Microservices und Eventgetriebene Architektur

Informatik BA, SS 2024

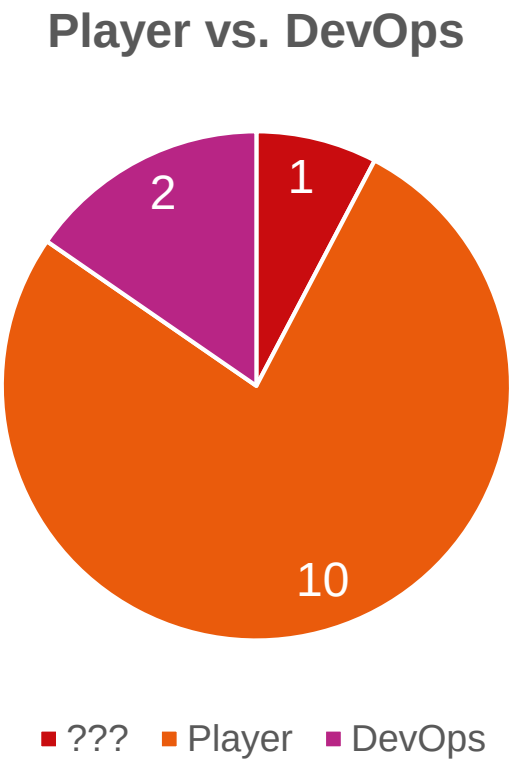
Prof. Dr.-Ing. Stefan Bente

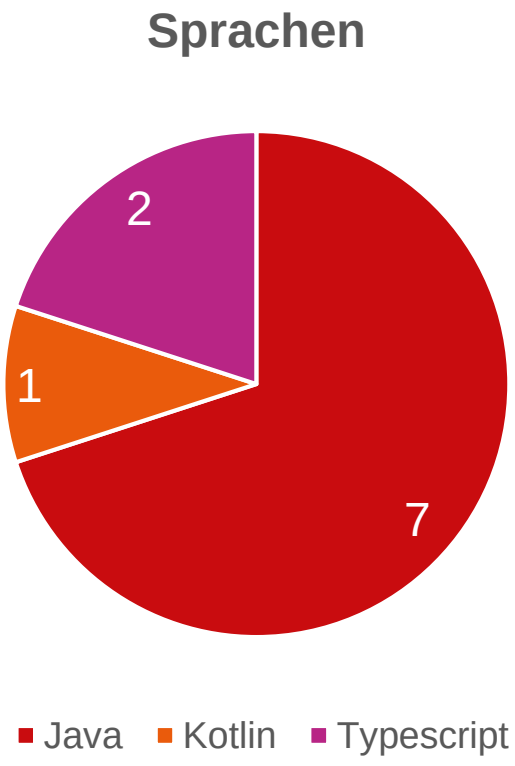
Workshop 2.5.2024

