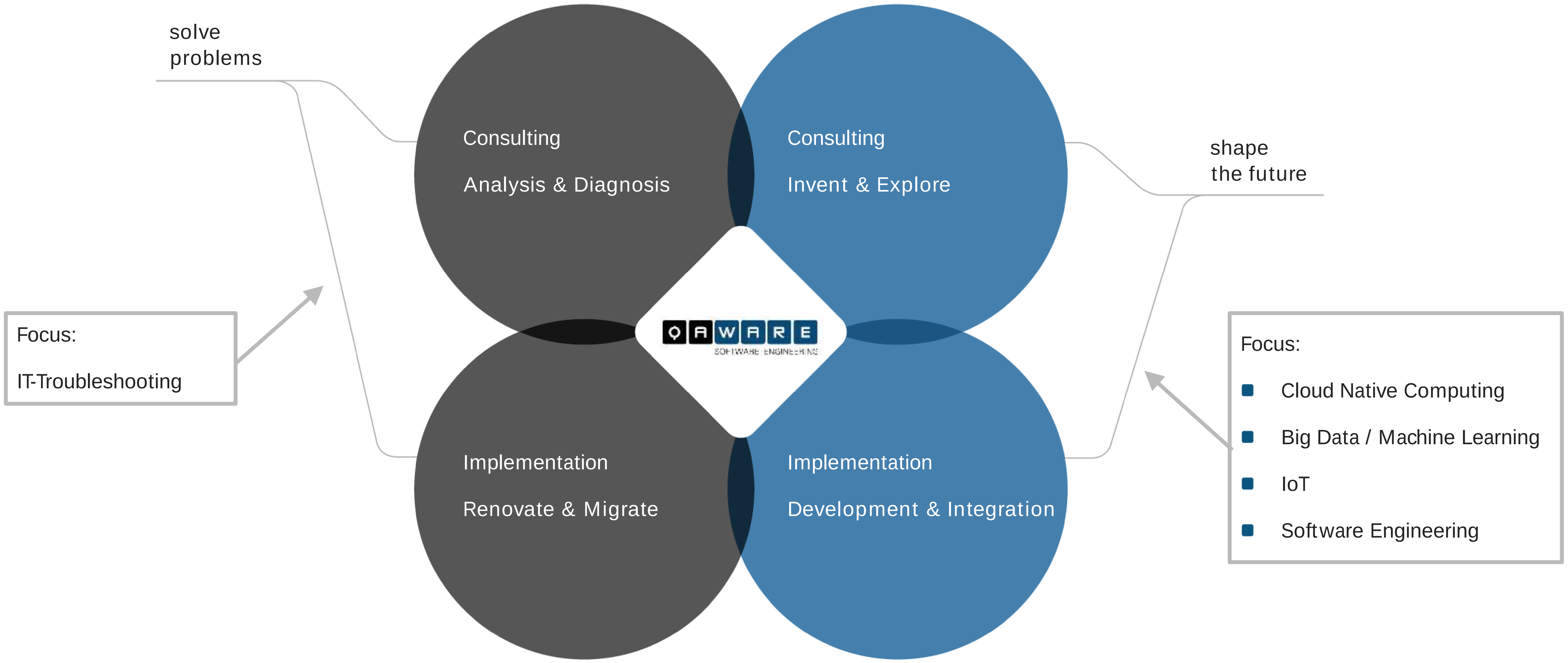
Agile Projects in Waterfall Surroundings— Challenges and Lessons Learned

Marcus Ciolkowski

marcus.ciolkowski@qaware.de

Innsbruck, 01 December 2017

QAware: We specialize in quality projects.



Focus:

IT-Troubleshooting

Focus:

- Cloud Native Computing
- Big Data / Machine Learning
- IoT
- Software Engineering

Experiences gained from long-running large agile IT development projects

agile fixed-price contract
(annually / quarterly)

development over
several years

program size
> 150 persons

~~AGILE~~

context / interface partner
not agile

peak team size > 25

A trend has become commodity: Everyone is agile now

All of our projects are agile.
Because we do Scrum.
This will increase our speed-to-market, business value, and project success rate.
And our customer satisfaction.
OK, maybe at higher cost.

Illustration: kbeis – gettyimages.de

A trend has become commodity: Everyone is agile now

Also, I don't need to plan, because I can change everything at any time!

And I am so modern!

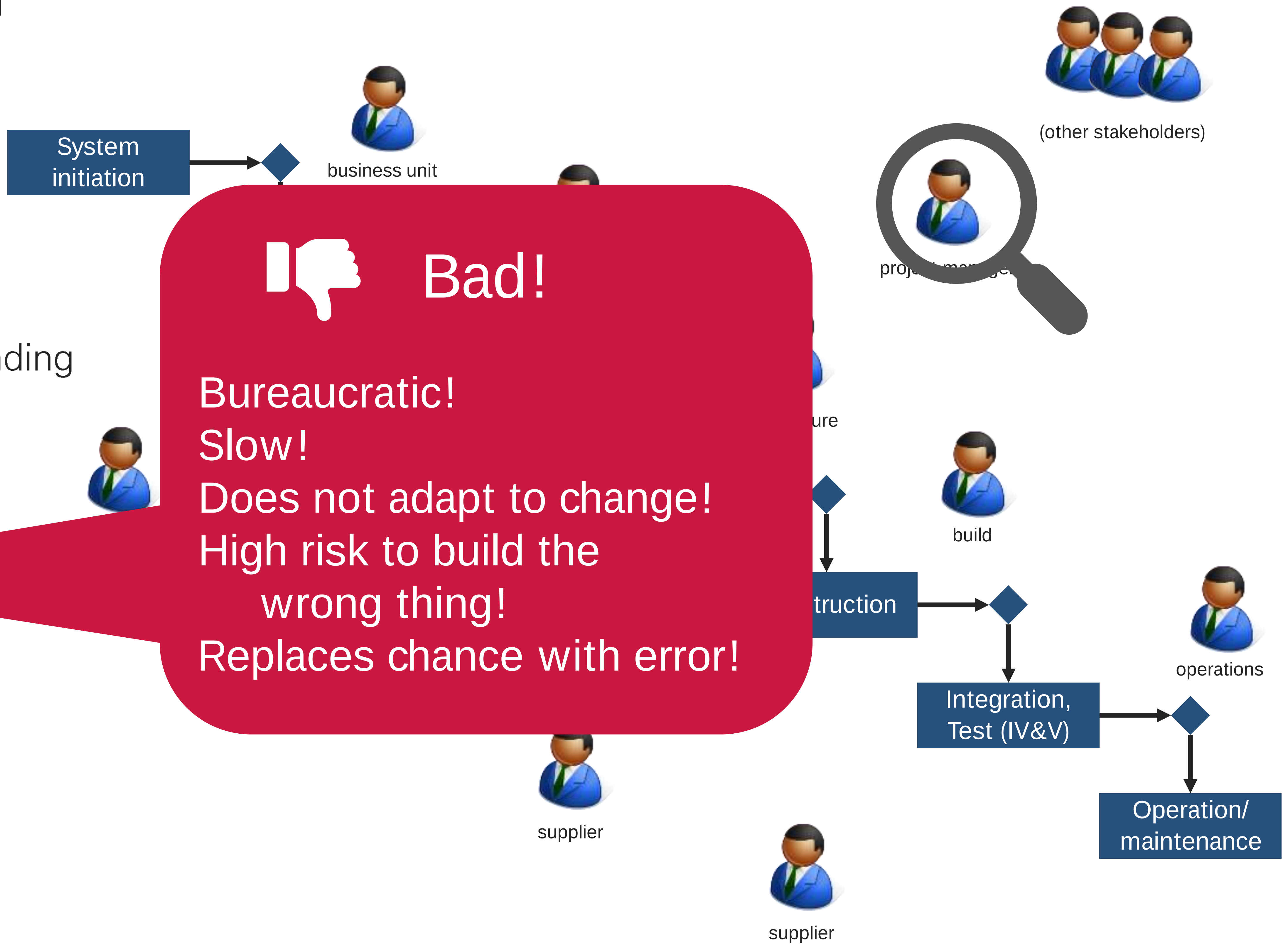
Great!

Illustration: kbeis – gettyimages.de

Bad Guy: Waterfall

- Big Design Up Front
- strict sequence of phases
- involving many stakeholders
- (… and it is historic misunderstanding as well as confusion)

Illustration: kbeis – gettyimages.de



Bad!

Bureaucratic!
Slow!
Does not adapt to change!
High risk to build the wrong thing!
Replaces chance with error!

But wait… not all is bad.

I love waterfall!

I know what I get, when I get it, and what I pay.

I control the project.

- I can plan and make commitments to my superiors.

Also, I have long time to think and plan for dependencies

There are clear gateways, roles and responsibilities

Illustration: 3Dmask – gettyimages.de

But wait… not all is bad.

I have a contract with Fine Print:

- guaranteed scope and quality
- guaranteed schedule
- warranty

I know who to blame for problems

OK, there are CRs, and we discuss about Bug vs. CR

Illustration: 3Dmask – gettyimages.de

But wait… not all is bad.

… but if anything fails, it's everyone's fault!

(Because everyone signs off gateways)

Illustration: 3Dmask – gettyimages.de

And now I have to do agile
I hate agile!

I know what I pay, but not what I get!

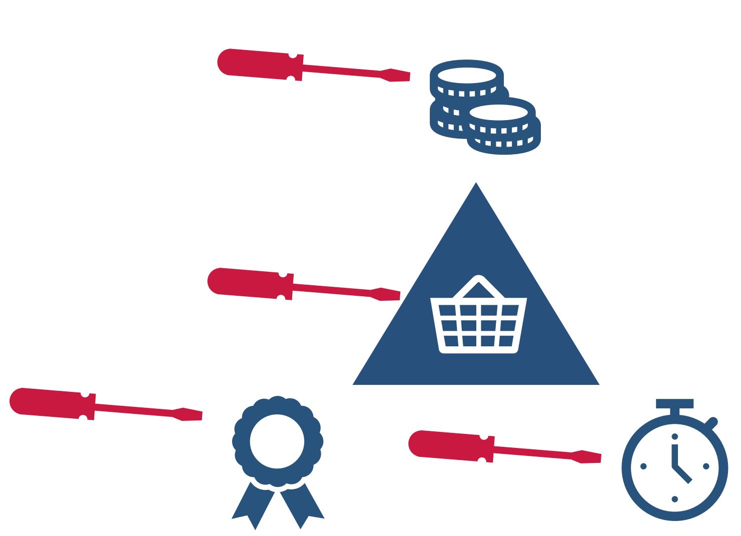
It’s too fast!
I have no control!

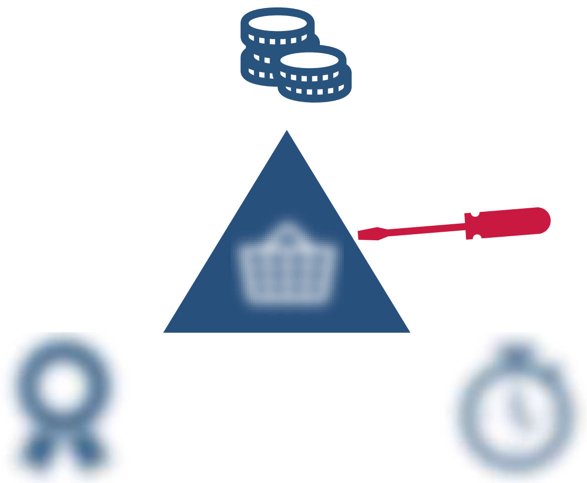
I don’t understand dependencies,

But I am responsible!

