ABB Technikerschule

Technik. Informatik. Wirtschaft. Management →

Dominik Meyer
Software Engineering

(Komponenten) Architektur

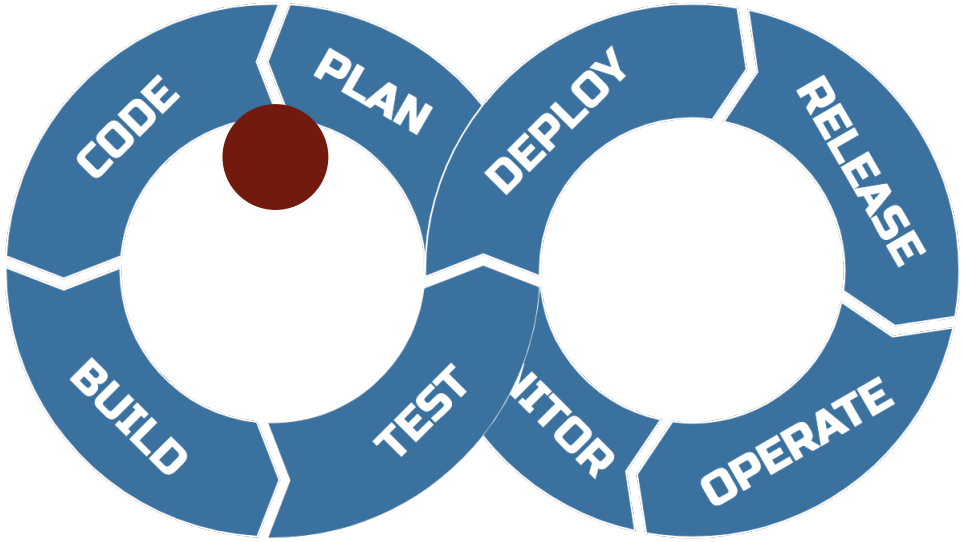
Ziel

Nach der Lektion wenden die Studierenden die *Design Considerations* korrekt an und können diese in ihre Applikation transferieren.

Nach der Lektion haben die Studierenden das Unterrichtsbeispiel *Jinder* nach als Schichtenmodell entworfen.

(Komponenten) Architektur

- Geschäftsprozesse- und Fälle
  - RME
- High Level Architektur gewählt
- Design
  - Umsetzung der Prozesse und Fälle planen
  - Anforderungen einplanen und reflektieren



- Wir graben uns weiter bis zum Code
  - *Software Engineering heisst Denken, nicht nur drauflos coden*

Agenda

- Ziel
- Design Considerations
- Layered → Hexagonal Design
- Zielkontrolle

Design Considerations

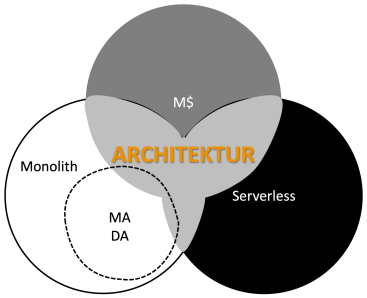
- Compatibility - With others or an older self
- Extensibility - Adding capabilities without major changes
- Maintainability - Documentation, DevOps, Transparency, clean
- Modularity - isolated components, exchangeable, division
- Reliability - functions under load for a specific time
- Reusability - Don’t repeat yourself DRY, reuse in other designs
- Robustness - Operate under stress / faulty data, fail graceful
- Security - Withstand hostile acts and influence, keep data safe
- Usability - Emotions, help, experience, journeys

- Design != Architektur, jedoch ist klar
  - Ungeeignete/schlechte Architektur kann kein gutes Design fördern
- Das können wir nicht von heute auf morgen
- Best practises, Reviews, Erfahrungen
  - Junior, Senior, Lead…