Technology
Arts Sciences
TH Köln

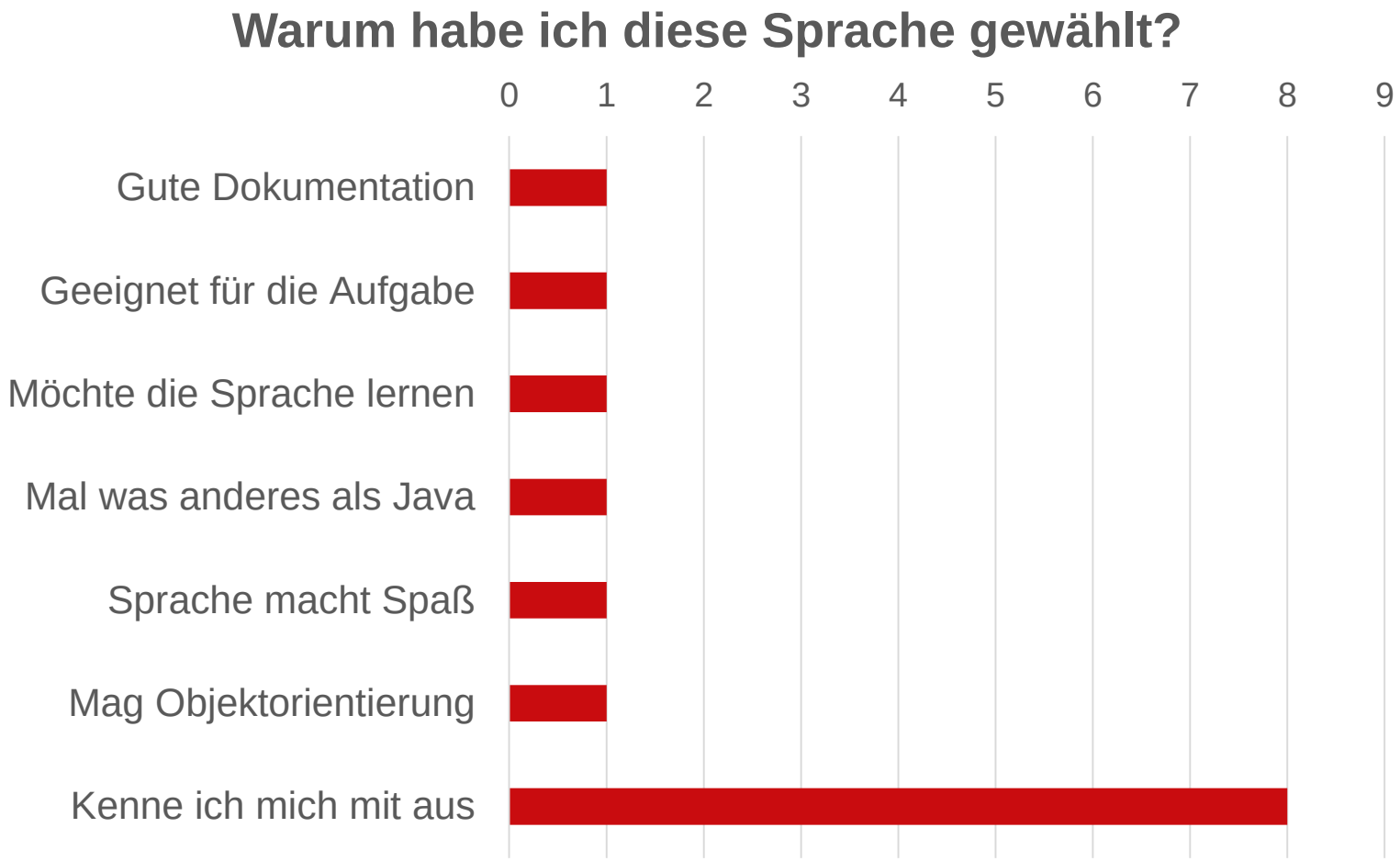
Ergebnisse der Umfrage

Aufteilung in der Veranstaltung

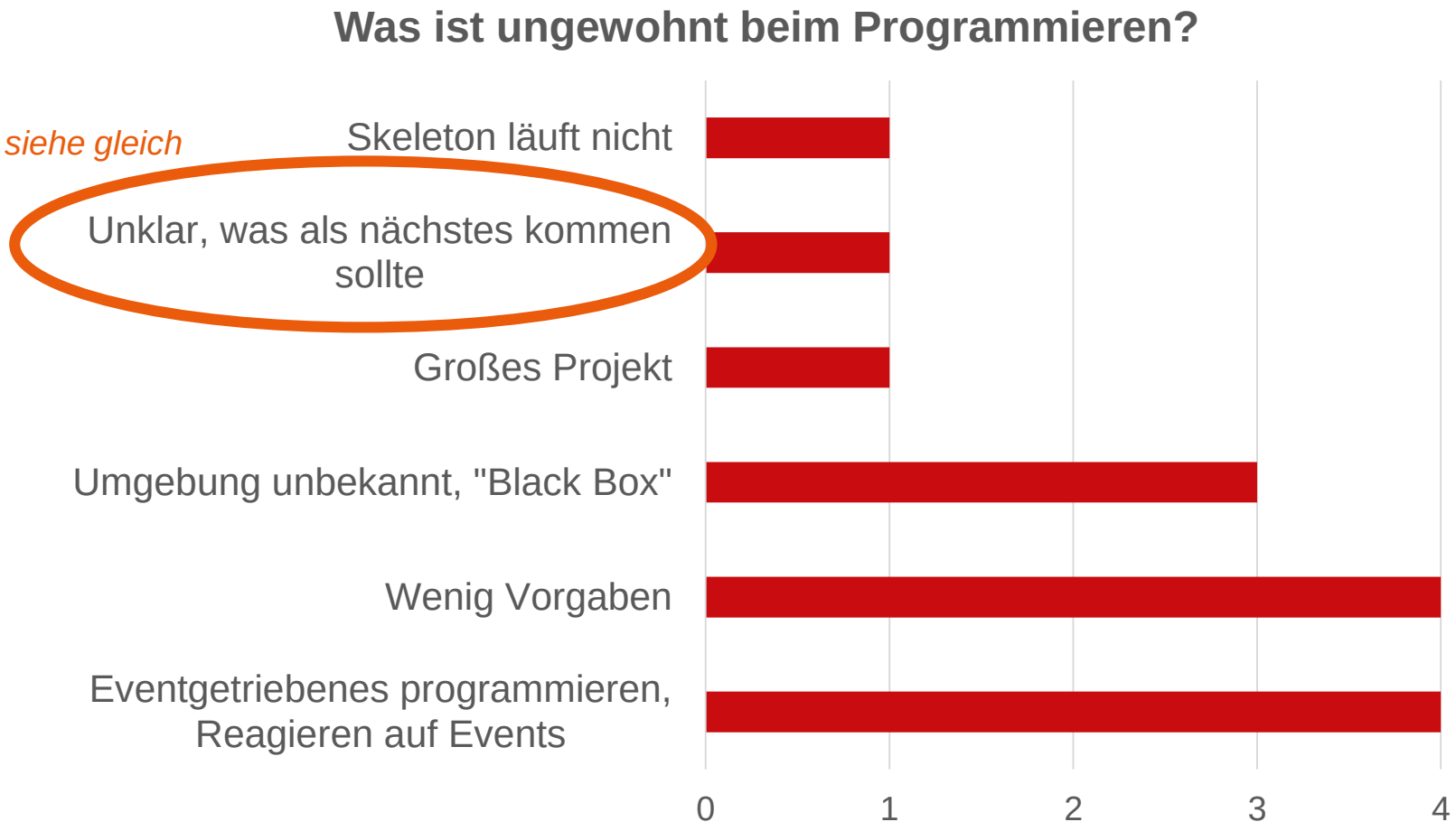




Motivation für Technologiewahl



