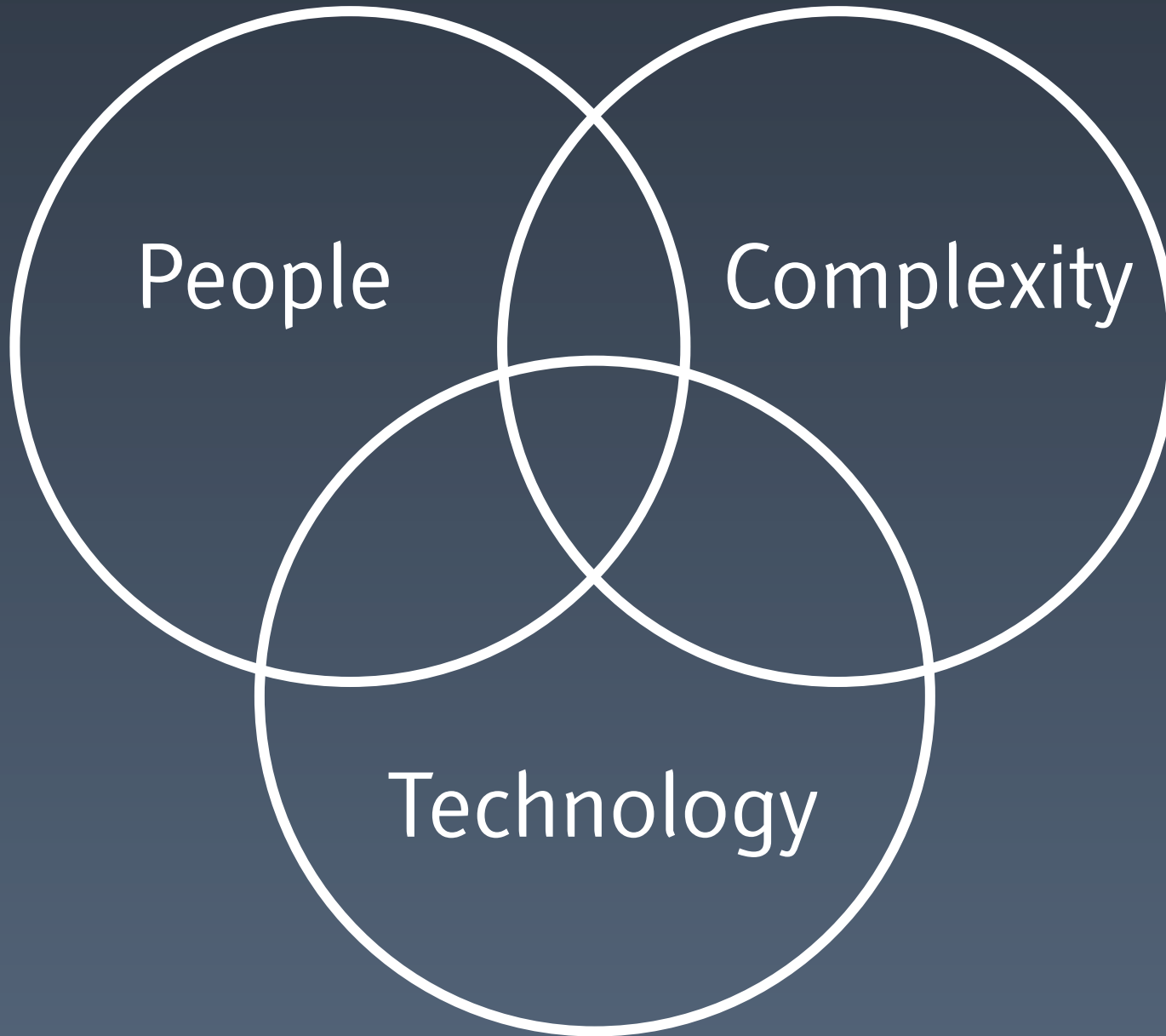
Lessons I've learnt ...