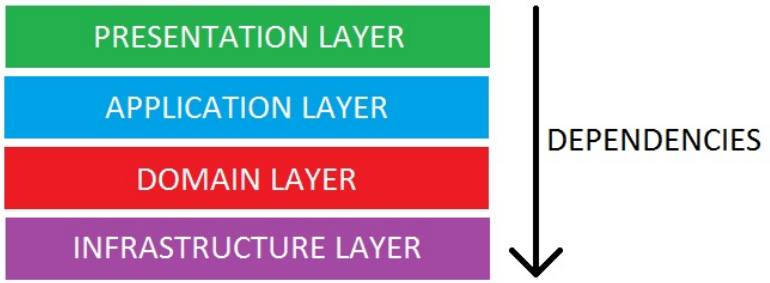
Architekturen

- Es gibt ± 7 Architekturen
- Diese sind kombinierbar, es gibt nicht **DIE EINE**
  - (wieder nicht 😊)
- Alle haben ihren Verwendungszweck
- Im Detail schauen wir an
  - Schichtenmodell
  - Hexagonal Architecture
  - sind die in MA, DA meist-genutzten

- Die restlichen im Anhang



Schichtenmodell / Layered Architecture



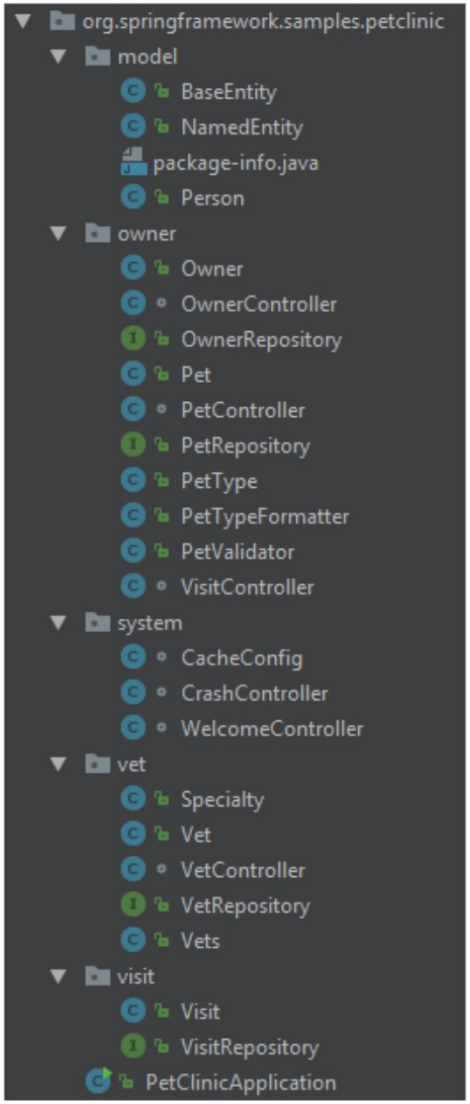
- Common, typically used with databases
- Layers allow devs to work on separate layers independently
- Data travels from outside inwards and back out

- ❑ *New applications*
- ❑ *Applications that mirror some business that is layered too*
- ❑ *Inexperienced teams that do not understand other architectures*
- ❑ *Maintainability - and Testability - focused applications*

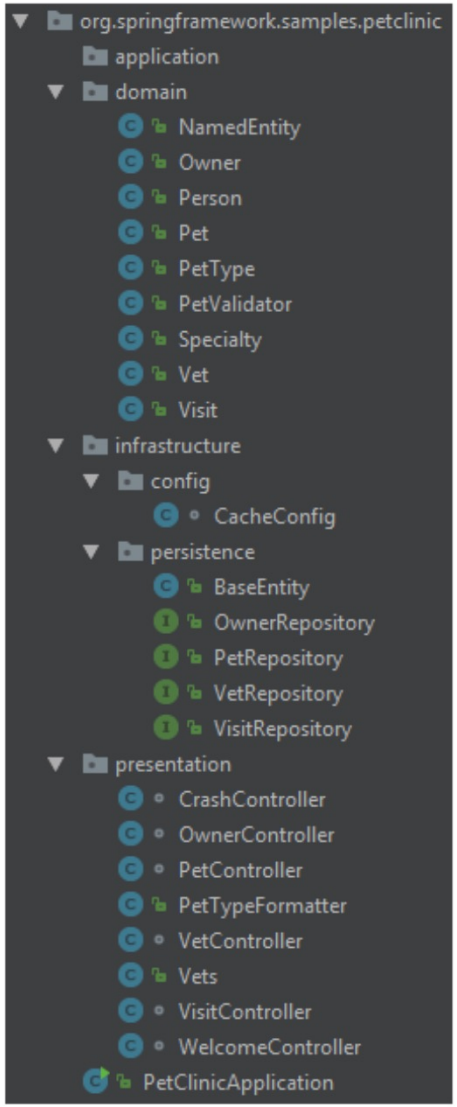
+ Maintainable
+ Testable
+ Easy to assign “roles”
+ Easy to update and enhance any layer