Hürden beim Programmieren



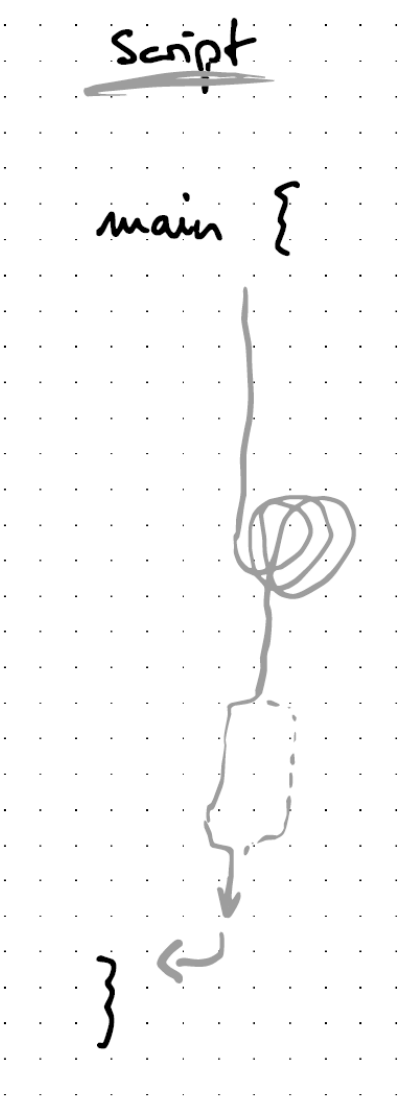
Offene Fragen
Anregungen für Input

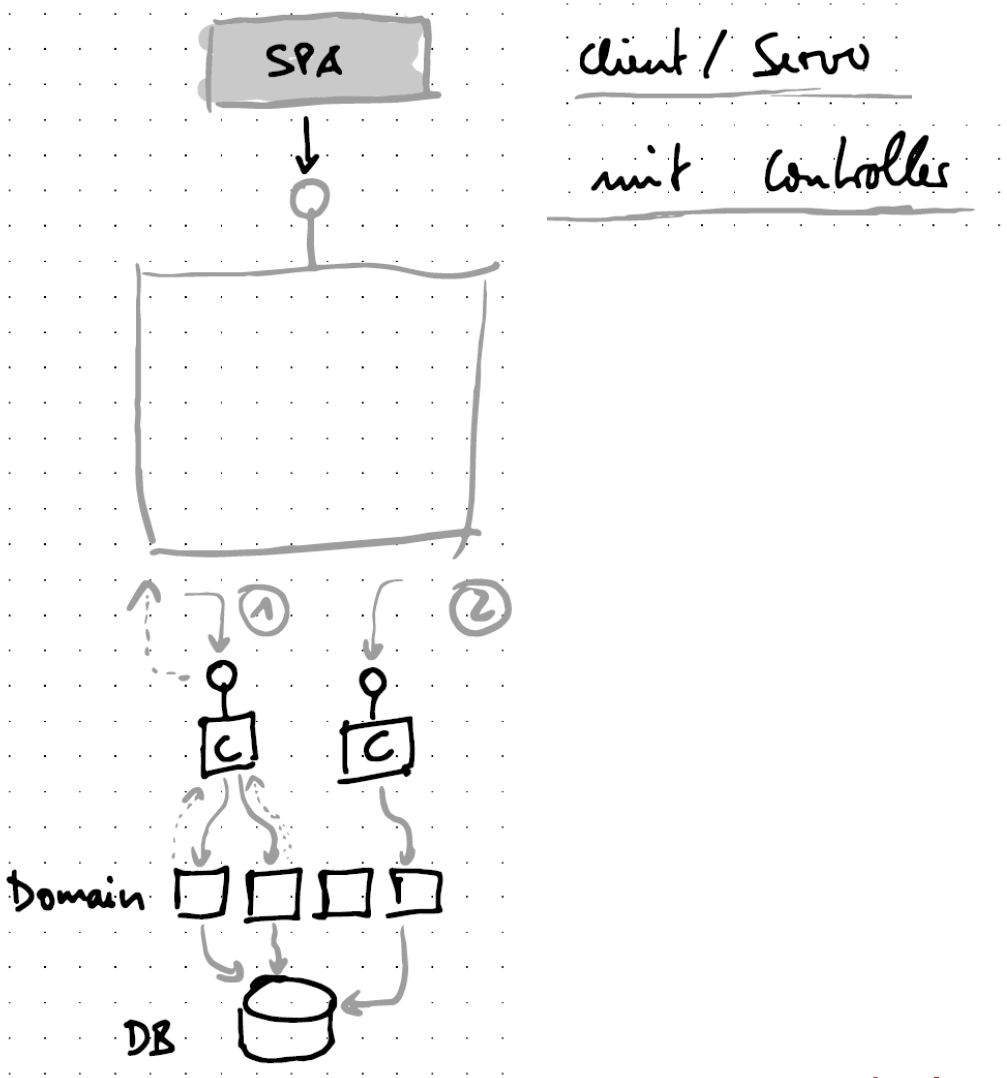
1) allgemein /
lässt sich schnell erklären