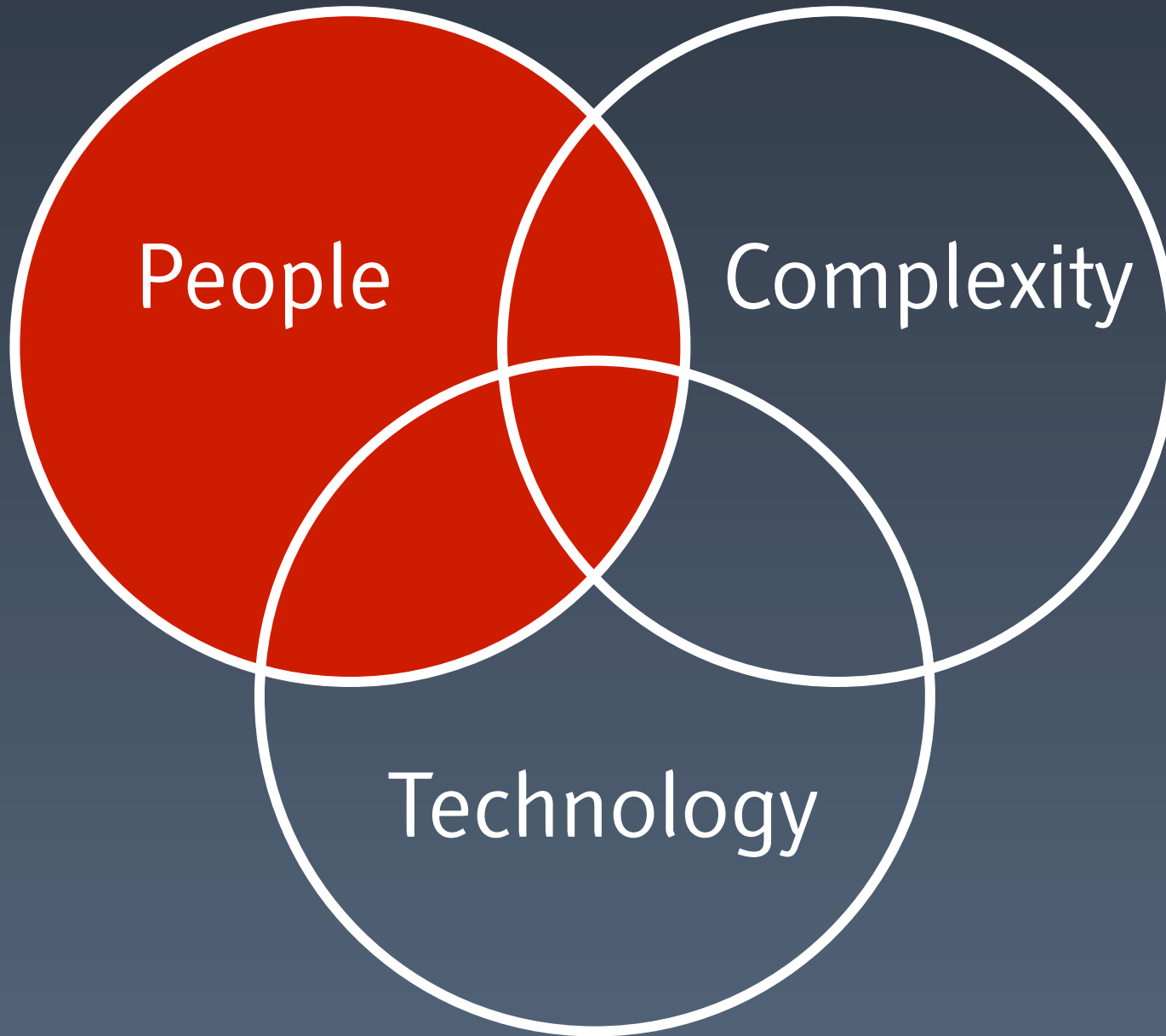
Why do projects succeed/fail?

People?

Technology?

... ?







#1 – Don't trust experts!

(Speakers, authors, including myself)



They don't know your context



They will be excited about their topic



What could be right for 98% of attendees at a conference, could be devastating for your particular project



#2 – People affect architecture



Where your team comes from (prior experiences, skill levels, shared way of thinking, ...) will influence the suitability of certain architectures



#3 - Good for *me* or for *the project?*



Newest stuff vs. mature and proven



Does my current project *really* need the
tech advantage?



Or is it just that / will need to use the
newest tech to stay current?



#4 – Research vs. Development



When we *just don't know the answer*, we need to make sure that our customers, employers, and colleagues understand that we are only researching



And: We need to make sure that *we*
understand this as well



Even six months later!
Critically (really!) revisit findings!