– Source code mess if not well done
– Slow code due to “handing down logic”
– Isolation can make it hard to understand
– Often results in monolithic deploys
– Coders can skip layers → Coupling

Schichtenmodell / Layered Architecture



Spring Pet Clinic Original Architecture



Spring Pet Clinic with Layered Architecture

https://dzone.com/articles/layered-architecture-is-good

Schichtenmodell / Layered Architecture

+

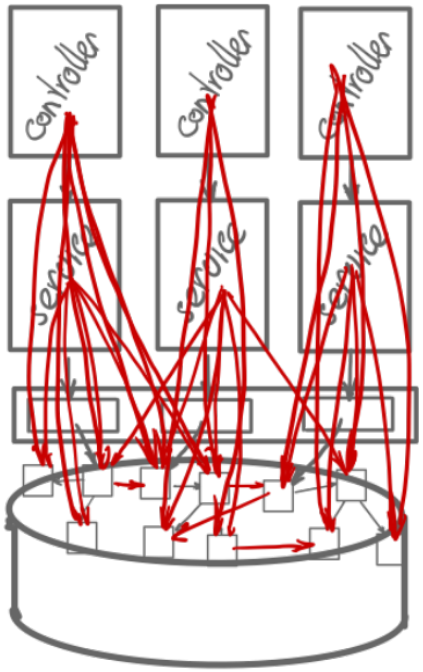
- Einfachheit (kleine Lernkurve von ALG / OOP)
- Konsistent (gut für kleine Gruppen)
- Saubere Trennung von Zuständigkeiten (Code und Personen)
- Dinge sind einfach zu finden

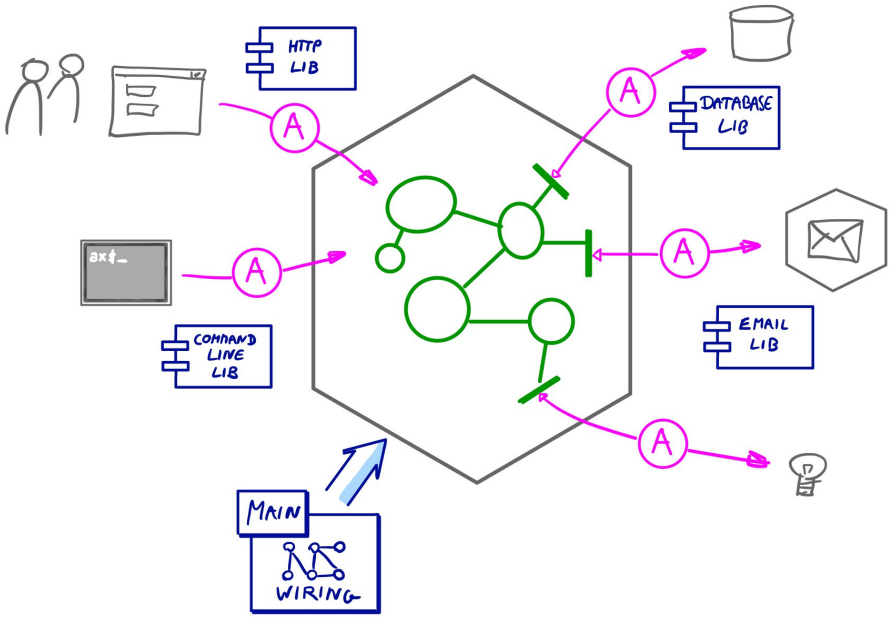
−

- Skaliert schlecht (wird ein Chaos)
- «Use Cases» der App nicht einfach ersichtlich
- Schwache Kohäsion (Logik kann weit verteilt sein)
- u.U. schwache Abhängigkeitsisolation (Dependency Inverstion)

https://dzone.com/articles/layered-architecture-is-good

Hexagonal Design/Architecture





https://www.qwan.eu/2020/08/20/hexagonal-architecture.html

Hexagonal Design/Architecture

+

- Fokus auf «unsere» Logik
- Einfaches «Was gehört wohin» (→ Lektion 22+ Design)
- Einfachere, fokussiertere Tests
- Klarer Fluss der Abhängigkeiten, Reduktion deren
- Unterstützt einfache Erweiterung des Domain Model