Illustration: 3Dmask – gettyimages.de

The agile challenge: Combine the best of both worlds.





plan-driven / contract work	agile
guaranteed quality & scope	fixed budget
ability to commit & plan	continuous planning
warranty	speed
everything is documented	start early react quickly
ability to plan dependencies	include feedback build it right

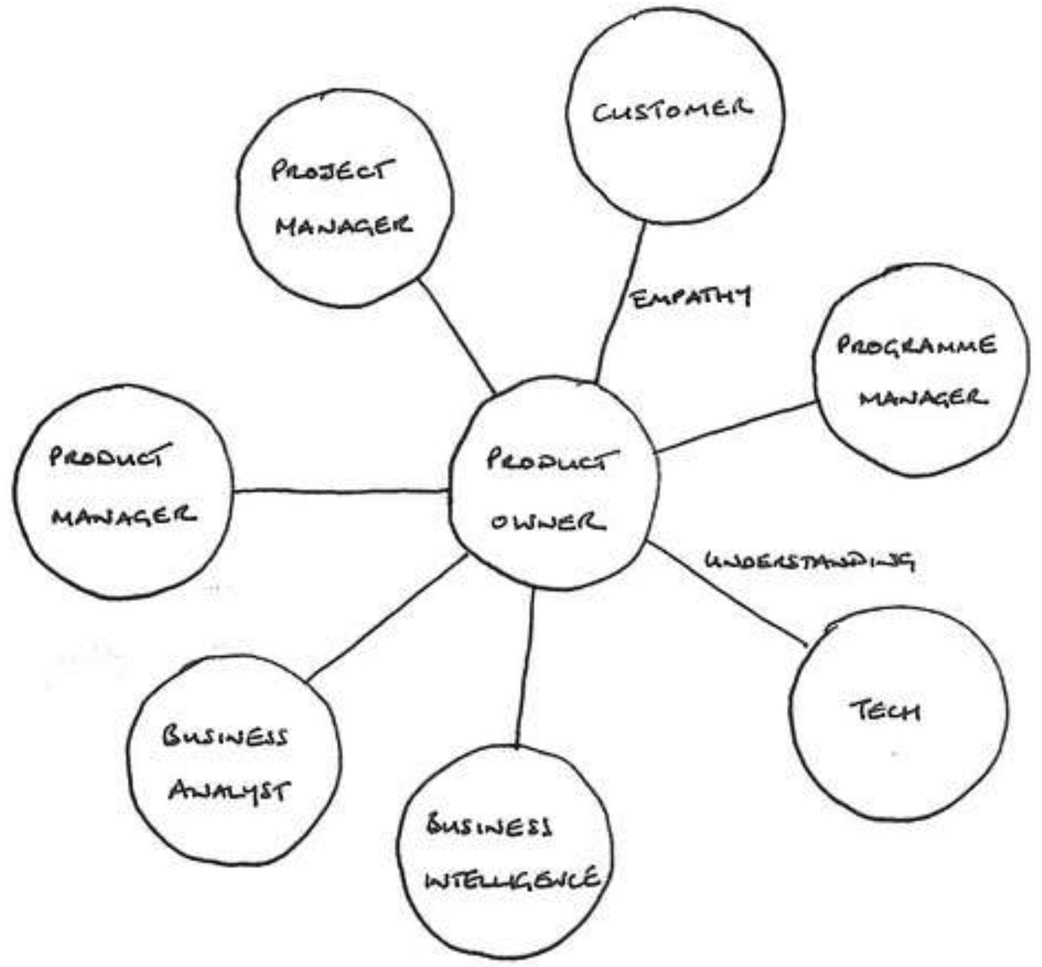
Illustration: caraman – gettyimages.de

Experiences and Challenges: Scrum by the book does not work

- Story cards contain insufficient information
- Story points
  - bad proxy for effort
  - effort estimation in person days works better
  - person-days more transparent for planning
  - open question: effective light-weight method for estimating functionality
- Scrum poker
- Meetings with the whole team
- Difficult: Burndown charts
  - user stories are too large
  - PO unwilling to break down into smaller units

Experiences and Challenges: Roles do not work as designed

- Scrum places strong requirements on its roles (product owner, scrum master, team)
- Product owners need many skills
  - …and often don’t have them
  - Planning: Project manager, business responsible, release manager, responsible for vision, long-term and detail plan, controlling
  - deliver “ready for implementation” user stories: requirements manager, IT design
- Other customer’s roles lose responsibilities (architecture, build, design, etc.)
- Where is the former project manager? Product owner? Scrum master?
- Who decides when quick decisions are needed? When (team) consensus can’t be achieved?



© James Nagy
https://www.jstechdesigns.com/Development/how-to-be-a-great-product-owner

+ Experiences: Some Scrum practices worked

- Daily Scrum meeting
  - quick status overview, even for large teams
- Kanban board
  - several boards: specification, implementation, defects
- DoD
  - includes documentation tasks
- Sprint retrospective

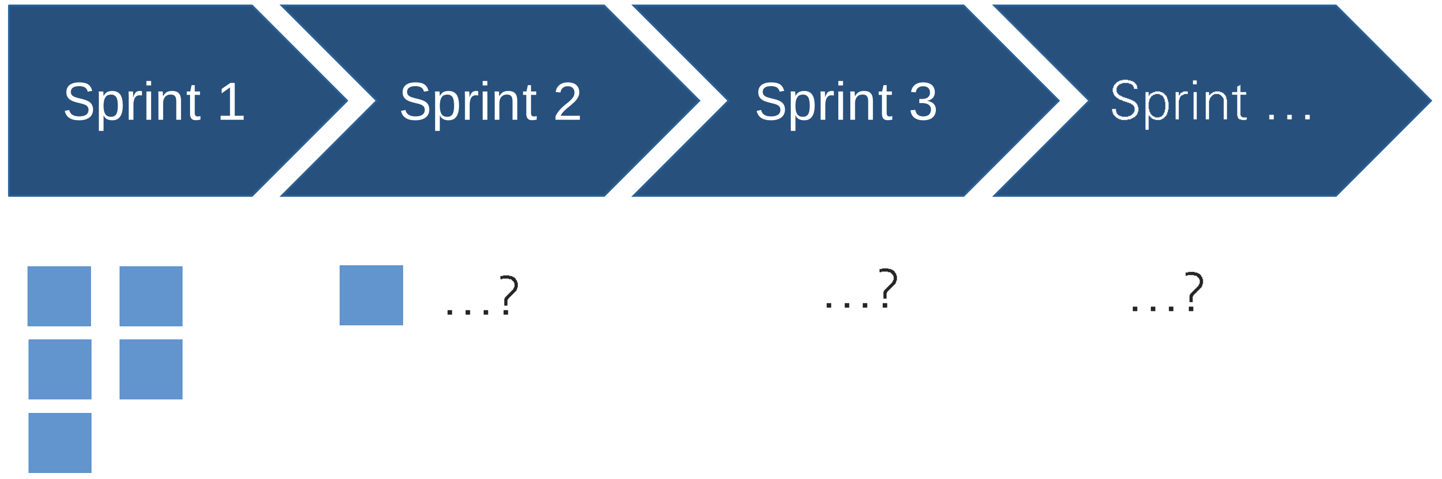
+ Experiences: Success Factors for agile projects

- Continuous planning
- Mini-Waterfalls per feature: Frontrunner and mini-specs
- Prioritize software quality
- Eliminate time killers
- Software-OEM
- Emphasize test automation
- One team approach

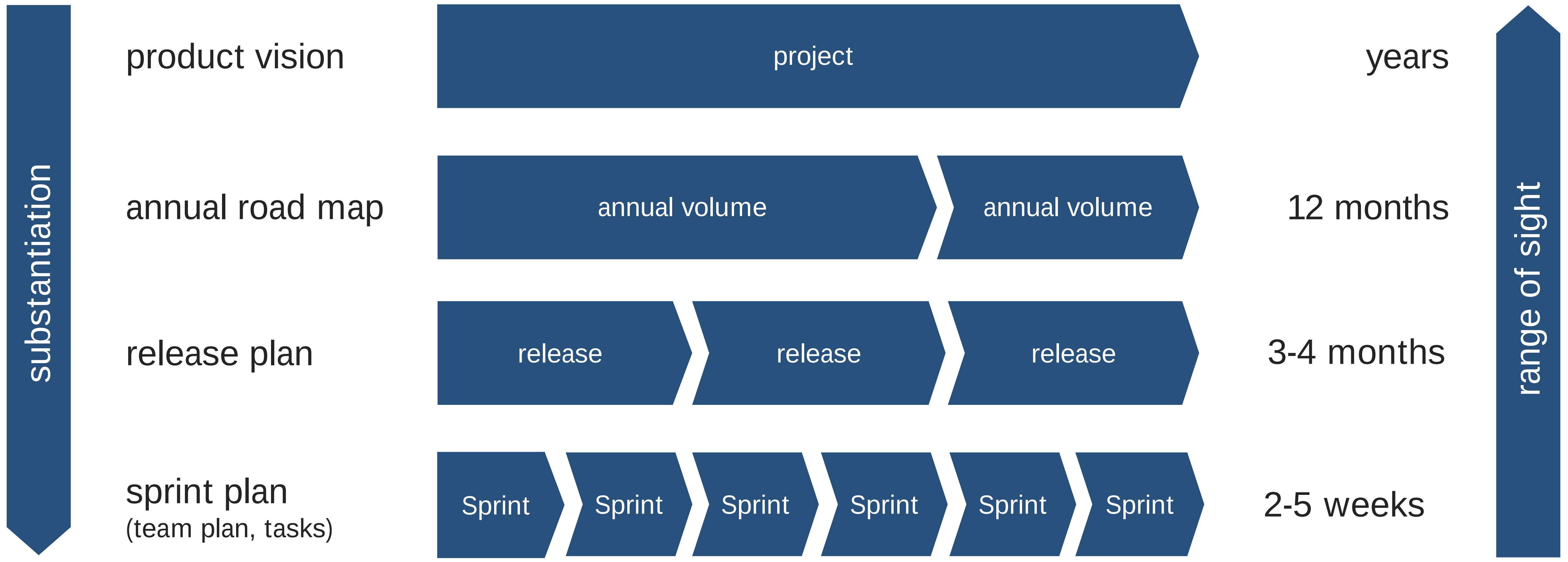
Success Factor: Continuous Planning in Levels

Crucial problem in agile projects: Missing range of sight in planning

User stories per sprint:



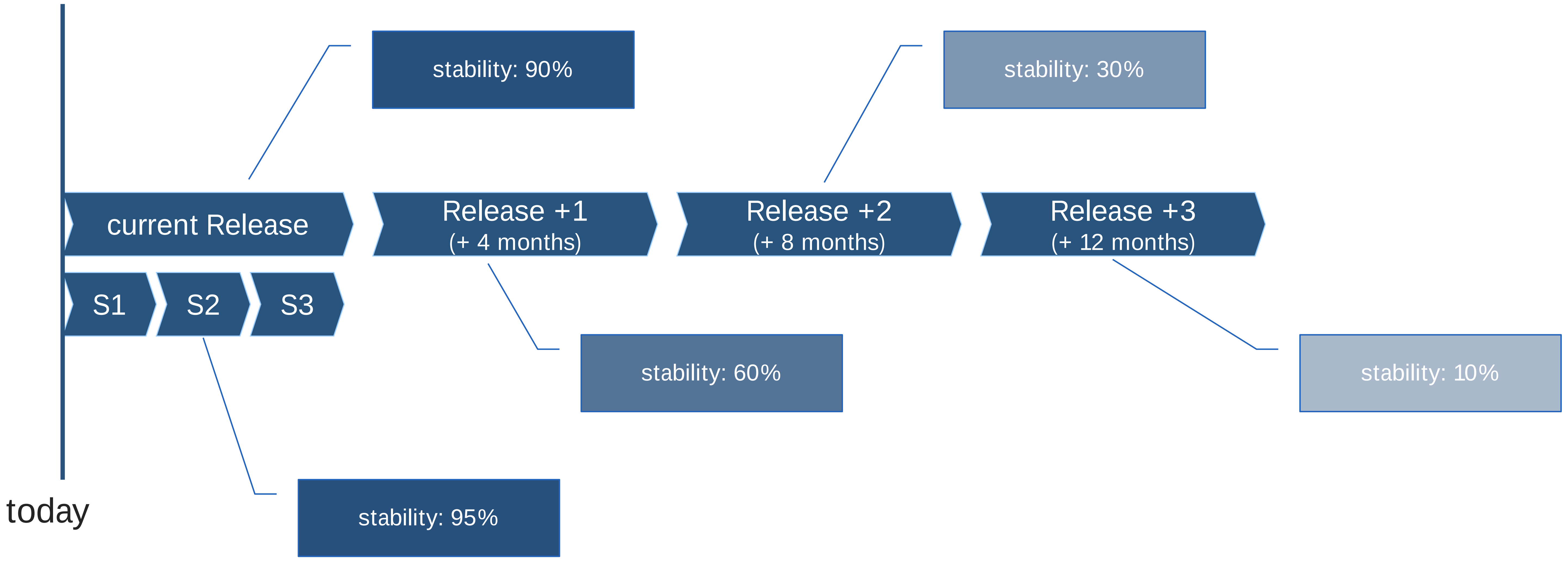
We do long-term plans in several levels.
This is critical for long-term success in large projects.



Although stability is low, long-term planning is critical for achieving vision and prioritizing correctly.

Short term planning: exact (person-days)

Long-term planning: approximate (T-shirts)



Continuous Planning: Adapt the product vision through estimation workshops with the customer.