#5 – Be wary of *The Second System*



First system (with new tech, new approach, new processes): we're always very careful



Second System: we *know it all*. We might re-use the things which have worked and do a 180° turn on the things which didn't.



Reality: either the requirements might be different (and the approaches won't work for that reason), or there could be a middle ground instead of the 180°



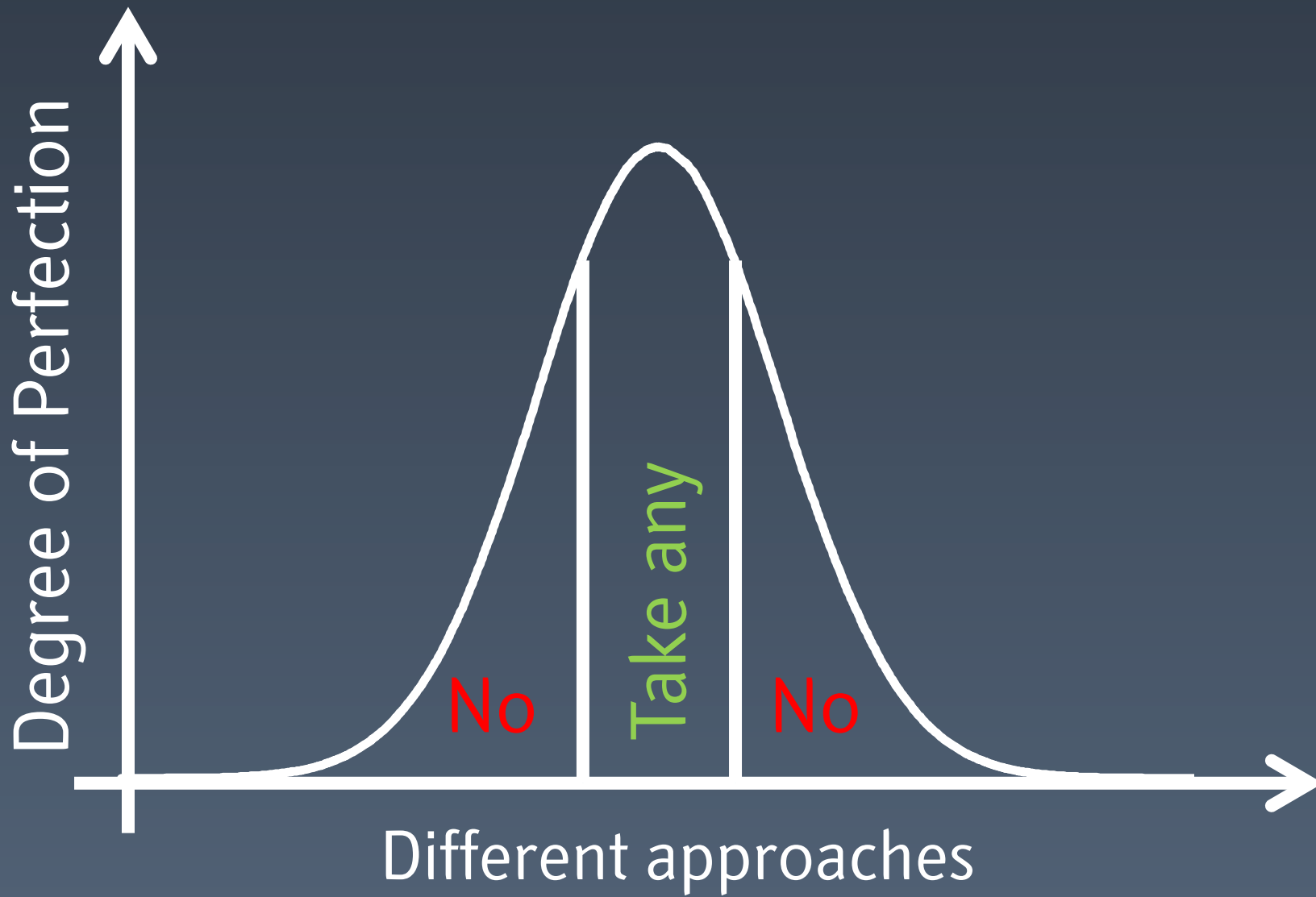
#6 – Some things need to
be discussed, others just
need to be done



Sometimes two, three or four different ways
can get a project closer to the target.



