

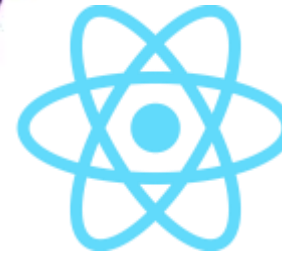
Software Architektur ist einfach, oder?



Thorsten Kansy (tkansy@dotnetconsulting.eu)

Meine Person- Thorsten Kansy

Freier Consultant, Software Architekt,
Entwickler, Trainer & Fachautor



Azure Cosmos DB



Mein Service- Ihr Benefit

- Individuelle Inhouse Trainings
- (Online on-demand) Projektbegleitung
- Beratung
 - Problemanalyse und Lösungen
 - Technologieentscheidungen



Agenda

- Einstieg
- Muss Architektur komplex sein?
- Nutzen durchdachter Architektur
- Prinzipien und Best Practices
- Kleine Projekte – lohnt sich das?
- Zusammenfassung & Takeaways



Einstieg

Einstieg

- Architektur = bunte Kästchen und Buzzwords?
- Architektur != gewachsene Strukturen
- Eine Pflichtübung? Weil es alle so machen?
- Oder für das Poster im Flur?



Kompliziert oder komplex?

Komplex und mysteriös?

- Kompliziert vs. Komplex
 - Kompliziert = schwierig, aber berechenbar
 - Komplex = unvorhersehbar, nur beobachtbar
- Architektur als Selbstzweck („Architektur-Elfenbeinturm“)
- Beispiele aus der Praxis:
 - Repository-Pattern trotz EF Core
 - „Vielleicht tauscht man ja mal SQL Server gegen Oracle“
 - ...
- Architektur soll verständliche Entscheidungen ermöglichen



Vorteile einer durchdachten Architektur

Vorteile einer **durchdachten** Architektur

- Wartbarkeit: Weniger Aufwand bei Erweiterungen
- Skalierbarkeit: Lastspitzen ohne Panik
- Langlebigkeit: Systeme über Jahre hinweg nutzbar
- Teamkommunikation: Gemeinsames Vokabular und klare Zuständigkeiten
- Einarbeitung neue Teammitglieder einfacher
- Spart Ressourcen



Wiederverwendbarkeit

Wiederverwendbarkeit?

- Entsteht nicht automatisch, auch nicht mit der richtigen Architektur
- Gibt es nicht zum Nulltarif
- Wiederverwendbarkeit bedeutet
 - Mehr Entwicklungsaufwand
 - Mehr Testaufwand
 - Laufenden Pflegeaufwand



Nebenschauplatz: Design-Pattern

Design-Pattern

- Kein Ersatz für Architektur, kommt aber oft in dessen Fahrwasser daher
- Architektur = „Stadtplan“ vs Design Patterns = „Bauweisen“
- Wieviele gibt es den? ChatGPT sagt

👉 Eine exakte Zahl gibt es nicht. Wenn du nach einer runden Kenngröße fragst:

- 23 (GoF) = Kern
- ~100 = allgemein gebräuchlich in C# und Enterprise-.NET
- >300 = wenn man akademische, Nischen- und Architekturmuster mitzählt

- Aber...

Design-Pattern

...es gibt Überschneidungen

Family	Category	Pattern	One-line description
Architecture	Layering	Clean Architecture	Dependency rules center around use-cases and boundaries
Architecture	Ports & Adapters	Hexagonal Architecture	Isolate domain from external actors via ports
Architecture	Layering	Onion Architecture	Domain-centric rings with outer infrastructure layers
Architecture	Service Decomposition	Microservices	Small autonomous services around business capabilities
Architecture	Monolith	Modular Monolith	Single deployable with clear internal module boundaries
Architecture	UI	MVVM	Model-View-ViewModel separation for testable UIs
Architecture	UI	MVP	Model-View-Presenter with passive view pattern
Architecture	Integration	SOA	Services communicate via standardized contracts and messages
Architecture	Integration	Event-Driven Architecture	Asynchronous events drive reactions across components
Architecture	API	API Gateway	Single entry point routing to internal services
Architecture	API	BFF	Backend tailored to a specific frontend or device
Architecture	Data	CQRS	Separate read and write models for scalability
Architecture	Data	Event Sourcing	Persist domain events then rebuild state when needed
Architecture	Data	Outbox Pattern	Ensure reliable message publish with local transaction
Architecture	Workflow	Saga / Process Manager	Coordinate distributed transactions with compensations
Architecture	Decomposition	Strangler Fig	Incrementally replace legacy by routing and carving out
Architecture	Runtime	Pipes and Filters	Compose processing steps as independent filters
Architecture	Deployment	Sidecar	Co-locate helper container alongside application container
Architecture	Networking	Ambassador	Offload cross-cutting networking concerns to sidecar
Architecture	Edge	Circuit Breaker	Fail fast and recover when downstream is unhealthy