Workshop objectives:

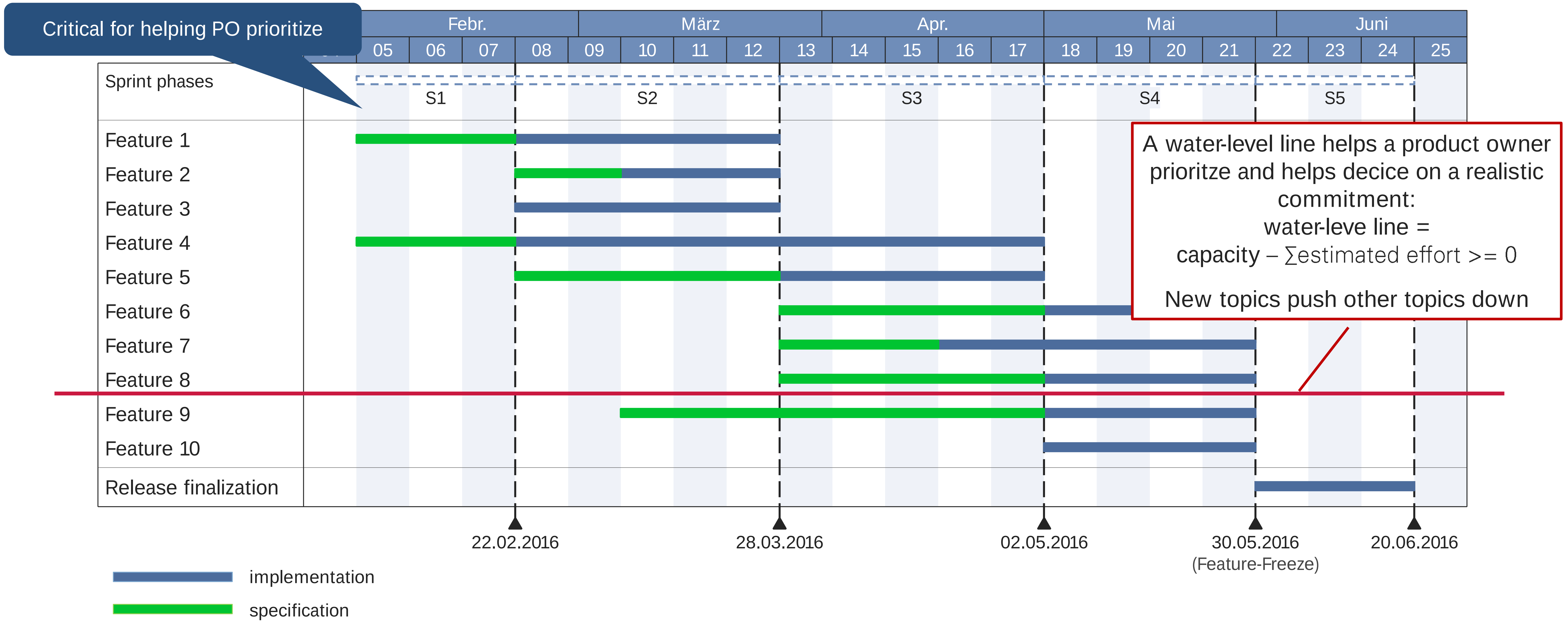
- Validate product (annual) roadmap: Rough efforts and feasibility.
- Identify risks.
- Identify dependencies.

Side effects:

- Customer's discloses goal to the entire team.
- Develop mutual trust.

Procedure: Estimation on the basis of T-shirt sizes (e.g. S, M, L) with defined rough effort intervals.

For each release, we plan and prioritize features and maintain a mile-stone plan.



Adapt sprint length to context: Don‘t confuse planning with deployment and IV&V

Confusion between sprints as planning units vs. deployments

Sprint as planning unit

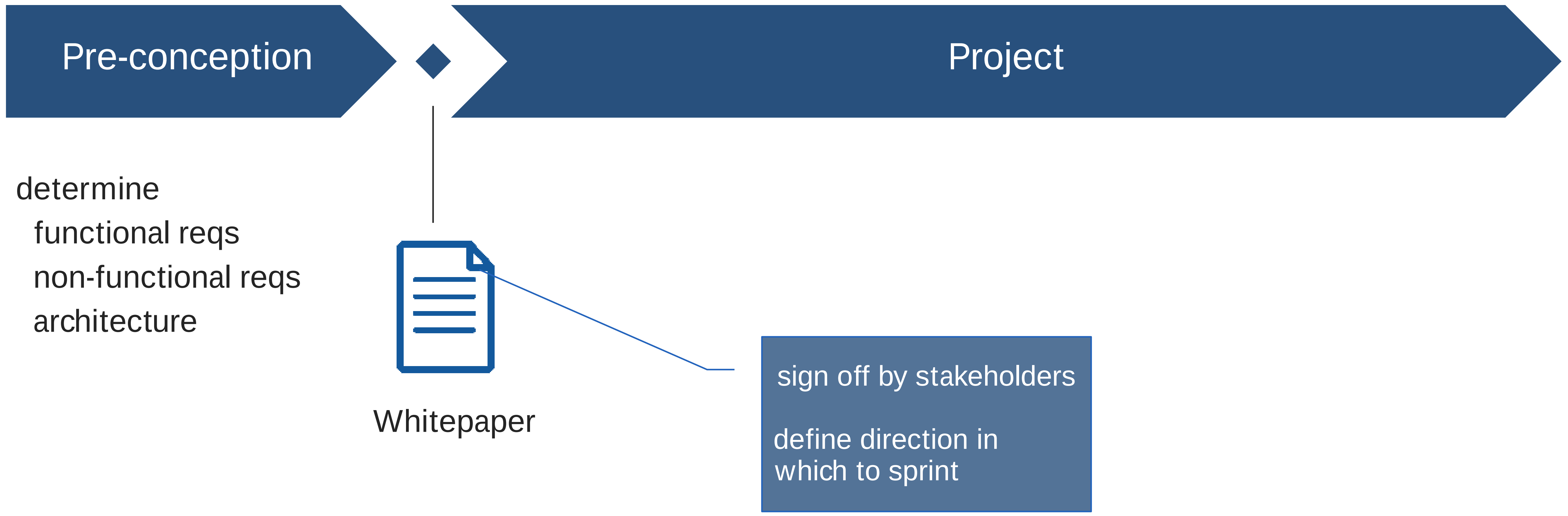
- commit user stories for upcoming sprint
- groom backlog for next sprints
- leads to: short sprints to keep a sprint stable and react to changes quickly

Sprints as (internal) deployment unit

- waterfall mindset:
  - tasks must be completed as committed
  - any defect found in V&V may prevent roll-out
- consequence: code freeze and high amount of testing before deployment
- leads to long sprints to reduce overhead

Success Factor:
Frontrunner and Mini-Specs

A pre-conception phase defines architectural cornerstones for each project and involves stakeholders.



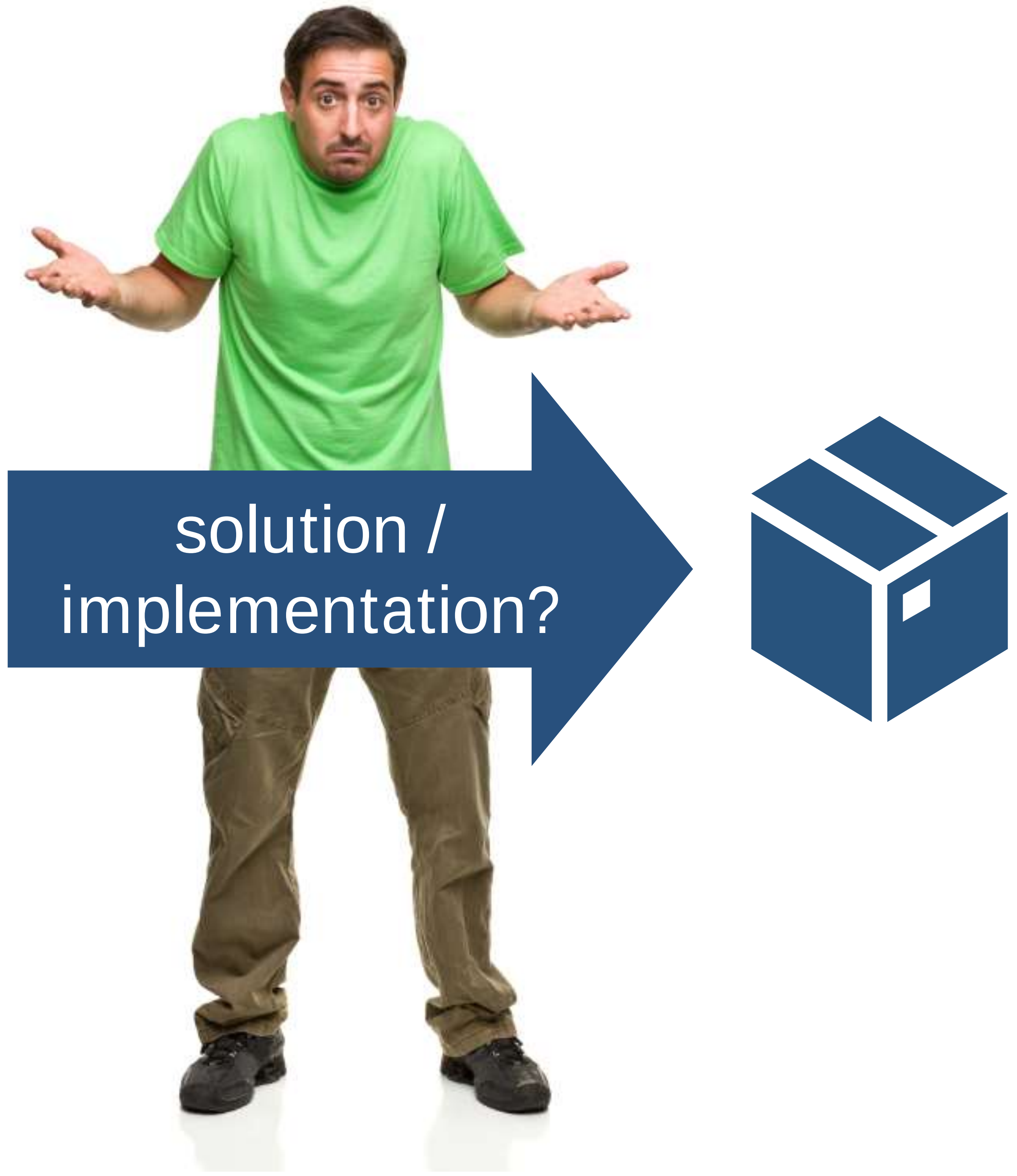
Story cards are insufficient: The path from complex business problem to technical solution is bumpy.

User Story

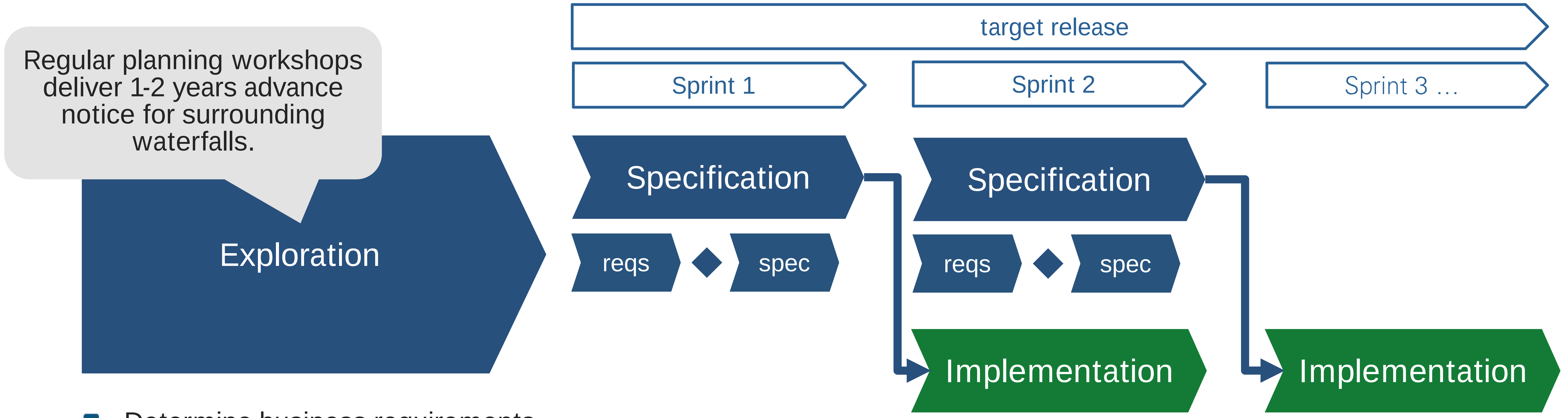
"*As* service consultant, *I want* to see the maintenance history of a vehicle, *so that* I can offer a better customer service"

Acceptance criteria:

- …



Frontrunner teams clear the path for implementation. Mini-waterfalls allow long-term planning.