Finding or waiting for the *perfect* way might
take longer than all negative effects of
choosing any of the others





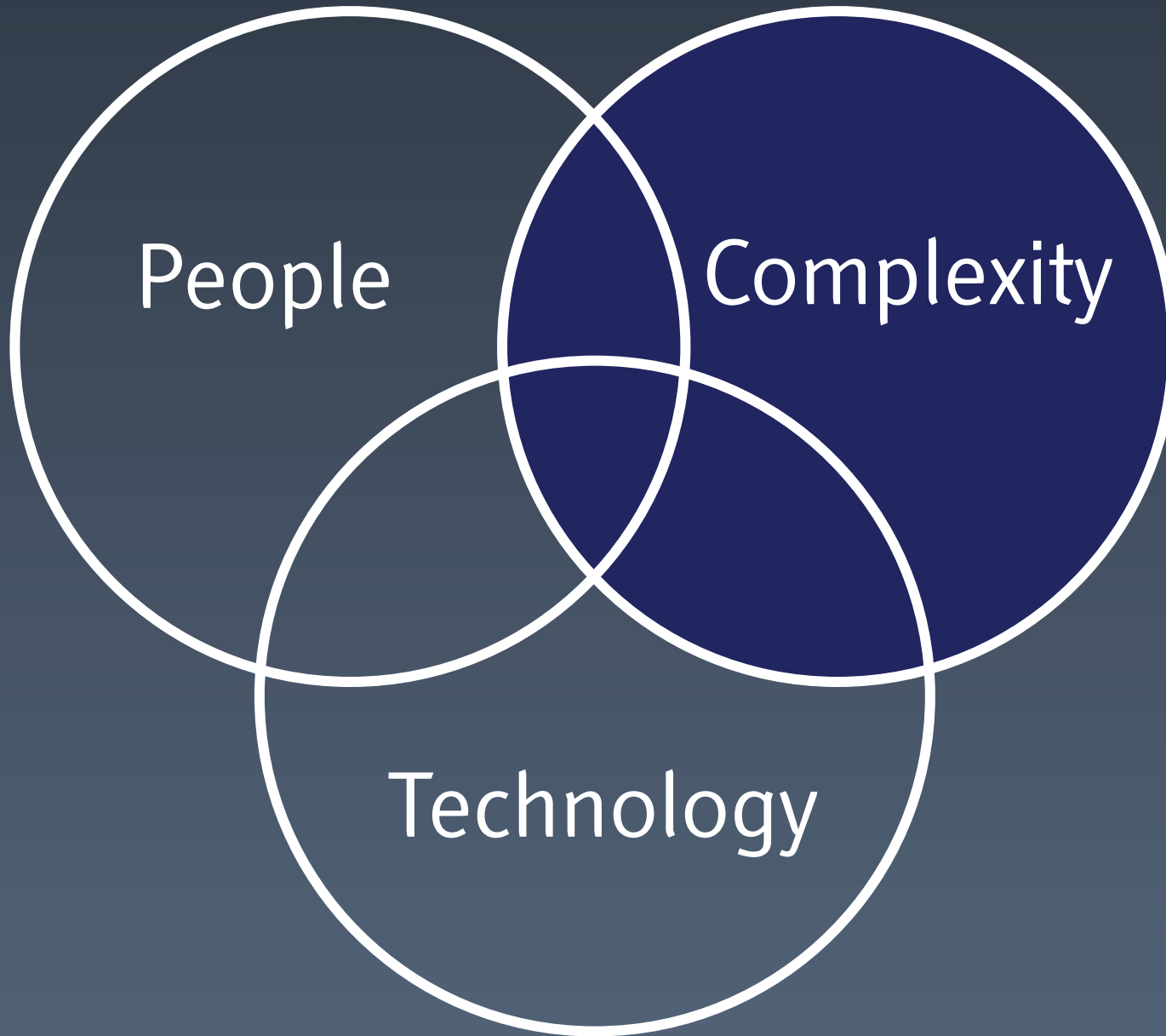
#7 – Build what people pay
us to build



If creating [*business software*] is boring for us,
we need to change to a different customer/
employer/project, but not *artificially* inflate
complexity to make it challenging enough to
be worth our while



(Otherwise we're wasting money and talent)





#8 – Always observe
problem complexity vs.
solution complexity



If you need to write your own O/R Mapper,
DI-Framework, MVC Engine and database for
a business application you might be
missing something



#9 – Make it simpler



If you haven't taken time to make it simpler, it's
quite likely too complex



In 15 years I haven't seen a single project fail because of ***lack of complexity*** but I've seen (literally!) double-digit millions of EURs lost to *too much complexity*



*Can also happen throughout a project: Review
architecture and code! Religiously.*



#9.a – If the solution somebody
advocates appears too complex,
it quite likely is



And, btw, it might be *you* who's advocating it to
your team members



If you're the only one who can describe end-to-end processes in your application, it's too complex.