Architektur-Prinzipien

Prinzipien

- Separation of Concerns – Aufgaben klar trennen
- Kopplung minimieren, Kohäsion maximieren – keine Spaghetti
- KISS (Keep it simple, stupid) – unnötige Abstraktionen vermeiden
- YAGNI (You aren't gonna need it) – nicht für die Zukunft über-engineerern

SOLID-Prinzipien

- **S**ingle responsibility principle
- **O**pen-closed principle
- **L**iskov substitution principle
- **I**nterface segregation principle
- **D**ependency inversion principle

Single responsibility principle

„Es sollte nie mehr als einen Grund geben, eine Klasse zu ändern.“

(Robert C. Martin)

Eine funktionale Einheit (Klasse, Funktion, Variable, etc.) darf nur für eine Aufgabe verantwortlich sein.

R.I.P Eierlegende Wollmilchsau

 Demo 

Open–closed principle (OCP)

„Module sollten sowohl offen (für Erweiterungen) als auch verschlossen (für Modifikationen) sein.“

(Bertrand Meyer)

Konkreter Code vs. abstrakter Code

Lösungsansätze

- Parameter
- Vererbung
- Composition/Injection

 Demo 

Liskov substitution principle

- Ein Funktion, die mit einem Type arbeitet, muss auch mit einem ihrer Untertyp arbeiten.
- In C# eingebaut durch Vererbung und Polymorphismus
- Eine Aushebelung via Code ist aber denkbar, aber keine gute Idee
 - `is` - / `as`-Operator, o.ä.

 Demo 

Interface segregation principle

„Schnittstellen sollte nur so umfangreich sein, wie nötig.“

- C#-Interfaces
- C#-Klassen

Notfalls aufteilen

 Demo 

Dependency inversion principle

„Module höherer Ebenen sollten nicht von Modulen niedrigerer Ebenen abhängen.“ (Wikipedia)

Abstraktionen durch

- Schnittstellen
- Abstrakte (Basis-)Klassen
- Pattern
 - IoC
 - MEF

 Demo 



Architektur im Kleinen – lohnt es sich?



Architektur im Kleinen – lohnt es sich?

- „Für kleine Projekte brauche ich keine Architektur!“
- Auch kleine Projekte werden schnell größer
- Es gibt leichtgewichtige Ansätze

Lösungsvorschläge

- Leichtgewichtige Ansätze:
 - Schichtenmodell
 - Namenskonventionen
 - Klare Projekt- / Ordnerstruktur

A wide-angle photograph of a beach at sunset. The sky is a gradient of orange and yellow, with the sun a bright white circle on the right. In the background, a line of palm trees separates the beach from a hill where a small white building with a red roof is visible. On the beach, several people are walking or standing. A yellow flag with a red emblem is on a pole. A small boat is in the water on the left. A dark blue horizontal band is overlaid across the middle of the image, containing the text 'Best Practices' in white. Another dark blue band is at the very bottom of the image.

Best Practices

Best Practices

- Gezielt einsetzen
 - Erst verstehen, dann nutzen
 - Kommunizieren, alle im Team mitnehmen
-
- Keep it simple, stupid



Zusammenfassung

Software Architektur ist einfach, oder?

- Sie sollte es sein
- Architektur = Werkzeug, kein Selbstzweck
- Weniger Mystik, mehr Pragmatismus
- Prinzipien helfen, wenn man sie pragmatisch nutzt
- Auch kleine Projekte profitieren

Fragen? Jetzt oder später!