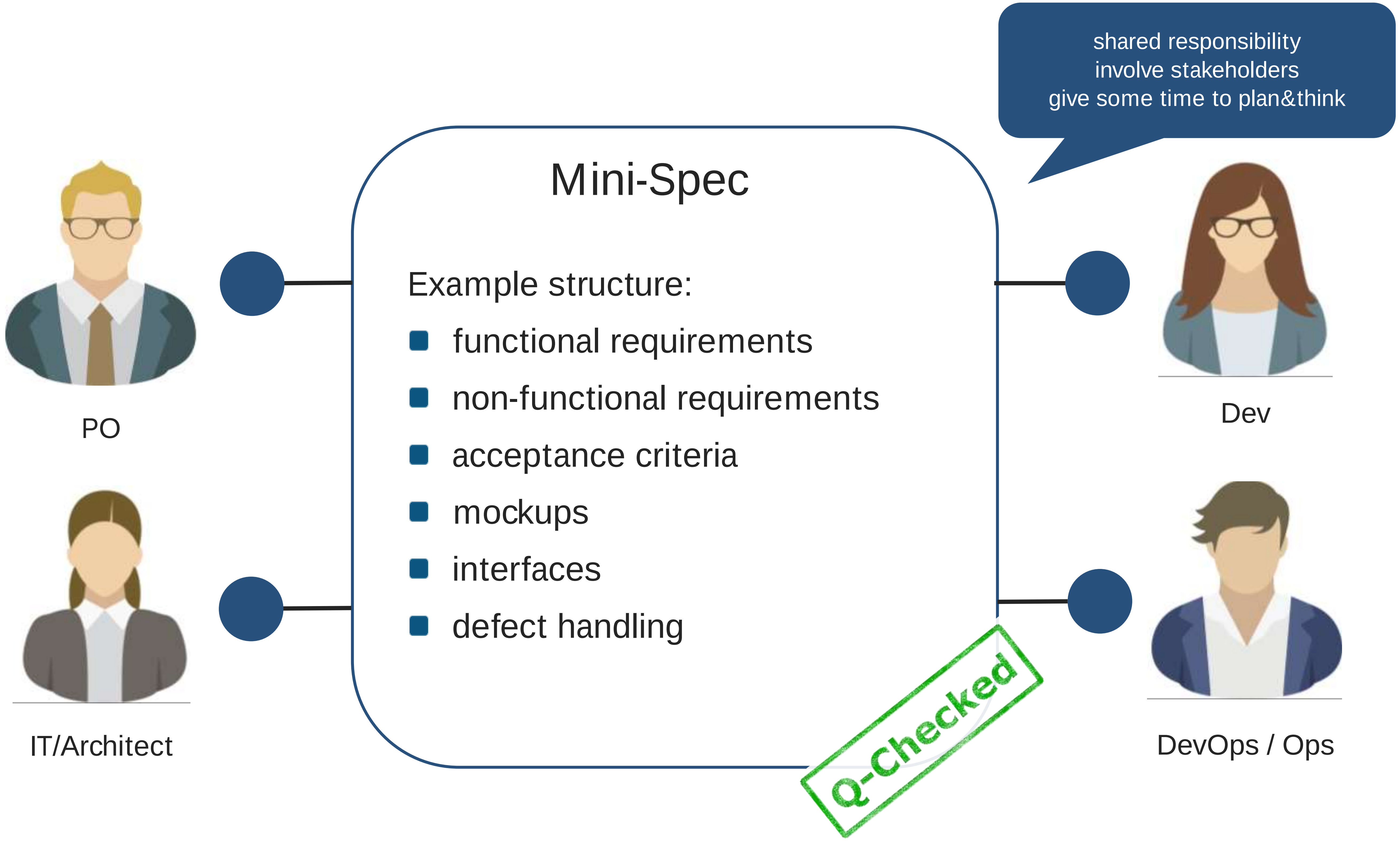


- Determine business requirements
- Develop solution idea
- Clarify risks:
  feasibility, proof of concept
- Determine dependencies

Recommendation:

Front runners accompany implementation or even do it themselves

Mini-Specs help achieve Definition of Ready.



Mini-Spec

Example structure:

- functional requirements
- non-functional requirements
- acceptance criteria
- mockups
- interfaces
- defect handling

Challenge: Documentation is often neglected when prioritizing features.

Documentation needs to be part of DoD.

- User manual
- System manual / design
- Architecture / detailed design

Documentation changes mindset compared to waterfall!

- Waterfall:
  - Definition & creation of documentation up-front
  - Design is blueprint for implementation: correctness & completeness
- Agile:
  - Create documentation close to implementation
  - Incrementally update documentation per feature: Integrate mini-specs into existing documentation

In agile projects, the challenge is to incrementally grow documentation and keep it up-to-date.

Main challenges:

- How to maintain documentation / keep it up to date?
  - Integrate documentation grooming into tasks, similar to refactoring?
- What is the minimum set of documentation required?
- Eliminate documentation redundancy.
- Automate and generate from single source of specification where possible.

Success Factor: Software Quality vs. Feature Greed.

Cost of development

= work hours
x cost per hour
+ technical debt

Build a counterpart to feature greed of product owners. This reduces quality debt.

Success factor: **Don‘t negotiate quality!**

Systems require phases in which fewer new features are developed and the system is solidified.

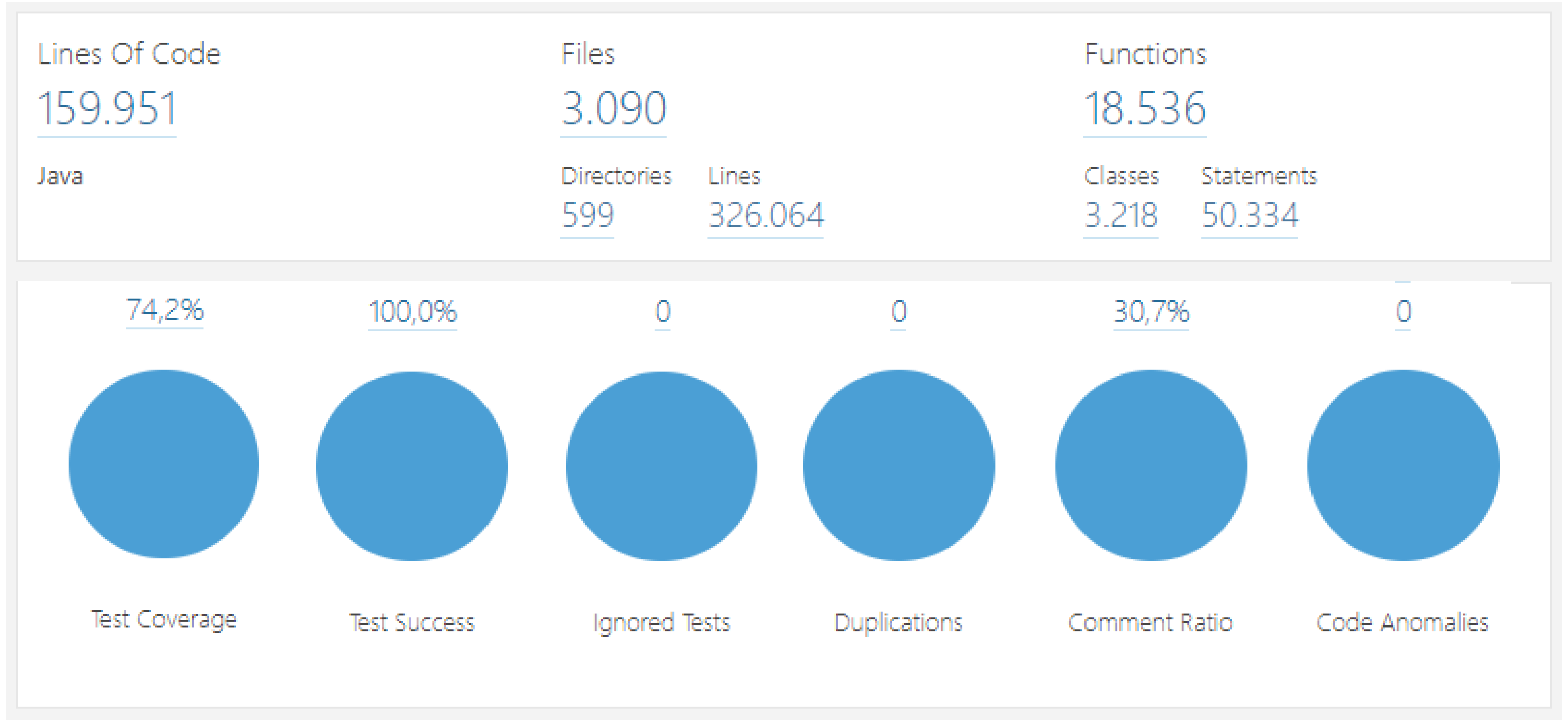
Solutions:

- A quality (refactoring) backlog with ca. 10-20% of Sprint capacity.
- Quality Sprints
- A Bug Hunting Day / Quality Day per release.
- A contractually agreed-upon quality contract based on measurable KPIs.
- A comprehensive Definition of Done, part of which is the quality contract.

Foto: QAware Quality Day

Product Quality is omnipresent in the project.

sonarqube



Lines Of Code	Files	Functions
159.951	3.090	18.536
Java	Directories 599, Lines 326.064	Classes 3.218, Statements 50.334

Test Coverage	Test Success	Ignored Tests	Duplications	Comment Ratio	Code Anomalies
74,2%	100,0%	0	0	30,7%	0

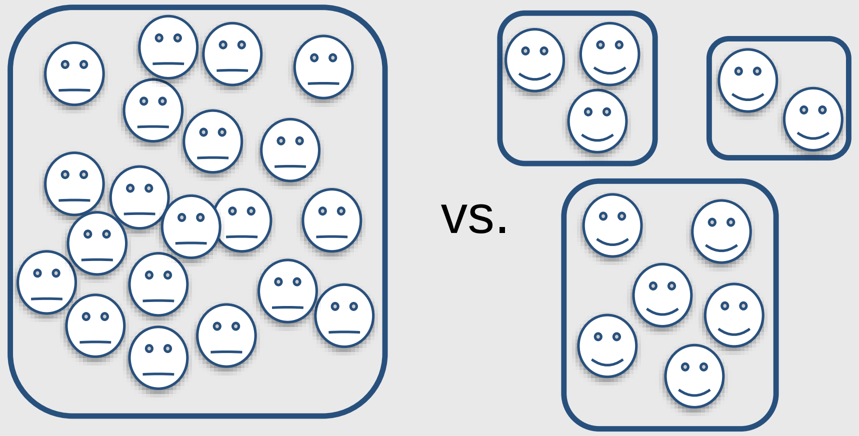
Success Factor:
Eliminate productivity killers.

Eliminate time consumers and productivity killers in Agile Methodologies.

Only relevant representatives for

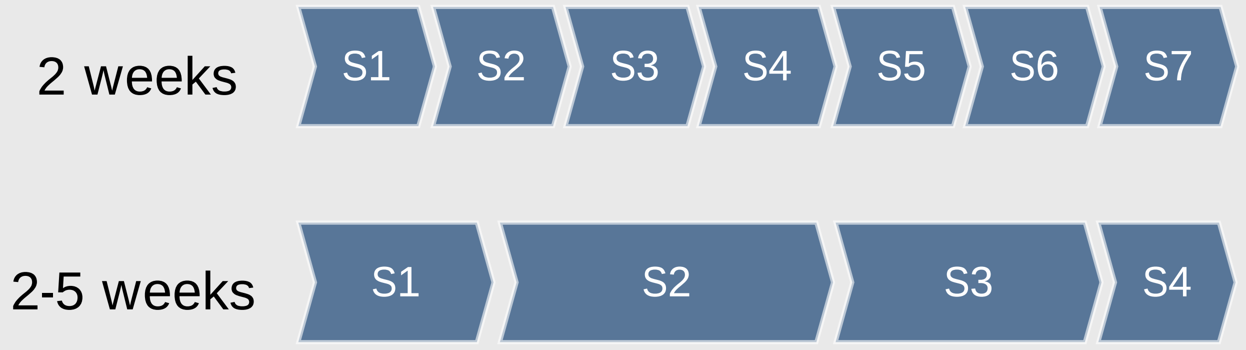
- Backlog-Grooming
- Sprint-Planning
- Sprint Review
- Definition of Done