Allgemeines *(lässt sich schnell erklären)* **– (1)**

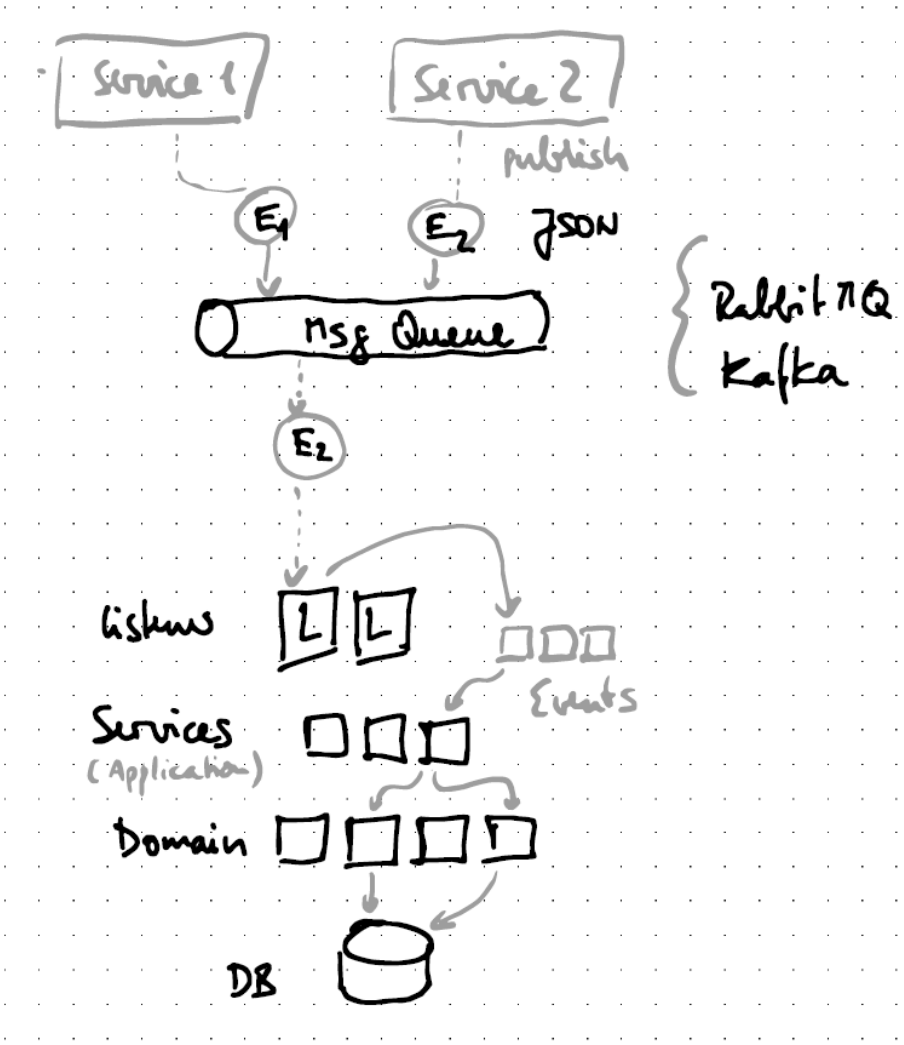
- Ich habe viel mehr Erfahrung, Server zu schreiben. Daher fällt es mir etwas schwerer, einen guten Player zu schreiben.

(Backend-)Server vs. Event-Driven Service (1)



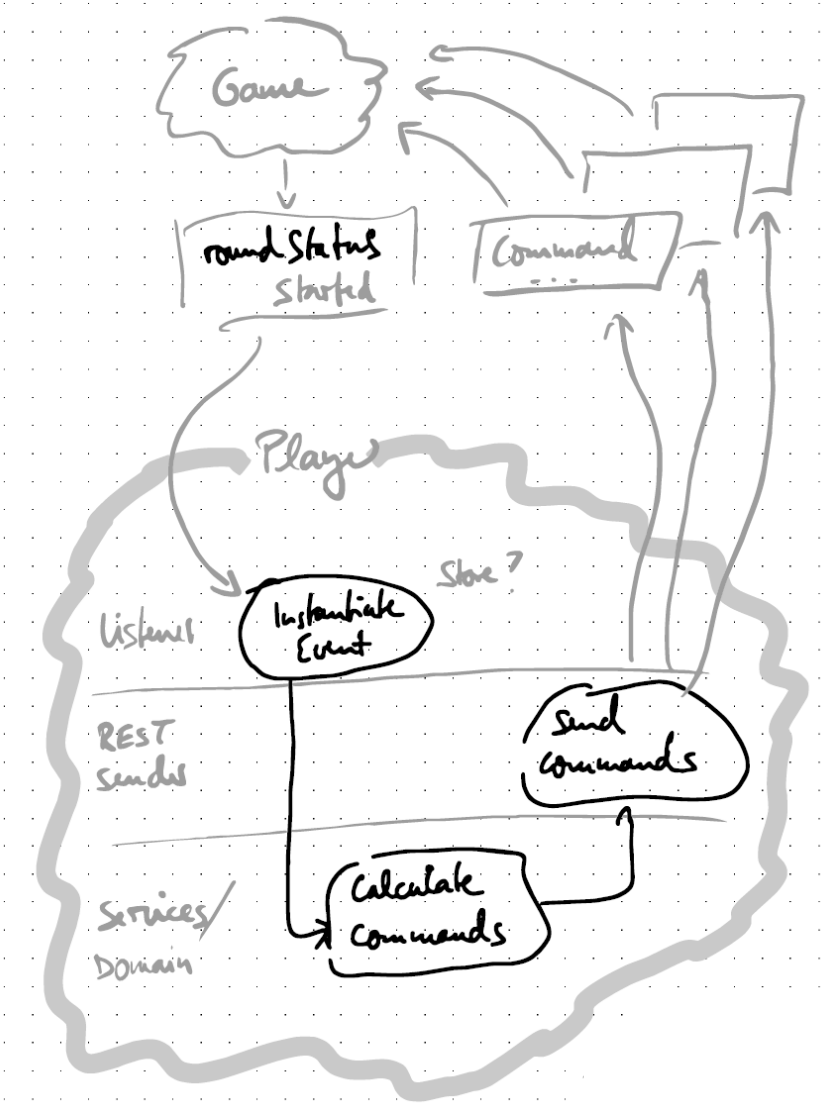


(Backend-)Server vs. Event-Driven Service (2)



Event - Driven Service

(Backend-)Server vs. Event-Driven Service (3)



Allgemeines *(lässt sich schnell erklären)* **– (1)**

- Der allgemeine Aufbau des Systems am lokalen Rechner, da ich mich für eine andere Sprache entschieden habe als die aus dem Workshop.
  - Ich habe da auch Probleme den Code auszuführen, was mich verwirrt, da ich unter der Annahme war, dass das Repository zumindest ohne manuell Fehler einzubauen funktionieren sollte.
  - Teils werden einige Sachen auch nicht richtig erklärt, aber da wieder das Problem, dass ich einfach eine andere Sprache gewählt hab als Java.