Change roles: let your architecture be explained to you by your team in one-on-one's. Does it still look the same?



#10 – Most of us don't need
Ebay/Amazon/Google/Bing-Scale



A lot of scalability can be achieved quite cheaply today. It's just the last 2% which are hard and expensive



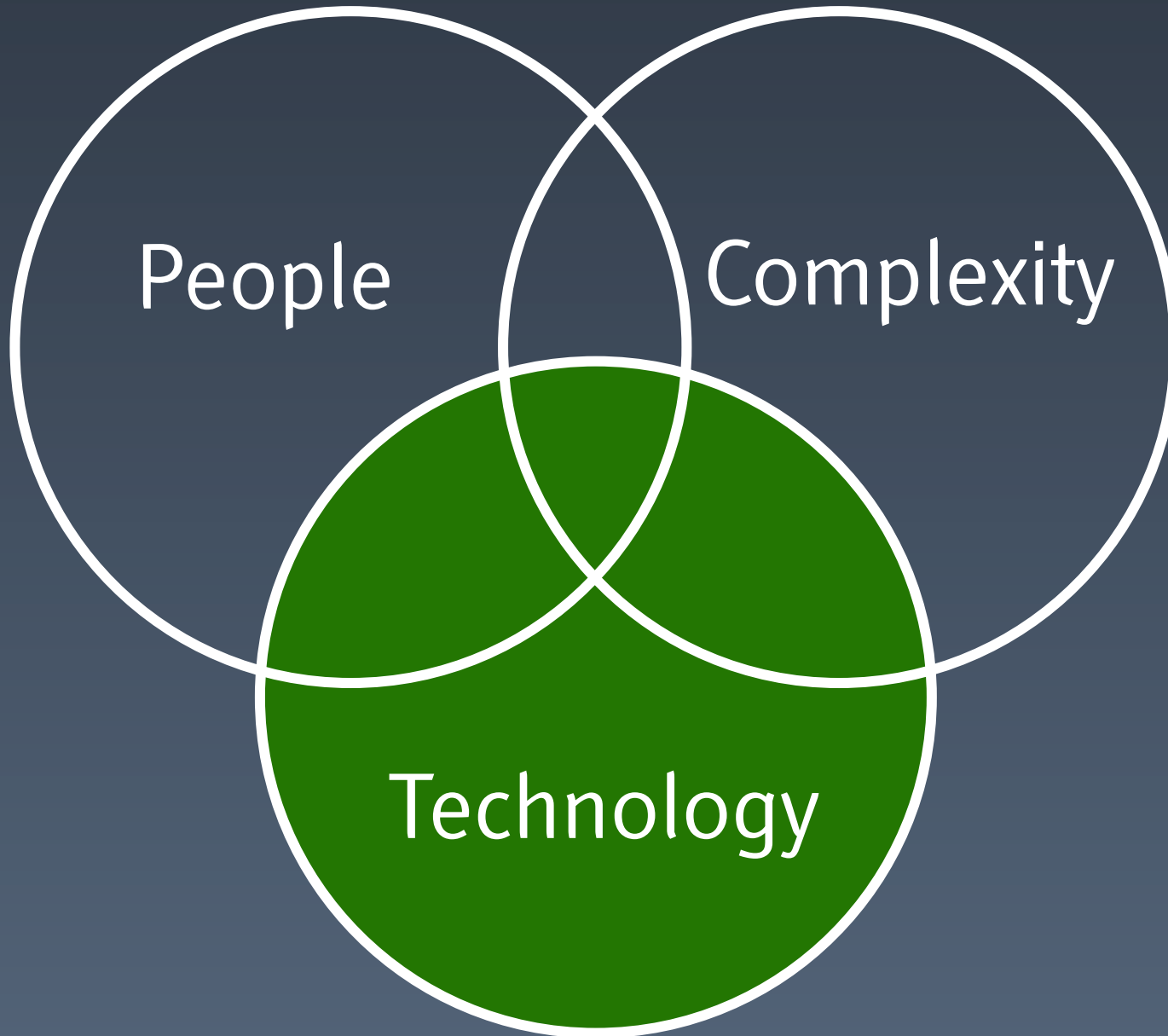
Fortunately most projects don't
ever need to go there