Develop solutions in small groups

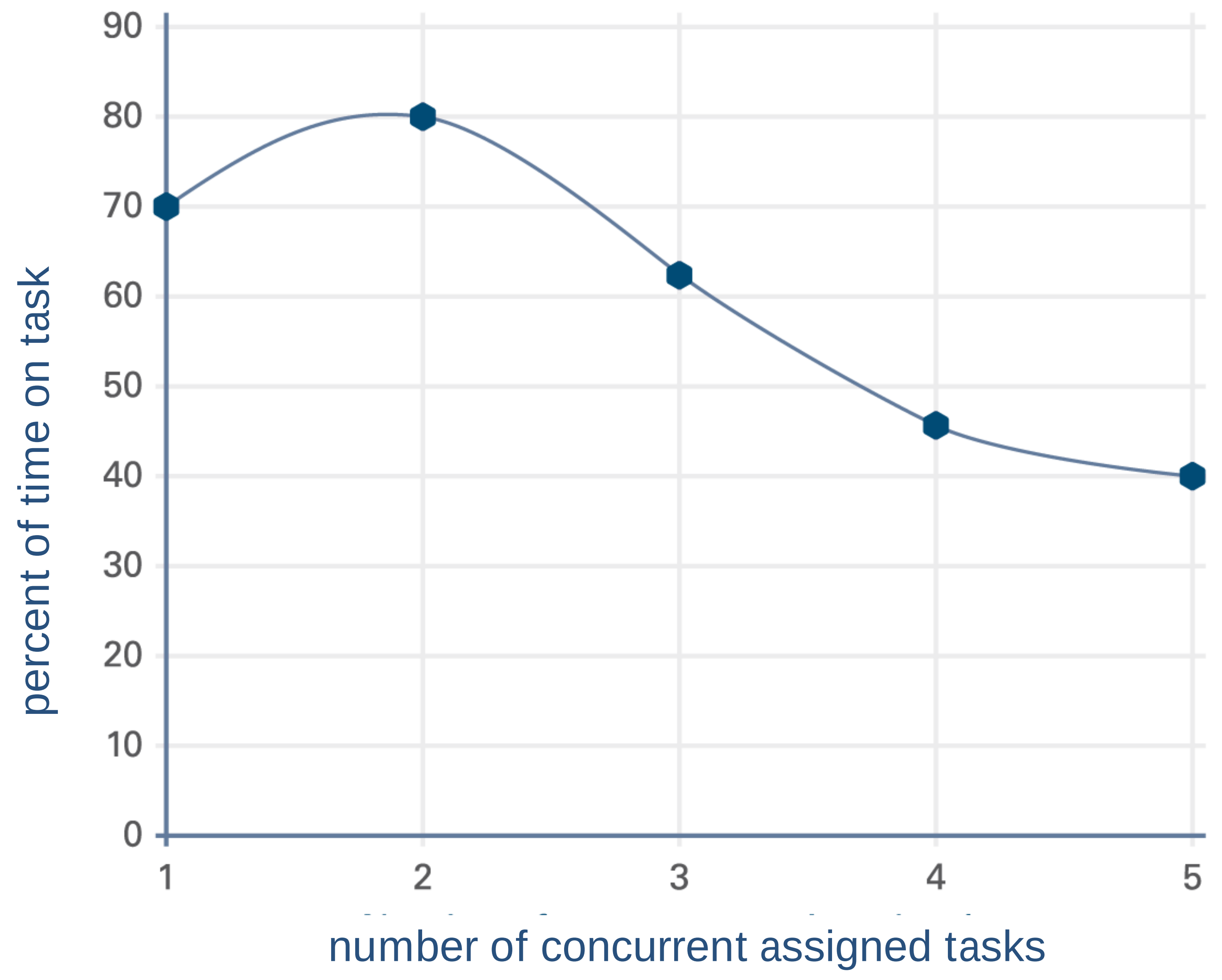


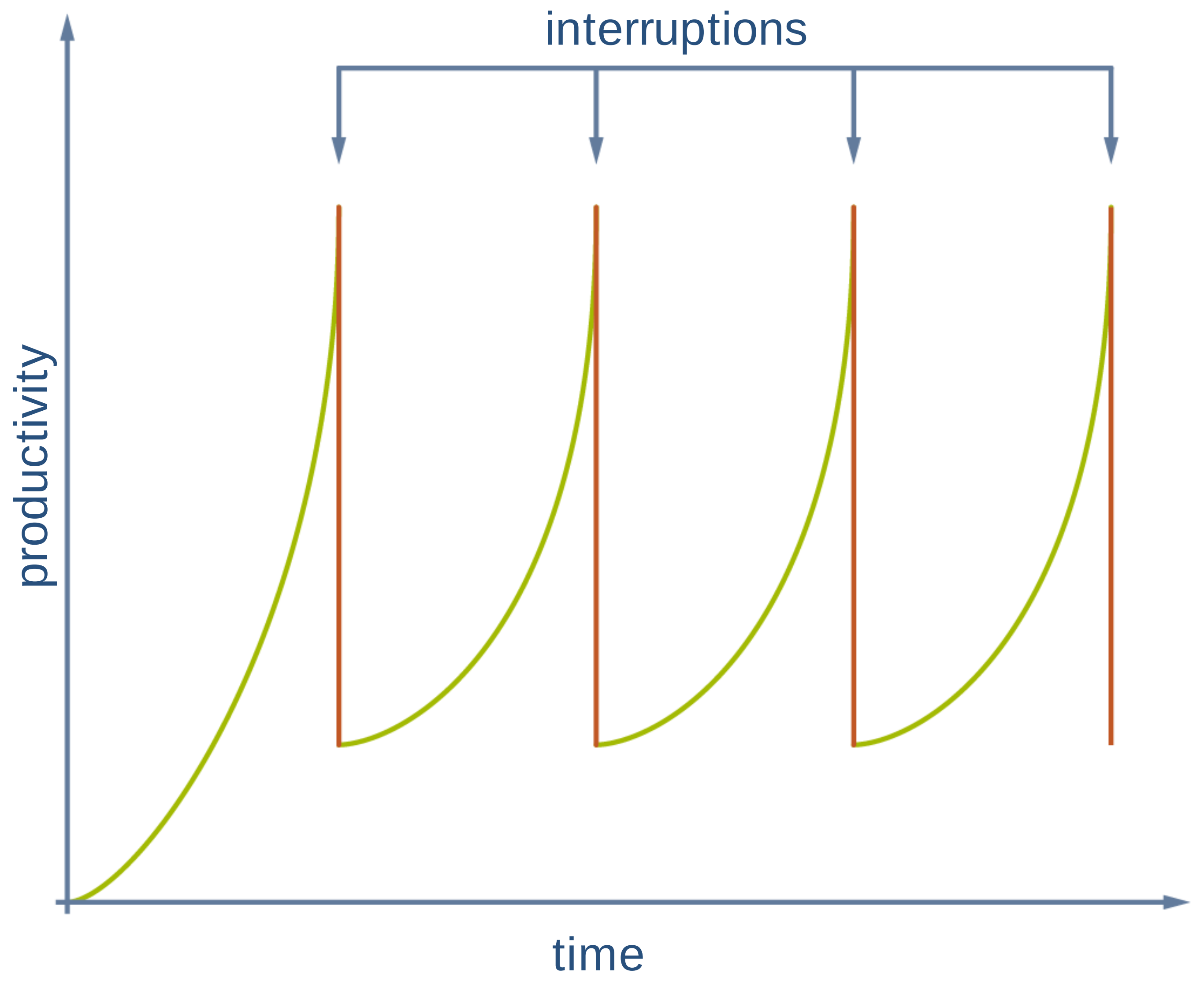
Daily Scrum is timeboxed

Flexible sprint duration within release



Unplanned interruptions are effective productivity killers.





Success Factor: Vertical integration depth

vertical integration depth := percentage of own code vs. code from third-party libraries (e.g. open source components)

Maximum speed in development through software OEM approach.

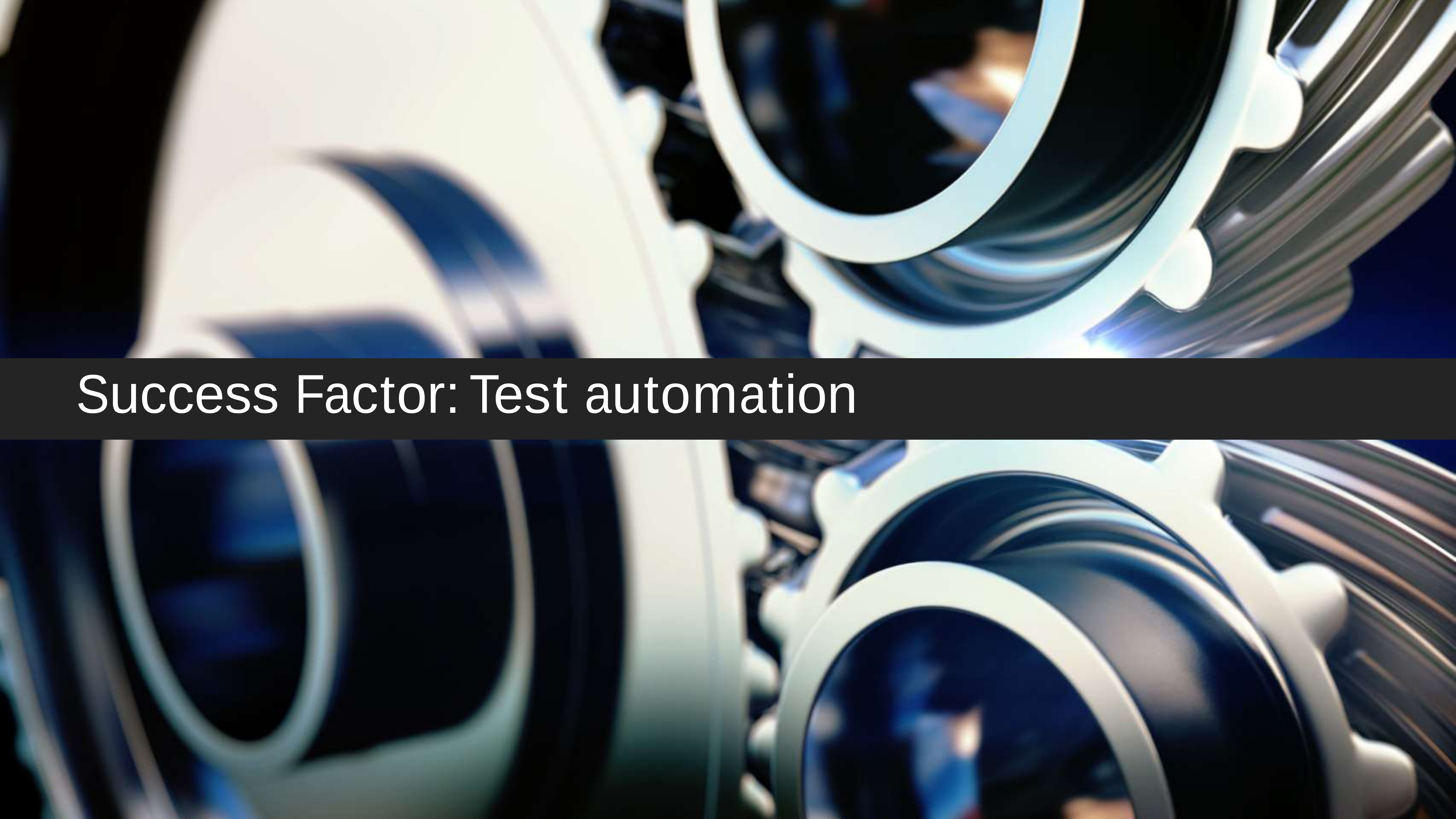
- Software-OEM means: Build software with low vertical integration depth based on Open-Source components.
- Careful selection of third-party components based on checklists:

… during research and selection:

- is this component necessary?
- is the licence compatible?
- approvement by customer needed?
- is the component maintained?
- are there known security issues?
- …