- Die im Skeleton bereits vorhandenen Domainprimitives, auch wenn ich weiß, dass ich diese für mich anpassen kann

Allgemeines *(lässt sich schnell erklären)* **– (2)**

- Die Dokumentation zum eigenen Setup eines Player-Skeletons ist sehr minimal.
  - Z.B. fehlt es an Informationen zum Setup via Message Queue zum lokalen Starten eines Spiels.
- Reale Use Cases
- Sonst komme ich bislang gut zurecht und empfinde die Dokumentation als gut, besonders für solch ein Projekt an einer Universität mit ständig wechselnden Studenten & Mitarbeitern

Offene Fragen
Anregungen für Input

2) Hosting & Deployment

Hosting & Deployment – (1)

- Im Moment wird alles über die local-dev-environment und auf örtlichen Entwicklungsumgebungen entwickelt.
  - Später soll der Player aber auch mit anderen interagieren können.
  - Wie wir uns später damit verbinden, bzw. wie wir dies später deployen sollen ist mir im Moment noch etwas unklar.

- Ich arbeite mit den DTOs aus dem Skeleton.
  - Im local-dev-environment läuft damit alles.
  - Muss ich noch etwas für später umstellen damit es läuft oder läuft alles, was im local-dev-environment läuft, auch später genauso?"

- Im Moment läuft meine Pipeline in Gitlab nicht.
  - Beim nachgucken im Skeleton läuft die Pipeline auch nicht.
  - Ist dies Absicht, bzw. wird dies vielleicht noch behandelt, oder stimmt noch etwas bei meinem Repo nicht?

Hosting & Deployment *(aus Anregungen für Input)* **– (2)**

- Nochmal kurz die Pipeline in Gitlab erwähnen (FALLS FÜR DAS PROJEKT ENTSCHEIDEND)
  - und vielleicht etwas zeigen wie man diese einstellen könnte / bzw eventuelle Empfehlungen vorstellen.

Offene Fragen
Anregungen für Input

3) Sinnvolle Entwicklungs-Schritte (Reihenfolge)

Offene Fragen: **Sinnvolle Entwicklungs-Schritte – (1)**

- Die Aufgabenstellung, so ein Leitfaden wäre ganz hilfreich z.B. erstes Ziel: Roboter auf einem Planeten mit Rohstoffen bringen, zweites Ziel: Roboter mining lassen

- Ich habe noch nicht verstanden, wie ich dynamisch (zwischen den Runden) reagieren und meine Strategie anpassen kann, da ja alles automatisch nach entsprechendem Event passiert.
  - Zum Beispiel beim Roboter kaufen, wie Sie es live programmiert haben, werden ja jede Runde versucht Roboter zu kaufen.
  - Sprich wenn man wieder Geld hätte würde man auch jedes mal wieder welche kaufen.
  - Da wäre die Frage, ob man einfach weitere Logik/Abfragen einbauen muss, um "dynamischer" zu reagieren, oder ob ich einen Aspekt nicht verstanden oder übersehen habe.

Anregungen für Input: **Sinnvolle Entwicklungs-Schritte – (2)**

- Wenn man damit nicht zu viel vorweg nimmt, sind weitere Beispiele immer sehr hilfreich, oder eben Herangehensweisen zur Strategie und Dynamik, wie ich es zuvor beschrieben habe ... wenn ich da nicht völlig auf‘m Schlauch stehe

- Die entscheidenden Punkte im Skeleton nochmal zeigen und eventuell ein Beispiel geben, die ich später zum praktischen Einsetzen meines Players brauche.

Vorschlag für Entwicklungsreihenfolge (1)

1. Buy robots
2. Move robots randomly
   - no notion of map yet
3. Enable robots to mine and sell resources
   - no search for high quality resources yet

You see something happening in the console

4. Make player ready to be deployed in the Kubernetes cluster

Your player can participate in the game

5. Develop an architecture to implement strategies
6. Implement a map

Your player has the foundation to act „smart“

Vorschlag für Entwicklungsreihenfolge (2)

7. Robot upgrades
8. Detect attacks by opponents, ability to run away
9. Attack yourself other players

Your player becomes „a force“

10. Your player makes use of the map
    - seeking out valuable resources
    - attacking and following weak opponents

Your player competes to win

11. Optimize your strategy

One ring to rule them all

Anregungen für Input: **Strategie**

- Manchmal die Spielmechaniken
  - welche Upgrades brauche ich für was, was bringen mir diese
  - wieviel Energie verbrauchen die Aktionen usw.
- Welche Strategien waren im letzten Jahr besser als andere?
- Vorschläge zu verschiedenen Strategien
- Vielleicht ein paar Tipps zur Strategie.

Thema für nächsten Workshop 23.05.: **Alumni-Panel**

Player-Strategie

Offene Fragen
Anregungen für Input

4) Architektur

Offene Fragen / Anregungen: **Architektur – (1)**

- Wie ich die Architektur meines Projektes bauen soll?
- Es würde helfen, wenn mir jemand bei dem Aufbau des Grundgerüsts helfen könnte, sodass ich mich nach und nach an die Struktur gewöhnen kann.
- Und welche Strukturen sich zum Speichern der Daten eher eignen.
  - (Ich arbeite viel mit Listen + Maps, aber eventuell wären weitere Entitäten oder neue Strukturen ähnlich zu den DTOs sinnvoll.)
- Andere Wege, die Projekt-Packages zu organisieren, als wir in ST2 gelernt haben

Alternative Formen der DDD-Schichtung:

Hexagonal Architecture

Onion Architecture

Clean Architecture

Hexagonal Architecture, Onion Architecture

- Hexagonal Architecture ist ein Konzept von Alistair Cockburn (1)
- Onion Architecture stammt von Jeffrey Palermo (2)
- Beide Konzepte sind in gewisser Weise „mehrdimensionale“ Varianten des DDD-Schichten-Konzepts
  - Sie stammen auch aus ähnlicher Zeit (DDD: 2003, HA: 2005, OA: 2008)
- Gemeinsamkeiten:
  - ringförmige Darstellung, statt gestapelter Schichten
- Haupt-Unterschiede:
  - Onion Architecture kennt Layers, Hexagonal Architecture nur Ports

- Im Weiteren stelle ich die Konzepte anhand eines Blogs von Herberto Graca vor, der eine sehr gute Infografik produziert hat (siehe nächste Folie)
- Quellen

(1) Cockburn, A. (2005). Hexagonal Architecture. https://web.archive.org/web/20180822100852/http://alistair.cockburn.us/Hexagonal+architecture, abgerufen 12.4.20

(2) Palermo, J. (2008). The Onion Architecture. https://web.archive.org/web/20180822100852/http://alistair.cockburn.us/Hexagonal+architecture, abgerufen 12.4.20

Hexagonal Architecture

- Hexagonal Architecture betont Ports & Adapters statt Layers
  - =~ „technische“ (Infrastruktur-) Aspekte des Application Layers
  - Adapter = explizite technische Controller-Elemente

- Application und Domain-Kern sonst ähnlich wie DDD

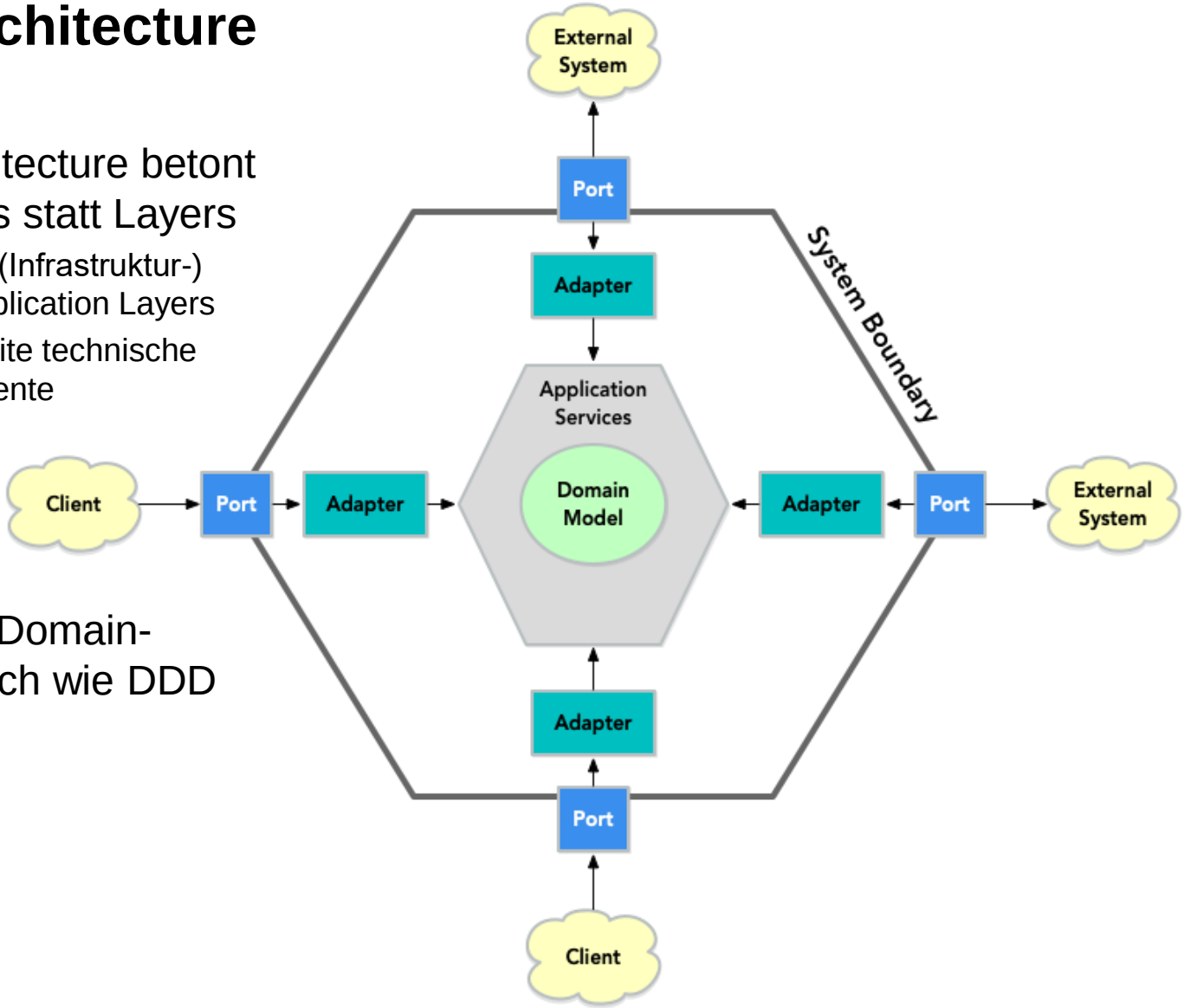


Bild: Holmström, P. (2020). *Domain-Driven Design and the Hexagonal Architecture*. Vaadin. https://vaadin.com/learn/tutorials/ddd/ddd_and_hexagonal