−

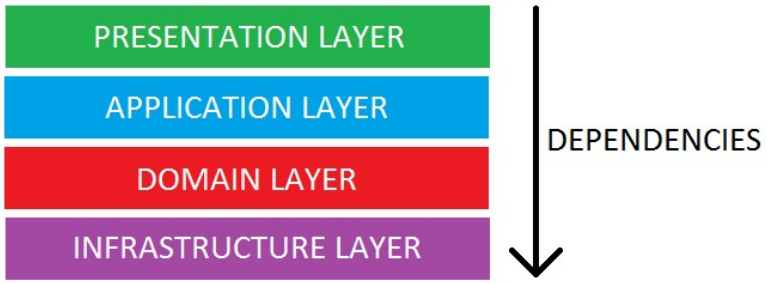
- Adapter schreiben kann repetitiv werden
- Impl. eines Features kann heissen diverse Orte berühren
- Isolation der Domain nicht immer einfach (Leak)
- Nimmt nicht alle Entscheidungen ab, z.B. Wie wir den Adapter implementieren

https://www.qwan.eu/2020/08/20/hexagonal-architecture.html

Hexagonal Architecture Untersuchung

- Kompilat von SOLID und anderen Design Prinzipien (L22+)
- “Weiterentwicklung” des Schichtenmodell
- Hexagonal Architecture ist der Urahne von “Clean Architecture”

- 🔑 sind immer die Schichten, egal in welche Richtung diese ”stapeln”
  - respektive deren Abhängigkeiten
  - deren Modularität
  - deren Abhängigkeiten



Task

Geschichtetes *Jinder*

AUFTRAG
⏰ 20min

/docs/tasks/bites/architecture

Iterieren Sie unser Jinder Modell, Schichten Sie dieses.

Zusammenfassung

- Architektur ist nicht einfach zu «trennen»
- Ist nicht 100% das gleiche wie Software Design
- Hat kein Handbuch was dazu gehört und was nicht
  - Wesentliche
  - Wichtigste
  - Intellectual Property
- Best Practises u.a. mit Mustern
- Erfahrung
  - Austausch, Senior Developer, Reviews, etc.

MA: Fachliche «Richtigkeit» wird weniger gewichtet, als ihre Erklärungen und Verfolgbarkeit ihrer Entscheidungen zu Ergebnissen aus RME oder den «oberen» Schichten der Architektur