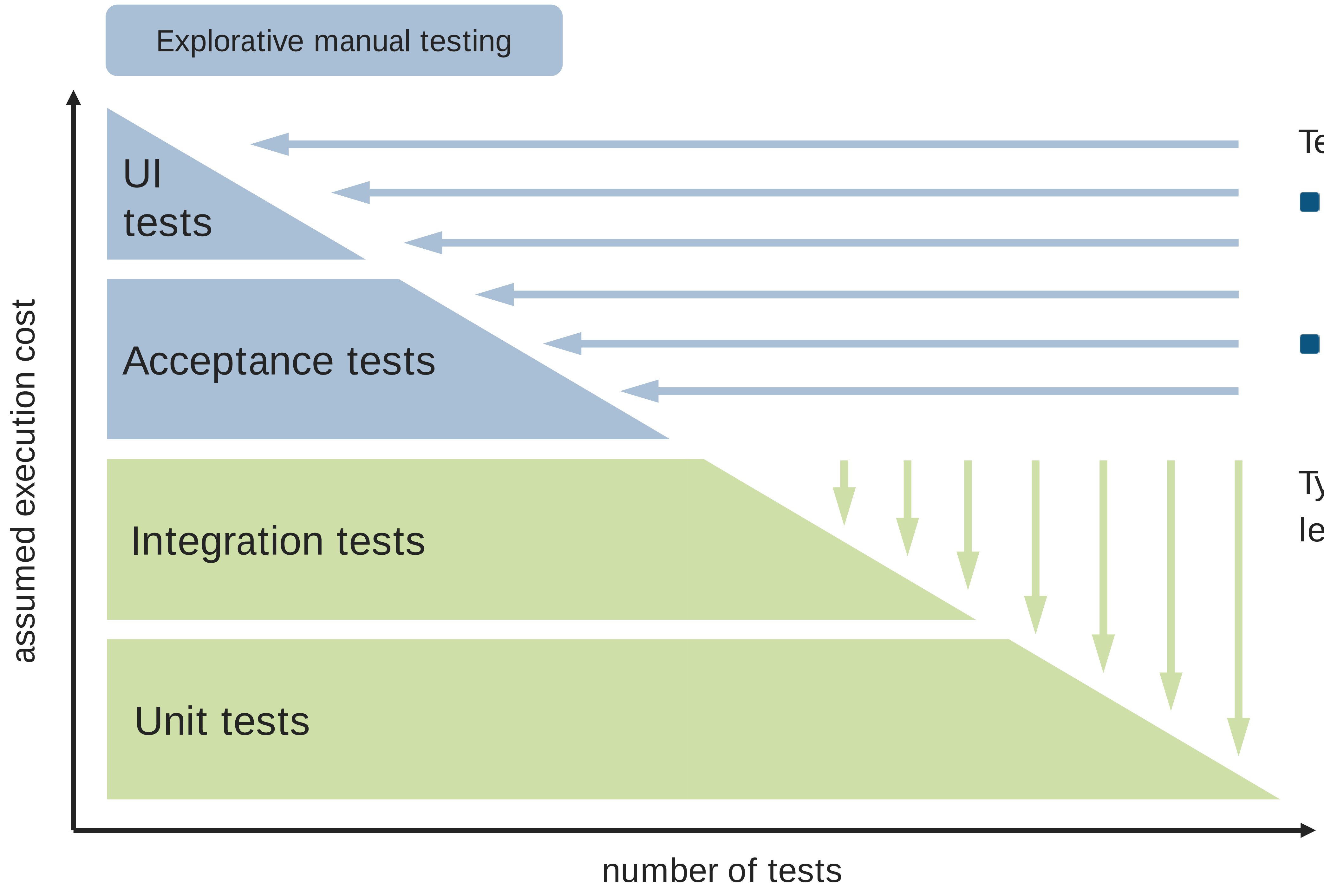
… during integration and maintenance:

- loose or tight coupling?
- is compliance documentation complete?
- are licences included appropriately?
- migration to newer version needed?
- …



Success Factor: Test automation

Large agile projects require test automation on all levels.



Test automation at these levels

- very well suited to regression tests and reduces the effort needed for manual testing
- does not relieve one of the obligation to perform manual and exploratory testing

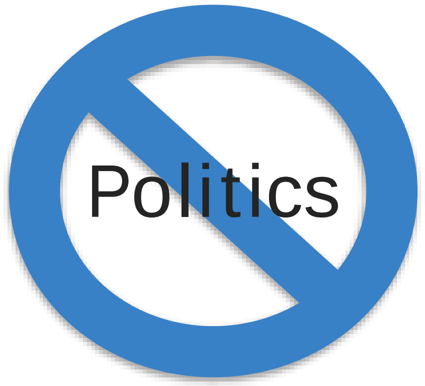
Typically good test automation at the lower levels with all advantages.

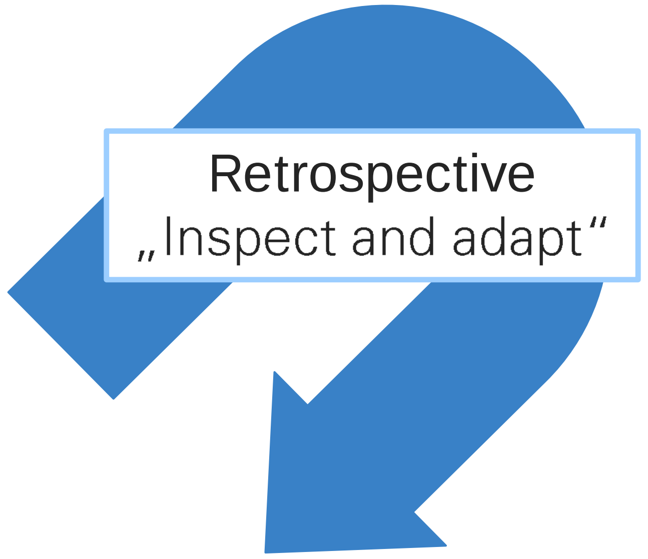
automated testing

One Team Approach

Boundaries between companies do not matter in collaboration. What counts is joint project success.

Close collaboration & partnership.
Close to the customer (co-location?)





Constructive solution-finding
Courage to bear friction

Celebrate successes together

1. Agile vs. Waterfall
2. Success factors
3. **Future Challenges**
4. Summary

There is DevOps on the horizon.

We are going DevOps.
This will even further increase our speed-to-market, business value, and innovation.
And our customer satisfaction.

Illustration: kbeis – gettyimages.de

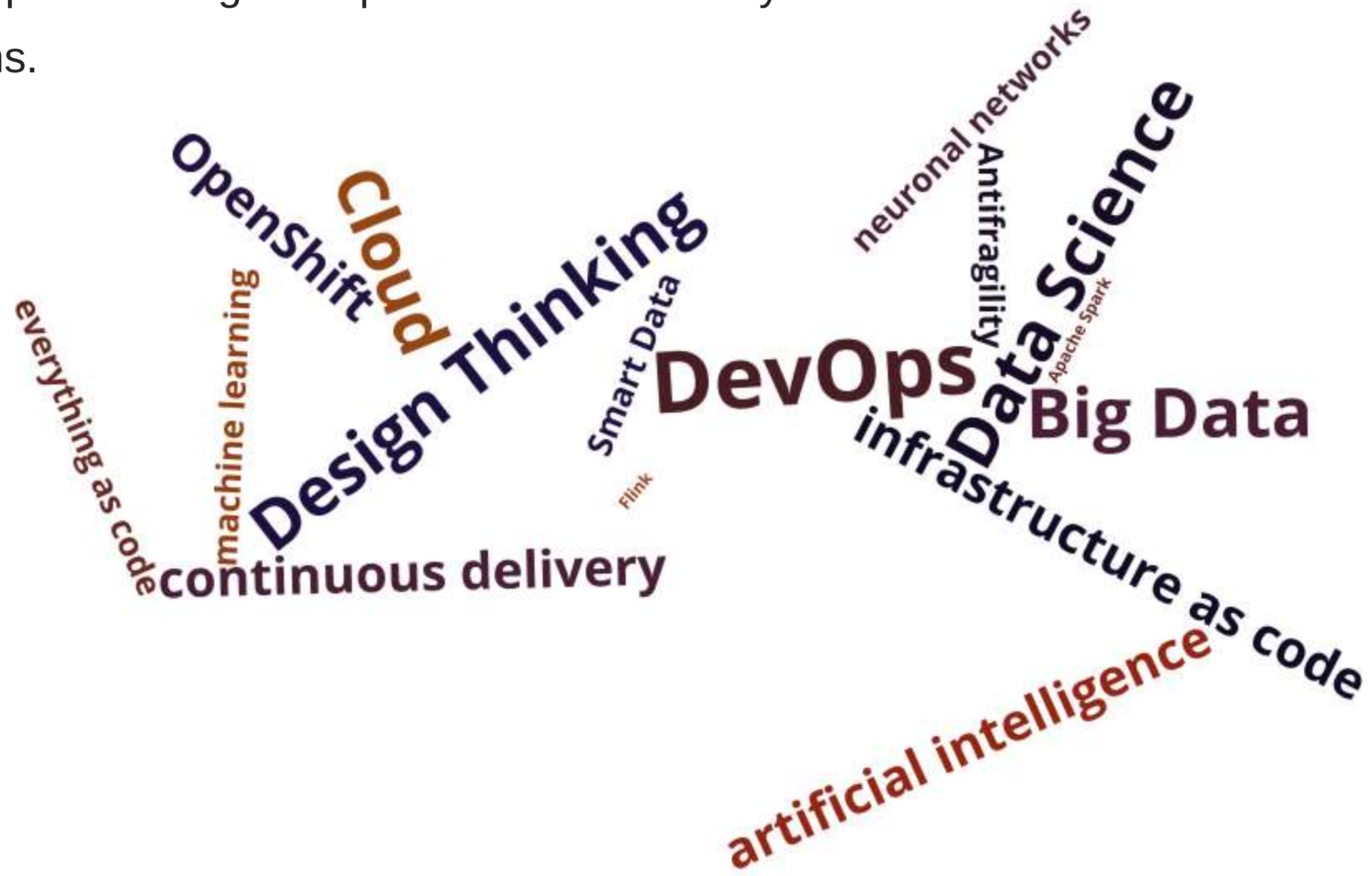
There is DevOps on the horizon.

Illustration: kbeis – gettyimages.de

Trends: Many Buzzwords raise expectations.

New technologies and processes gain importance and visibility.

This raises expectations.



Challenge: Master difficulties to create opportunities.

Peak of inflated expectations: Technologies will solve all problems.

However, nothing is for free.

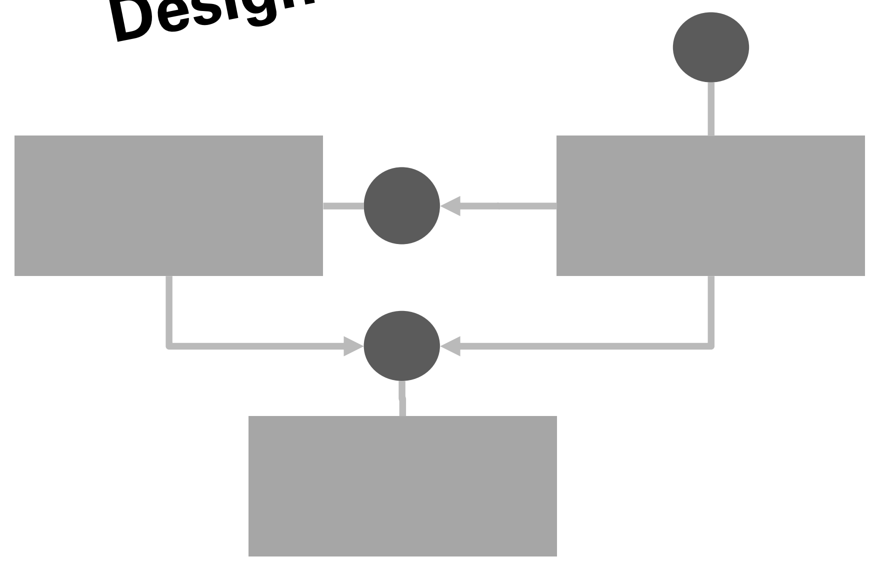
Example: DevOps

- Pressure from customers growing
- Expectation with DevOps: 24/7 support
- Problem: SMEs can‘t follow the sun
- Challenge + Opportunity: Antifragility
  - Combines techniques from AI and cloud.
  - Goal: achieve self-healing / repairing properties.
  - Benefit: reduce need for manual support.

Cloud-native applications increase design complexity: Components change along lifecycle.