DDD / Hexagonal Architecture / Onion Architecture

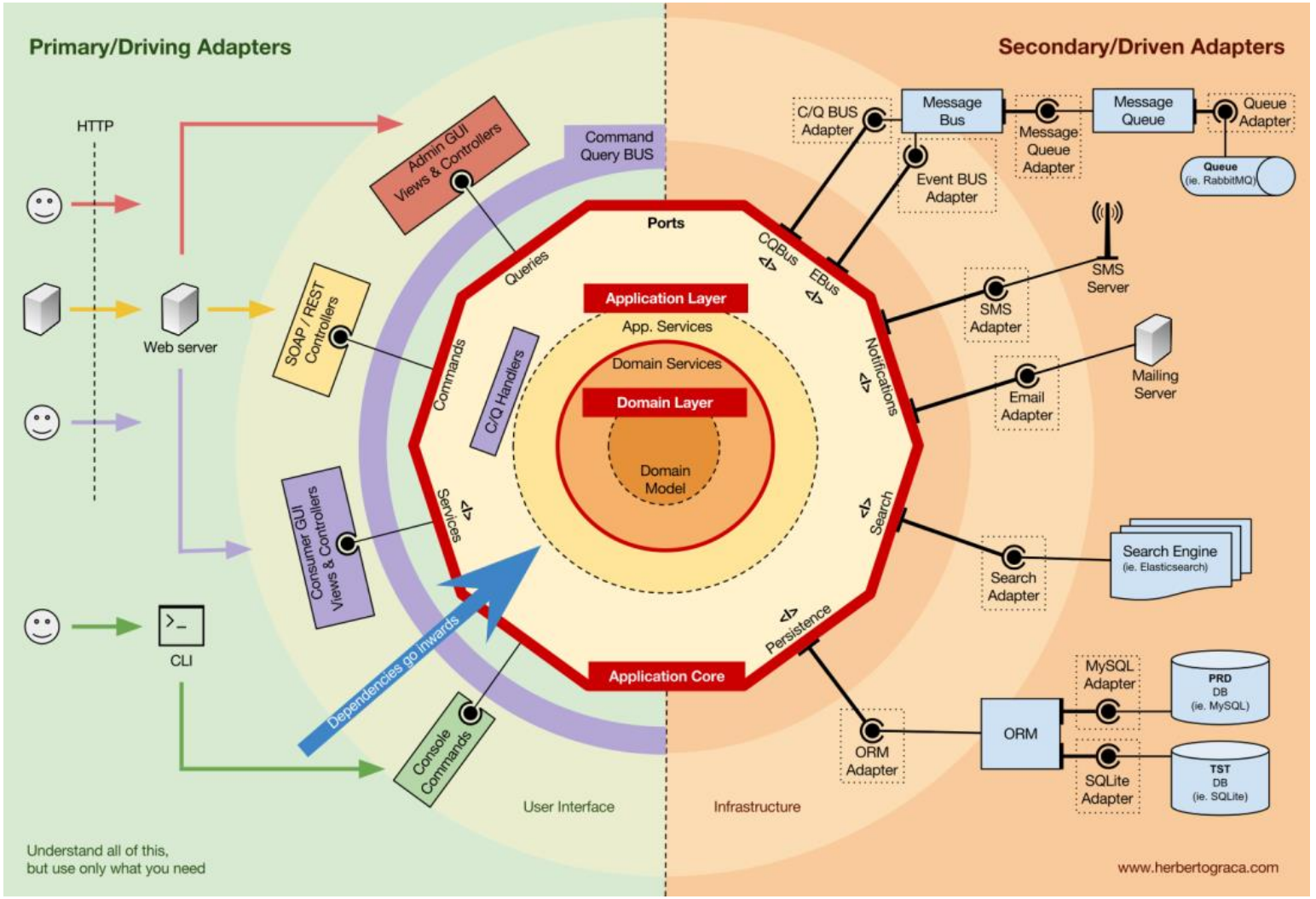


Bild: Graca, H. (2017, November 16). DDD, Hexagonal, Onion, Clean, CQRS, … How I put it all together. *@hgraca*. https://herbertograca.com/2017/11/16/explicit-architecture-01-ddd-hexagonal-onion-clean-cqrs-how-i-put-it-all-together/

Clean Architecture (Bob Martin)