Zielkontrolle

AUFTRAG
⏰ 5min

https://forms.office.com/r/2R93fQdj57

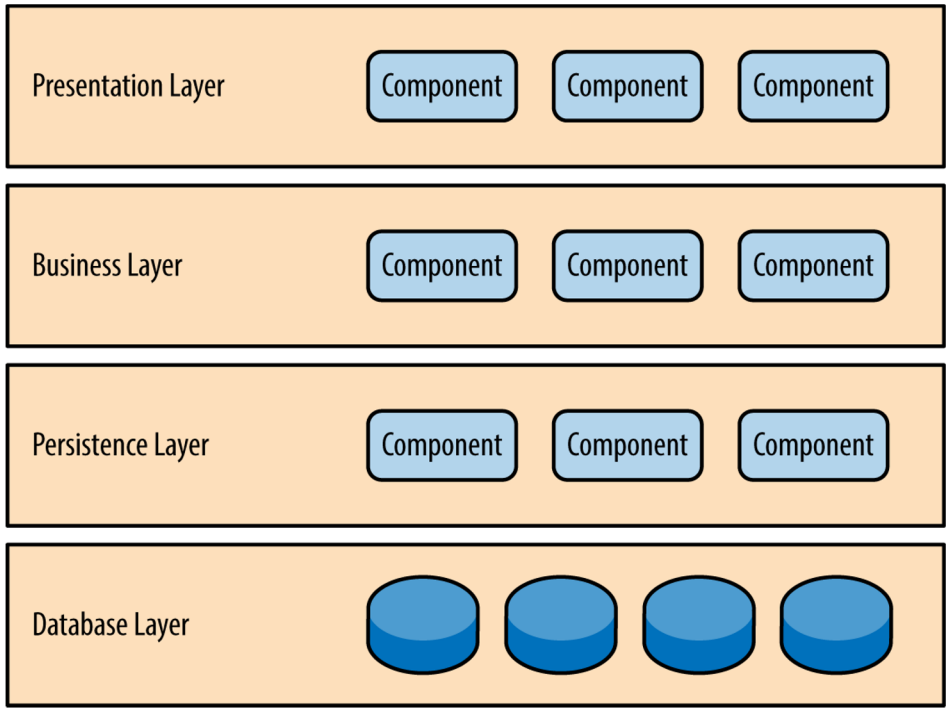
🔐 ABB Login

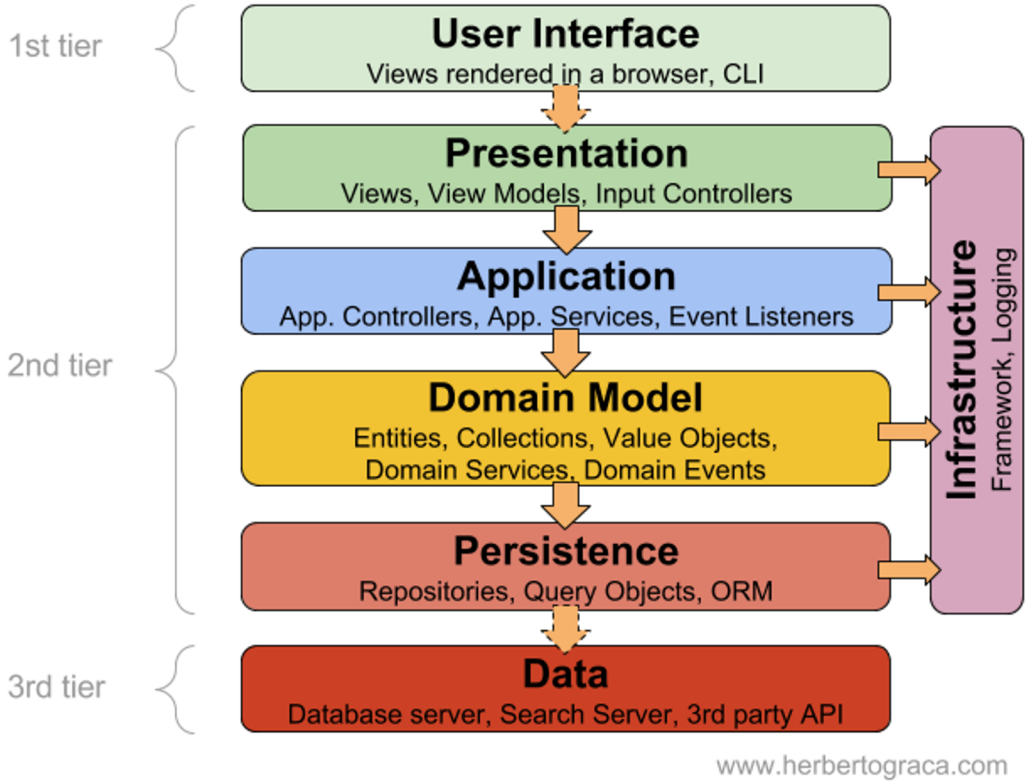
WEITER WISSEN.

Wir begleiten Sie!

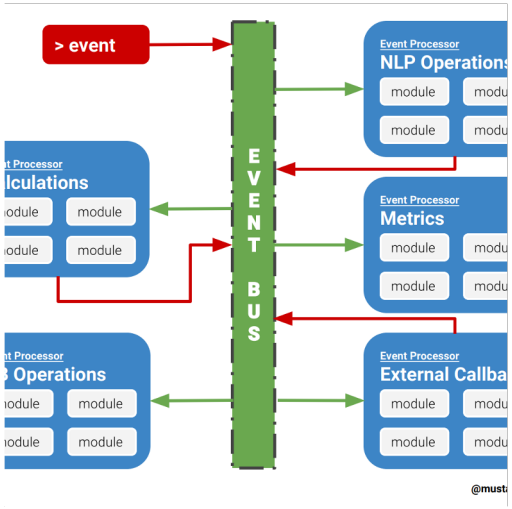
Nachfolgend Slides aus früheren Vorlesungen

als ergänzende Lektüre


Presentation Layer
Component
Component
Component
Business Layer
Component
Component
Component
Persistence Layer
Component
Component
Component
Database Layer