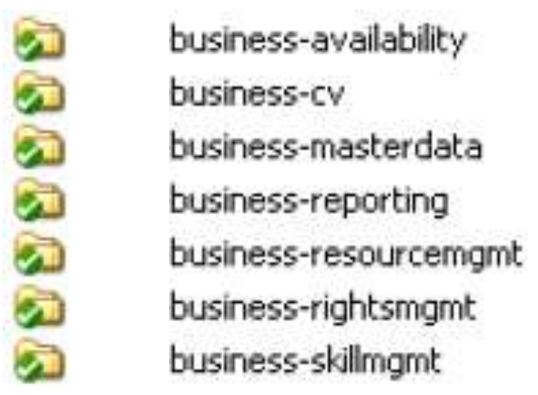
DESIGN | BUILD | RUN

Design Components



1:1

Dev Components


business-availability
business-cv
business-masterdata
business-reporting
business-resourcemgmt
business-rightsmgmt
business-skillmgmt

?:1

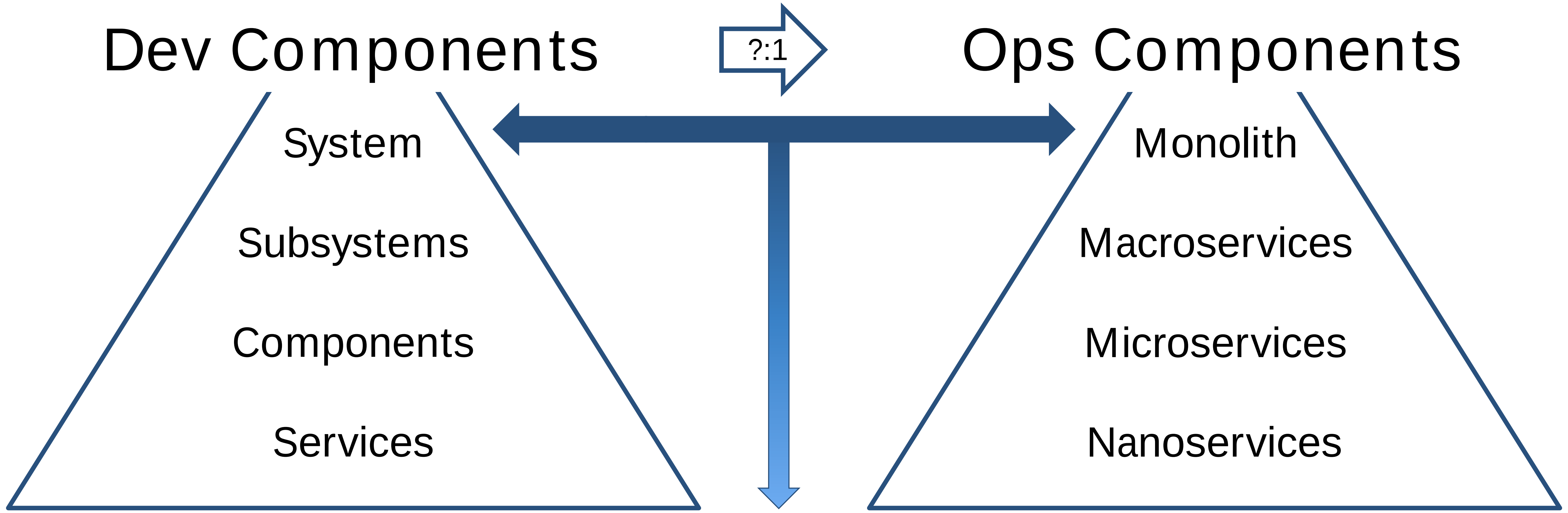
Ops Components

- Complexity unit
- Data integrity unit
- Coherent and cohesive feature unit
- Decoupled unit

- Planning & Assignment unit
- Knowledge unit
- Development unit
- Integration unit

- Release unit
- Deployment unit
- Runtime unit (crash, slow-down, access)
- Scaling unit

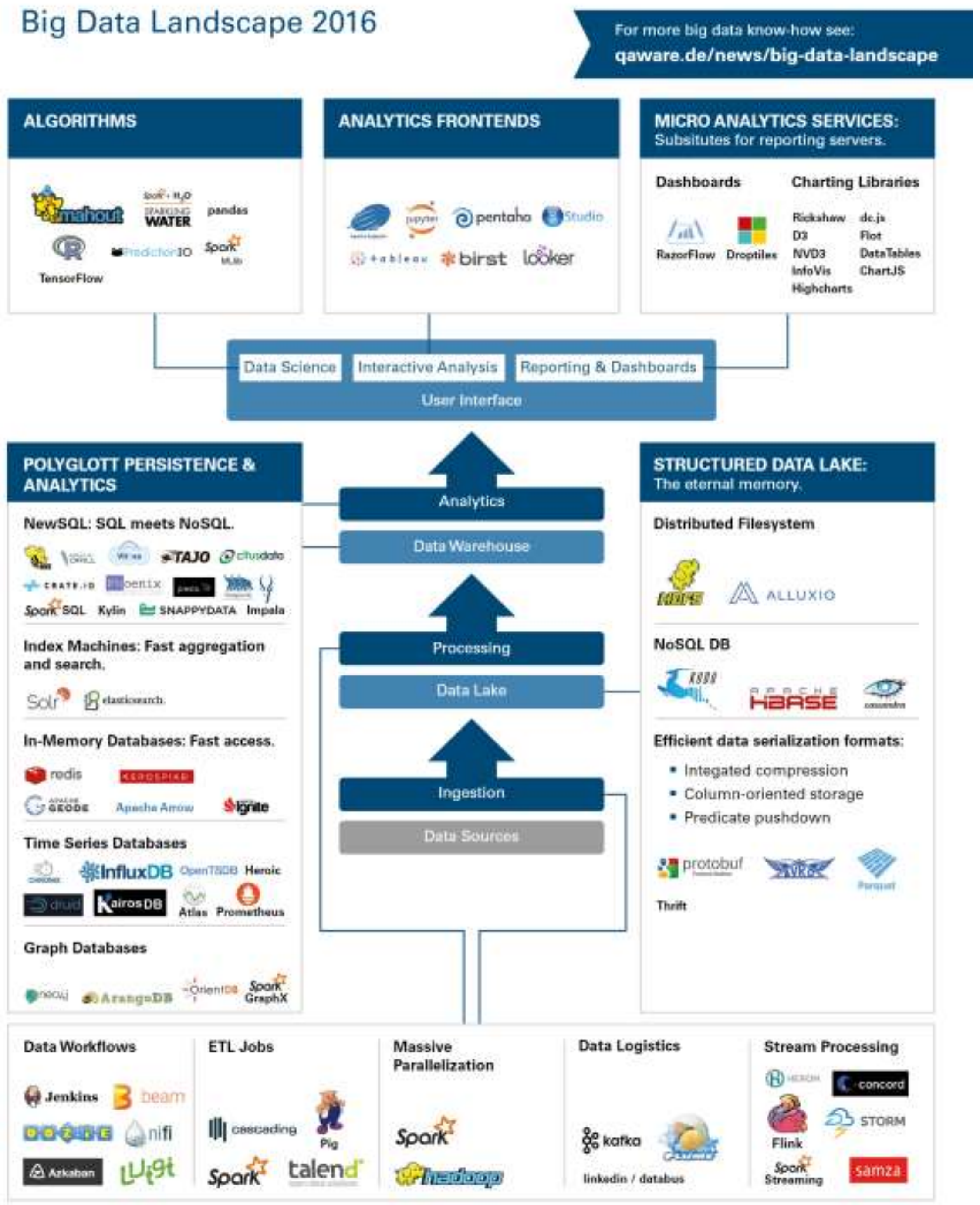
New technologies introduce new complexity. There is never a silver bullet.



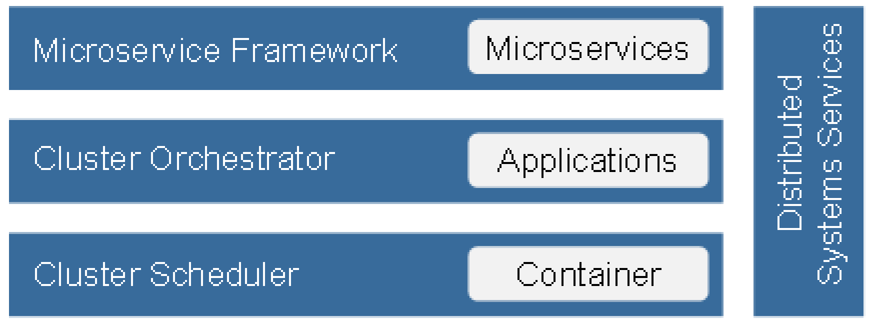
Decomposition Trade-Offs

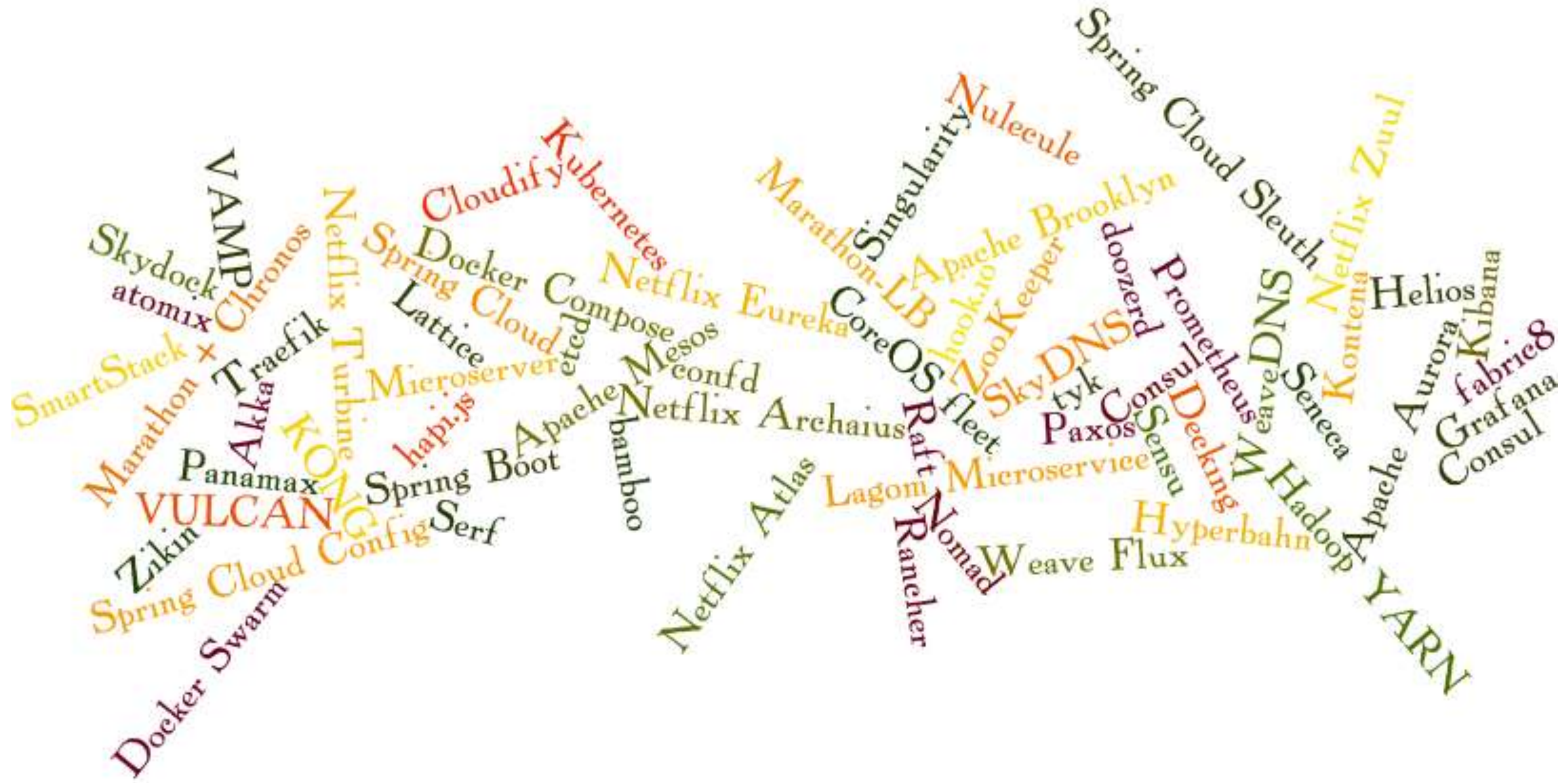
+ more flexible scaling	– distribution debt: Latency
+ runtime isolation (crash, slow-down, …)	– increasing infrastructure complexity
+ independent releases, deployments, teams	– increasing troubleshooting complexity
+ better resource utilization	– increasing integration complexity