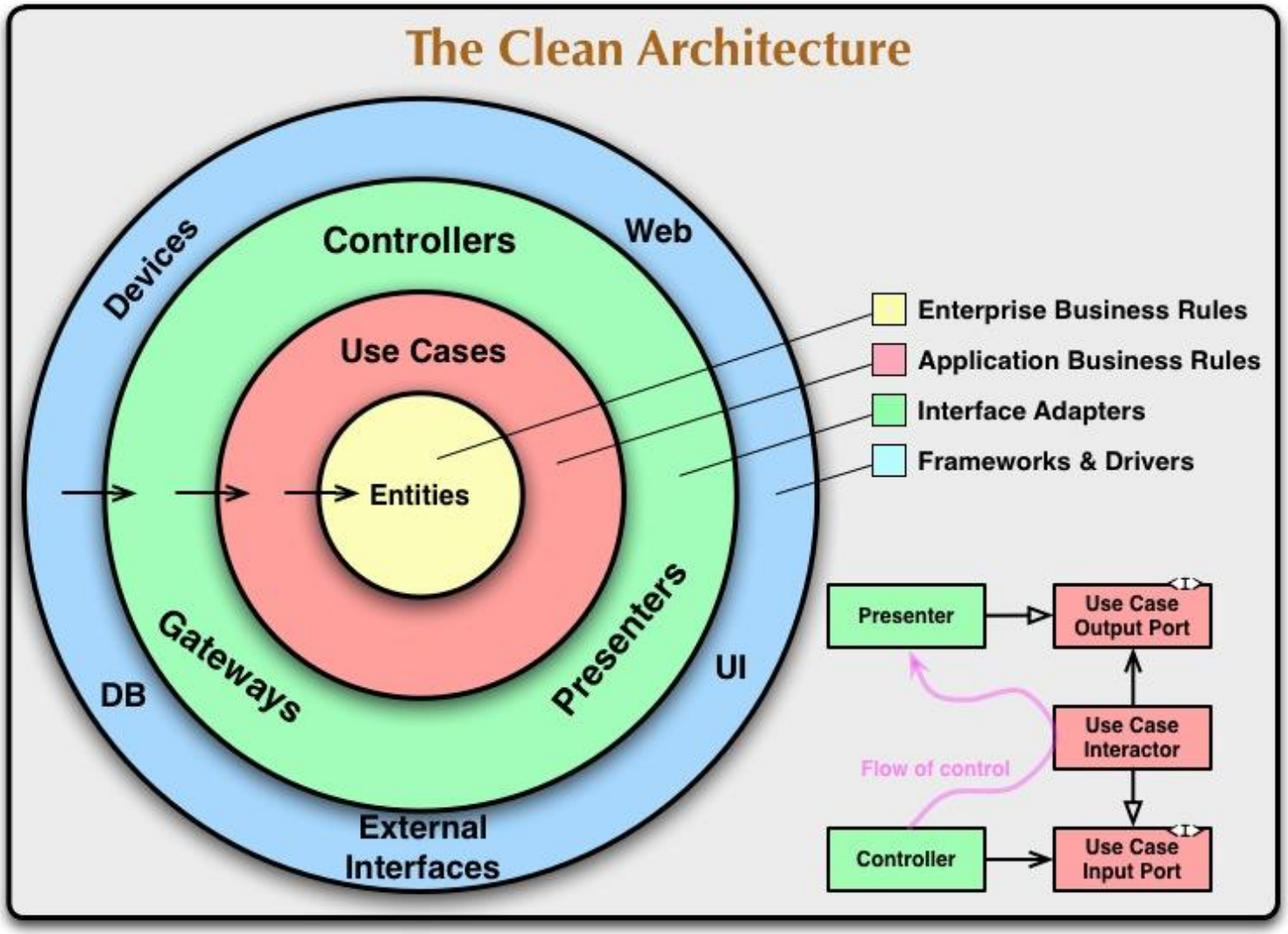
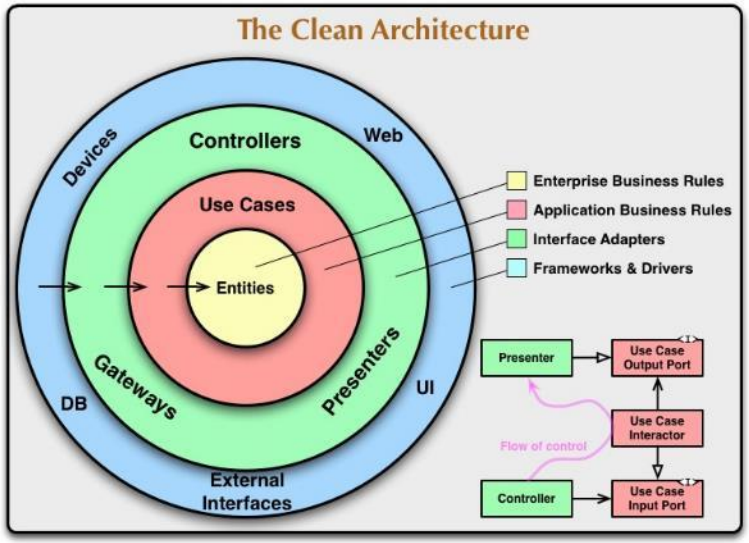


Bild: Martin, R. C. (2012, August 13). The Clean Architecture. *Clean Coder Blog*. https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

Clean Architecture (Bob Martin)

- Clean Architecture hat eine ähnliche ringförmige Struktur wie Hexagonal Architecture und Onion Architecture
- Vergleich zu DDD:
  - Der **äußere blaue** Ring repräsentiert (wie bei HA und OA) eine Mischung aus Presentation und Infrastructure Layer
  - Der **grüne** und der **rote** Ring entsprechen dem Application Layer
    - Use Cases entsprechen Application Layer Services (siehe später in dieser Vorlesung)
  - Der **gelbe** Kern entspricht dem Domain Layer



Offene Fragen / Anregungen: **The Big Picture – (1)**

- Best Practices zu Event-Driven-Architecture
  - (ähnlich zu ST 2)
- Der Vergleich mit anderen Architekturen (MVC, SOA, REST, Monolitith etc.)
- „Most common“ Fehler zeigen und wie man die löst
- Vor- und Nachteile der Architektur
- Common Pitfalls

Common Pitfalls / Challenges

- Performance
  - Überschreitung der Rundenzeit im Player fällt erstmal nicht auf
    - Player schickt dann „wenig sinnvolle“ Kommandos weg
  - ggfs. auf DB verzichten
  - Repository-Pattern bleibt sinnvoll, kann man aber gut auch ohne DB implementieren

- Mapping externe => interne Events
  - Wie stellt man sicher, dass eine gewisse Reihenfolge eingehalten wird?
  - Beispiel: RobotsRevealedEvent
    1. Player aggregate: gibt es einen neuen Enemy player?
    2. Robot aggregate: speichere eigene & enemy robot Positionen
  - Spring Eventing: sync vs. async
    - Sync: erlaubt Reihenfolge (@Order(1), @Order(2), …)
    - keine gute Lösung
    - besser: Eventual Consistency

Thema für nächsten Workshop 23.05.: **Alumni-Panel**

Dungeon Do‘s & Don‘ts