1st tier
User Interface
Views rendered in a browser, CLI
2nd tier
Presentation
Views, View Models, Input Controllers
Application
App. Controllers, App. Services, Event Listeners
Domain Model
Entities, Collections, Value Objects,
Domain Services, Domain Events
Persistence
Repositories, Query Objects, ORM
Infrastructure
Framework, Logging
3rd tier
Data
Database server, Search Server, 3rd party API
www.herbertograca.com

Event driven architecture

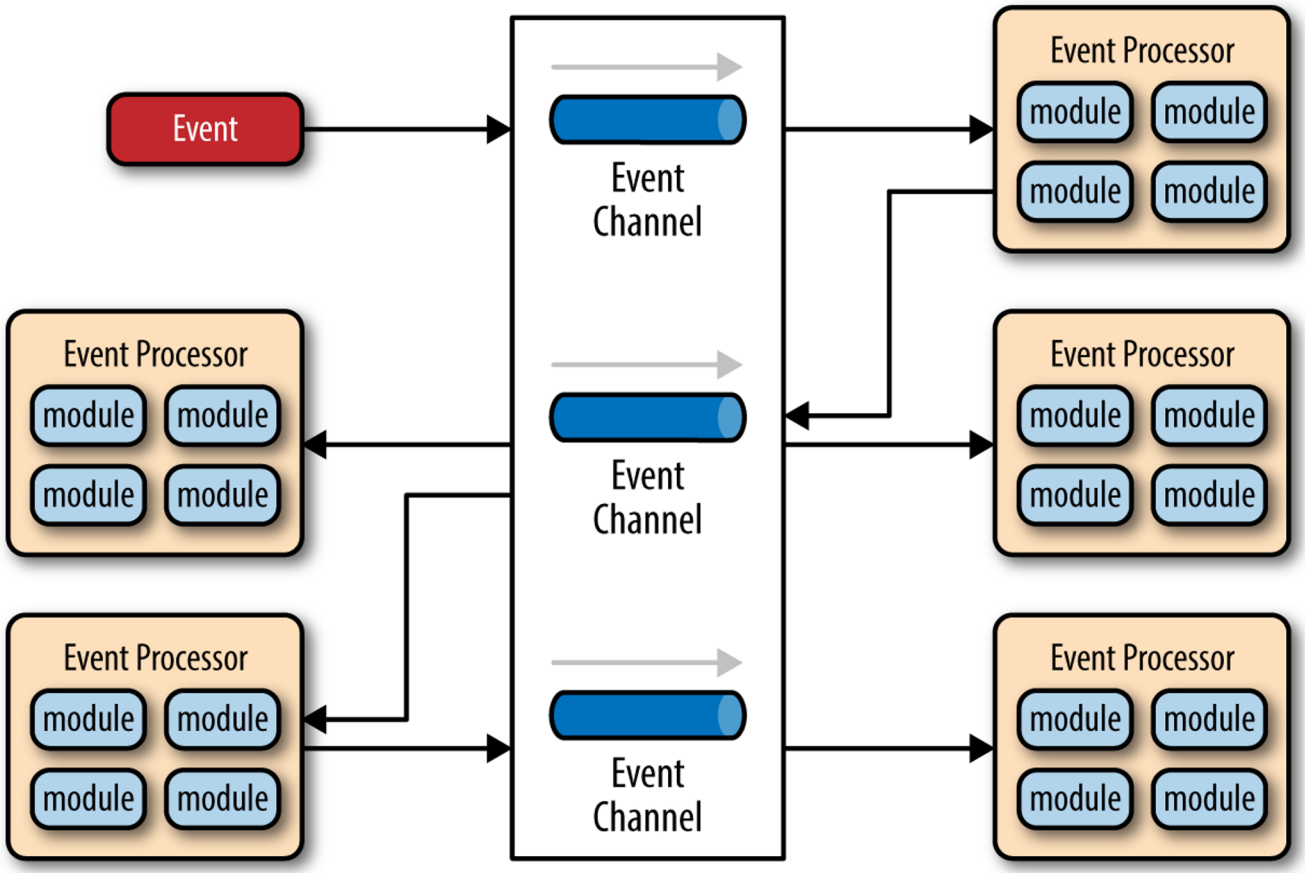


- Central unit accepts data and generates event
- Only specific modules react to certain events
- Good example: Events in Javascript (Browser, click, subscribers)
- Very different from layers because only subscribers react

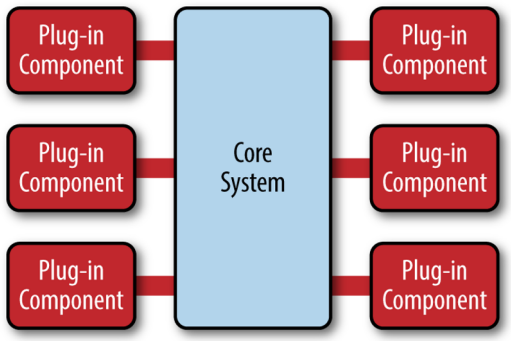
❑ *Asynchronous systems with asynchronous data flow*
❑ *Where individual data only interacts with few modules*
❑ *User interfaces (because non blocking)*

\+ Suitable for complex and “chaotic” environments
\+ Scale easily and fast
\+ Extendable whenever a new event type appears

\- Difficult testing and error handling
\- One of n units can fail…
\- Messaging overhead affects speed
\- Transaction based operations difficult
\- Unifying events for different needs


Event
Event Channel
Event Channel
Event Channel
Event Processor
module
module
module
module
Event Processor
module
module
module
module
Event Processor
module
module
module
module
Event Processor
module
module
module
module
Event Processor
module
module
module
module

Microkernel architecture



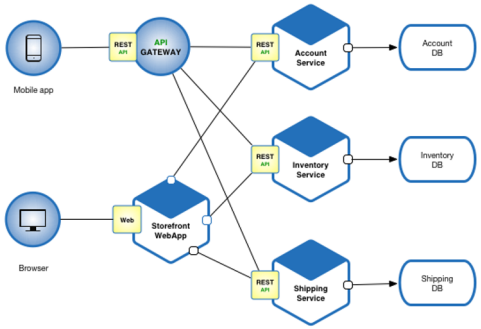
- Basic tasks go into a kernel (usually small)
- Developers develop additional logic as “plug-ins”