Trend: Applications combine many complex frameworks



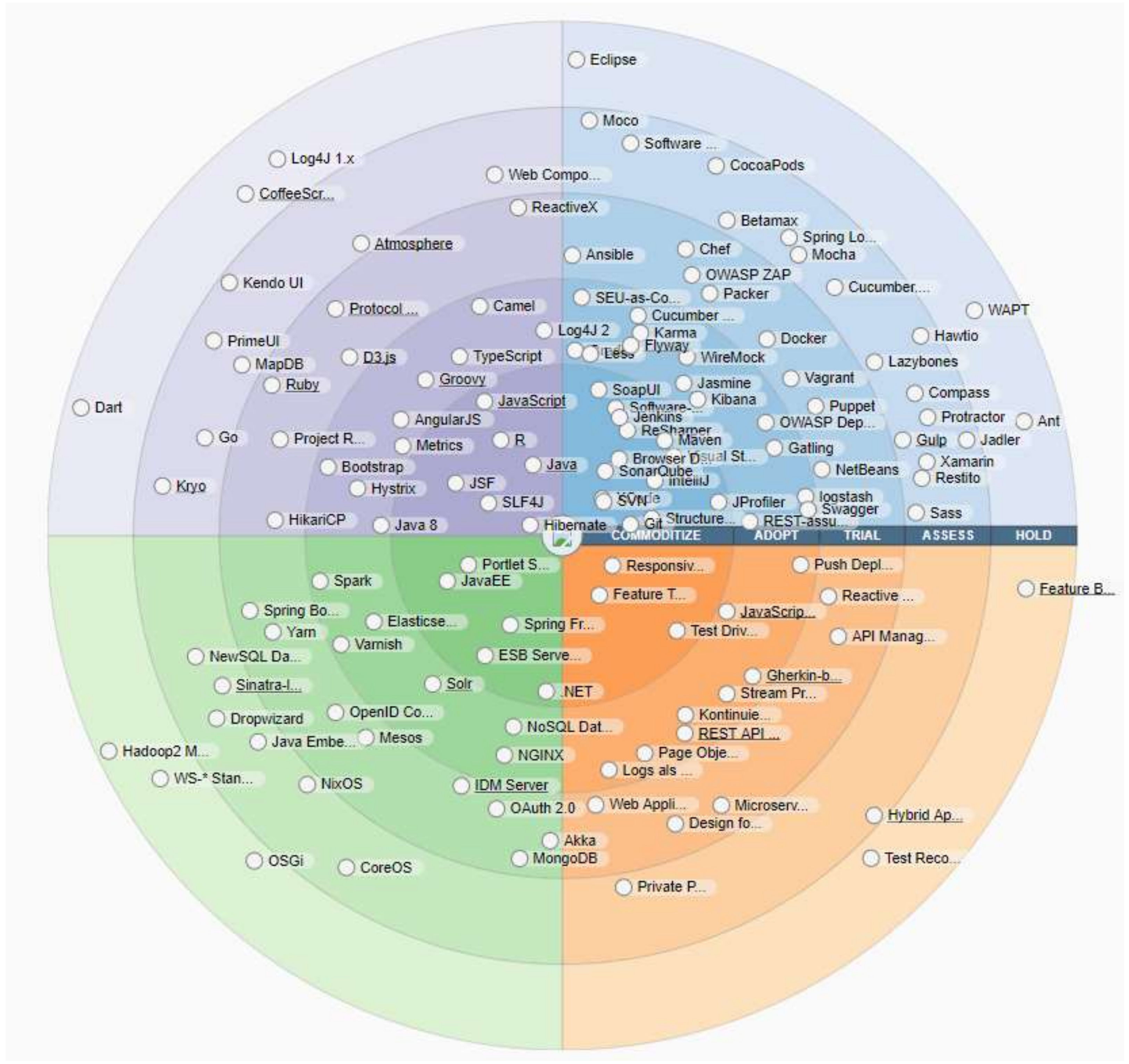
Big Data landscape





Cloud landscape

We use a technology radar to aid technology decisions.



Challenge: Combining powerful tools without planning or understanding is dangerous.

Complexity of technologies is rising.

Technologies used based on coolness without clear goal/value.

Agile Trap: Agile teams start using barely-understood technologies without planning.

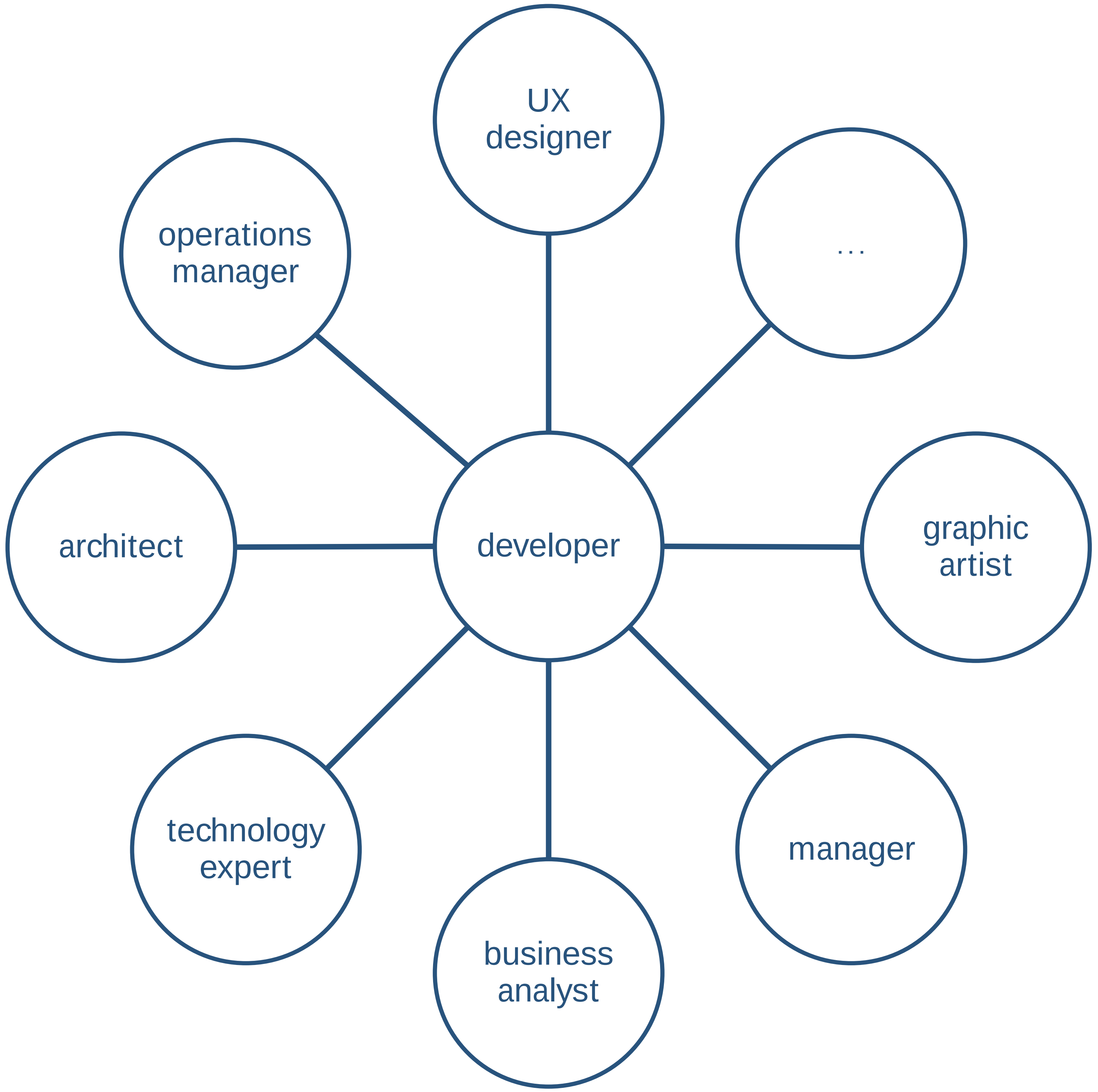
New trends lead to:

- Combining many different products (with many defects)
- Complex, powerful tools: use with care

Challenge:

- How to do software engineering?
- How to maintain an overview?
- How to maintain intellectual control?

Challenge: Requirements on developers rise.



Challenge: Mixed teams without boundaries diffuse responsibility.

Teams mix across company boundaries

- across suppliers: different suppliers develop the same components
- across customer/supplier: customers are part of development team
- across skills: development, UX experts, designers, graphic artists, operations, …
- no clear separation between contributions of different teams/companies

This raises many questions

- Co-location vs. distribution & communication?
- What happens to warranty?
- Who is responsible for quality?

Challenge: Contracting

Contracting evolves towards time & material

- Who is responsible for the outcome? Scope?
- Who is responsible for quality?
- How can emergent architectures work for such teams?

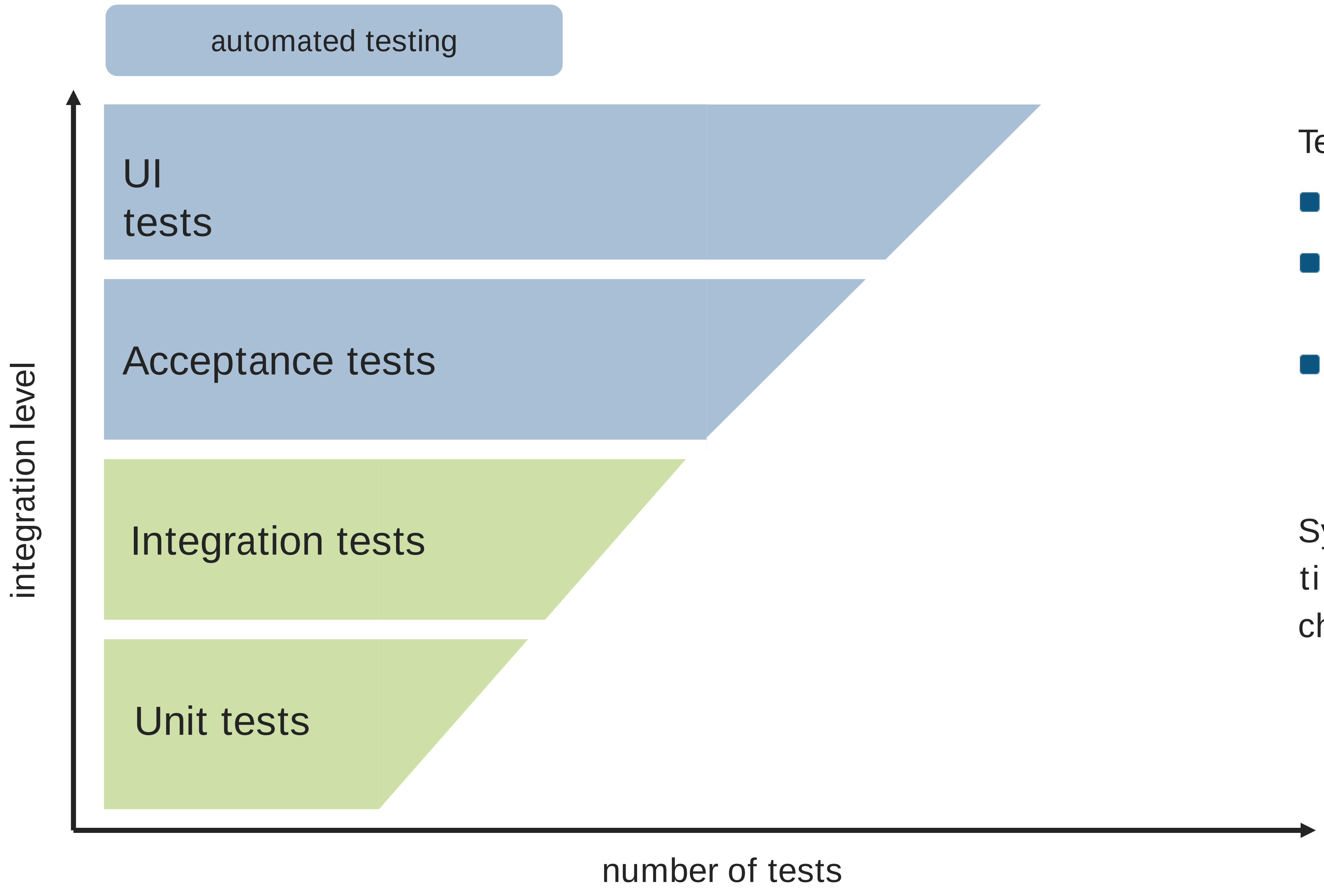
As supplier:

- How to keep quality as USP? What about customer satisfaction?
- How to distinguish your company's contribution from others?

How can we write contracts in the future?

- Is it meaningful to model teams, contribution and collaboration as components with interfaces?
- Components: teams, responsibilities; Interfaces: communication, data flow.
- If yes, do the same rules apply as for software design?

Challenge: Testing pyramid may need to be inverted



Test automation at high levels required:

- anything can change at runtime
- critical: detect backend (service) failure quickly
- manual and exploratory testing are still needed

System behavior undefined at integration time: (Micro-) services are external and can change at runtime.

Takeaway Messages: Success Factors for Agile Projects

- Agile processes are often surrounded by or originate from waterfall thinking.
- Success requires embracing and involving stakeholders.
- Opportunity: Combine best of contract work and agile:
  - ability to long-term planning and commitment
  - flexibility to change and incorporate feedback

In our experience, critical for successful agile projects are:

- Courage to plan in agile mode.
- Frontrunning: keep up speed while bridging the gap between problem and solution.
- Establishing quality as counterpart to feature greed.
- Establishing light-weight formalities.
- Establishing efficient meeting structures.

Takeaway Messages: Future Trends

Future trends: Since everyone is agile now, we can increase speed even more

- Mixed teams: Throw people into team; they will organize themselves efficiently?
- Combination of complex, poorly understood technology will lead to great, robust systems?

Challenges

- Contracting: Fixed-price contracts with warranty lose importance.
  - Future: Contracts on time and material only (Labour Leasing)?
  - Process and Contract modelling: contract/process models using components and interfaces?
- Education: Skills and interdisciplinarity gain importance.
  - Software Engineering teams need to master many more skills.
  - What are Software Engineering methods for managing fast-evolving technology zoos?
- Documentation: Incremental light-weight documentation needed.
  - How can we keep (incremental) documentation up to date?
  - Which documentation is needed?

Q&A

Marcus Ciolkowski
marcus.ciolkowski@qaware.de

twitter.com/qaware
linkedin.com/qaware
github.com/qaware
xing.com/qaware
slideshare.net/qaware
youtube.com/QAwareGmbH