Kontakt

 **E-Mail**
tkansy@dotnetconsulting.eu

 **Telefon**
[+49 \(0\) 6187 / 2009090](tel:+49(0)61872009090)

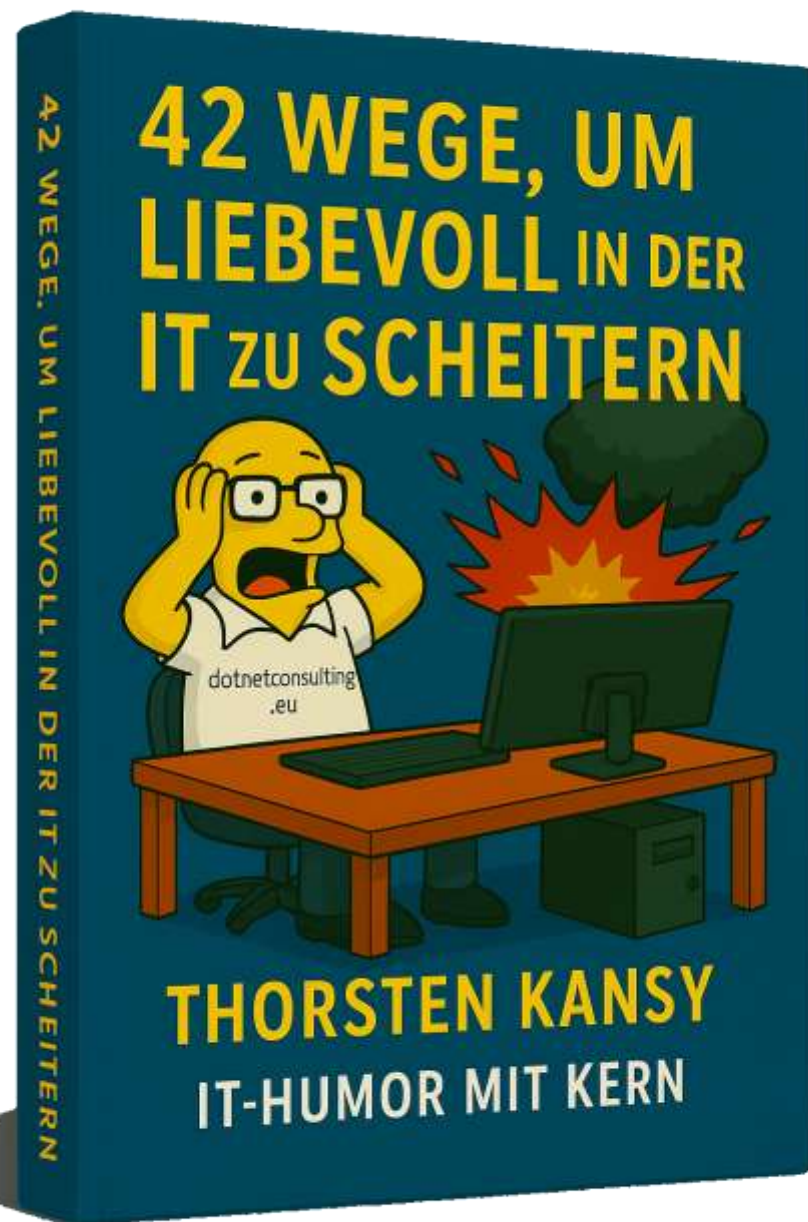
 **Microsoft Teams**
[Meet now](#)

 **LinkedIn**
[Link me](#)

 **XING**
[Xing me](#)

 **X (Twitter)**
[@tkansy](#)





IT-Humor mit Kern

IT ist lustig - bis du drinsteckst.



Jetzt lesen und mitlachen

www.dotnetconsulting.eu

SQL Server meets .NET (Core)- professionally!



Ich berate, coache und trainiere im Bereich Entwicklung von .NET (Core) Anwendungen mit Microsoft SQL Server- mit Allem, was dazu gehört- und was man vielleicht weglassen sollte.