- Sometimes called plugin-architecture

❏ *Tools used by huge groups / variety*
❏ *Applications where division of basic/advanced is clear*
❏ *When core is routinely fixed but surrounding changes frequently and dynamically*

\+ Very modular and “agile”
\+ Testable
\+ Lightweight, high performance
\+ Easy to deploy, hot-deploy

\- Deciding kernel features is an art
\- Handshake between kernel and module
\- Modifying the kernel difficult/impossible
\- Kernel granularity difficult to choose

Microservice architecture



- Instead of building one big program, small ones are built
- Netflix UI, every component has its service, why?
- Similar to microkernel and event architecture
  But used when tasks can be completely separated

❑ *Websites with small components*
❑ *Datacenters with well defined boundaries*
❑ *Rapidly developing new businesses and applications*
❑ *Spread out development teams*

\+ Super modular
\+ Scaling capabilities
\+ Allows max. Development agility
\+ Buzzword (Caution)

\- Independency is tricky, Balance is needed
\- Not all tasks can/should be split
\- Performance can drop due to overhead
\- Delay in some services can confuse users

More on architecture

Term	Short summary	Common examples (or part of)
Monolithic	A single huge structure that does everything	Word, Excel
Peer-to-Peer P2P	Equal right peers distribute the tasks/load between them	BitTorrent
REST / SOA (REST is an approach to SOA)	Mostly used in combination with others, simplify communication	Wordpress backend, any service provider basically (Netflix, Uber, Airbnb….)
Rule-based	Store and manipulate knowledge in useful ways	Machine learning, airline seat pricing, online mortgages pricing