Even if you do: it's usually better to first find out what your market really wants and LATER re-engineer when successful



Google, Ebay, Amazon, and most others
did it this way, too





#11 – Code is written to be read



Sometimes code has to be really smart



Most of the time it has to be readable



Because *you* might not be around five
years later



Maintainability = Understandability
+ Discoverability
+ Consistency
+ Cohesion
- Coupling



#12 – Don't talk about solutions
before understanding the
problem



It's very hard to not fall into the solution-looking-for-a-problem trap

"We could really do this using CQRS, ..."



Especially in the few weeks after you've read or heard about a new idea/pattern/framework/approach.



Yes, this means: no changes to architecture within
four weeks after a conference ;-)



#13 – When in doubt, pick the
technology you *know*



... instead of taking the newest and *Best Thing Ever* coming from the vendor



Why? it might not be ready, yet. So the question is: is the risk/reward profile positive enough?



#14 – There is no silver bullet



On all levels, we're promised solutions:

Quality: TDD, Unit Testing, ...

Data: Autosharding, NoSQL, BigTable, ...

Dependencies: IoC, DI, EBC, ...

Responsibility: CQRS, ...

Do they deliver? Are they worth the risks?



#15 – There is no good idea
which can't be used in a totally
wrong way



Quality: TDD, Unit Testing, ...

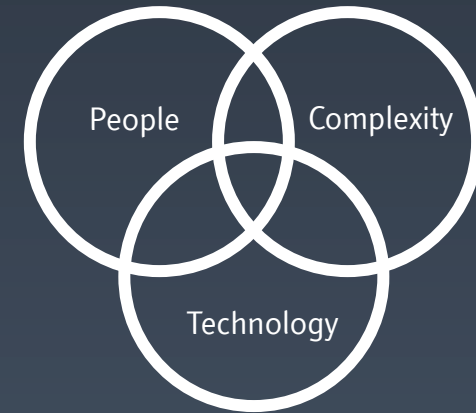
Data: Autosharding, NoSQL, BigTable, ...

Dependencies: IoC, DI, EBC, ...

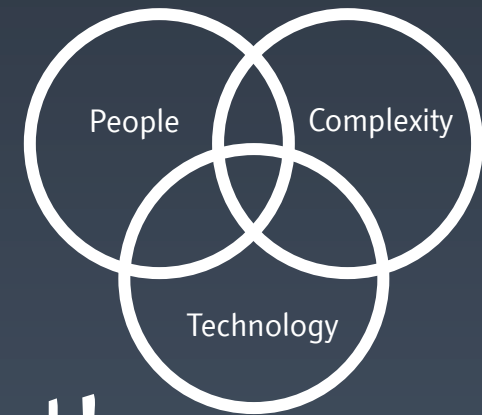
Data Responsibility: CQRS

But even classics: O/R Mapping, Event-
Based Decoupling, ...

The Top #15



- #1 – Don't follow others!*
- #2 – People affect architecture*
- #3 - Good for me or for the project?*
- #4 – Research vs. Development*
- #5 – Be wary of The Second System*
- #6 – Some things need to be discussed, others just need to be done*
- #7 – Build what people pay you to build*
- #8 – Always observe problem complexity vs. solution complexity*
- #9 – Make it simpler*
- #9.a – If the solution appears too complex, it quite likely is*
- #10 – Most of us don't need Ebay/Amazon/Google/Bing-Scale*
- #11 – Code is written to be read*
- #12 – Don't think about solutions before understanding the problem*
- #13 – When in doubt, pick the technology you know*
- #14 – There is no silver bullet*
- #15 – There is no good idea which can't be used in a totally wrong way*
- Bonus – Shipping is a feature!*



Be pragmatic and honest!

Simplicity wins!

Don't let every hype get you!

... and talk to the business people

Contact

ingo.rammer@thinktecture.com

Weblog

<http://weblogs.thinktecture.com/ingo>