iSAQB® Glossar für Begriffe rund um Softwarearchitektur

2025.1-rev2-DE-20250117

Inhaltsverzeichnis

Einführung . . . 1
Persönliche Kommentare . . . 1
Begriffe können referenziert werden . . . 1
License . . . 1
Danksagung . . . 2
Begriffe . . . 3
A . . . 3
B . . . 9
C . . . 11
D . . . 13
E . . . 14
F . . . 15
G . . . 17
H . . . 17
I . . . 17
K . . . 19
L . . . 22
M . . . 23
N . . . 25
O . . . 25
P . . . 26
Q . . . 29
R . . . 37
S . . . 39
T . . . 47
U . . . 48
V . . . 49
W . . . 50
Z . . . 50
Übersetzungstabellen . . . 52
Referenzen und Quellen . . . 63
Anhänge . . . 66

Einführung

Hier finden Sie ein Glossar für *Begriffe rund um Softwarearchitektur*.

Es ist als Hilfsmittel für die Vorbereitung auf die Prüfung zum *Certified Professional for Software Architecture - Foundation Level®* des iSAQB® e. V. konzipiert.

Bitte beachten Sie: Dieses Glossar ist **nicht** als Einführung oder Lehrbuch zu Softwarearchitektur gedacht, sondern lediglich als eine Sammlung von Definitionen sowie von Links zu weiterführenden Informationen.

Außerdem finden Sie Vorschläge für Übersetzungen der iSAQB® Fachausdrücke, aktuell für Deutsch nach Englisch und andersherum.

Zu guter Letzt beinhaltet dieses Buch zahlreiche Verweise auf Bücher sowie auf andere Quellen, aus welchen wir in etlichen Definitionen zitieren.

Dieses Buch ist in Arbeit und unfertig.

Sie können Fehler oder Auslassungen in unserem Issue-Tracker auf GitHub melden, wo wir die Quellen für dieses Buch verwalten.

Persönliche Kommentare

Einige der Begriffe in diesem Buch wurden von einem:r oder mehreren Autor:innen kommentiert:

Kommentar (Gernot Starke)

Einige Begriffe mögen besonders wichtig sein, manchmal sind ein paar dezente Aspekte betroffen. Kommentare wie dieser hier spiegeln eine persönliche Meinung wider und sind **nicht** notwendigerweise die Meinung des iSAQB®.

Begriffe können referenziert werden

Alle Begriffe im Glossar haben eindeutige URLs auf die kostenlose Onlineversion des Buches. Daher können Sie durchgängig referenziert werden, in Onlinemedien wie auch in Printmedien.

Das Schema der URL ist sehr einfach:

- Die Basis-URL ist `https://public.isaqb.org/glossary/glossary-de.html`
- Wir fügen anschließend das Prefix `#term-` vor dem jeweiligen Begriff ein, der referenziert werden soll, dann den Begriff selbst, mit Bindestrichen anstelle von Leerzeichen. Allerdings ist es hierfür notwendig, den englischen Begriff zu kennen, da dieser als Anker verwendet wird.

Beispiel: Unsere Beschreibung des Begriffs "Softwarearchitektur" kann wie folgt mit einem Link referenziert werden: `https://public.isaqb.org/glossary/glossary-de.html#term-software-architecture`

Fast alle Begriffe sind über ihren vollen Namen verlinkt, nur ein paar Ausnahmen werden über ihre (gängigen) Abkürzungen verlinkt, wie etwa UML oder DDD.

License

Danksagung

Begriffe

A

Abhängigkeit

Siehe Kopplung.

Abhängigkeits-Umkehr-Prinzip / Dependency Inversion Principle

(Abstrakte) Elemente höherer Ebenen sollten nicht von (spezifischen) Elementen niedrigerer Ebenen abhängen. Details sollten von Abstraktionen abhängen ([Martin 2003]). Eines der SOLID-Prinzipien, das Brett Schuchert anschaulich erläutert, und das eng mit dem SDP und SAP zusammenhängt.

Abhängigkeitsinjektion / Dependency Injection (DI)

Objekte erzeugen abhängige Objekte nicht selbst, stattdessen werden die benötigten Abhängigkeiten an den Konstruktor übergeben oder via Setter gesetzt. Damit wird die Erzeugung von spezifischen Abhängigkeiten zum *Problem anderer Leute*. Wird oft benutzt um das Abhängigkeits-Umkehr-Prinzip sicherzustellen.

Ablage

In der Architekturdokumentation: Ort, an dem Artefakte gespeichert werden, ehe sie durch einen automatischen Prozess zu einem konsistenten Dokument zusammengestellt werden.

Im Domain-driven Design: Die Ablage ist ein Baustein des Domain-Driven Designs. Eine Ablage verbirgt technische Details der Infrastrukturschicht vor der Domänenschicht. Ablagen geben Entitäten zurück, die in der Datenbank bestehen.

Abstraktheit

Kennzahl für den Quellcode von objektorientierten Systemen: Zahl der abstrakten Typen (Schnittstellen und abstrakte Klassen), geteilt durch die Gesamtzahl der Typen.

Abstraktion

Betrachtung eines Elements, die sich auf die für einen bestimmten Zweck maßgeblichen Informationen konzentriert und die übrigen Informationen ignoriert.

Abwägung

(Syn.: Kompromiss). Erreichte oder ausgehandelte Balance zwischen zwei gewünschten oder vorgegebenen, aber üblicherweise unvereinbaren oder widersprüchlichen Eigenschaften. Beispielsweise muss in der Softwareentwicklung in der Regel ein Kompromiss zwischen Speicherbedarf und Laufzeitgeschwindigkeit gefunden werden.

Umgangssprachlicher gesagt, wenn etwas zunimmt, muss etwas anderes abnehmen.

Und noch umgangssprachlicher: Es gibt nichts umsonst. Für jedes Qualitätsmerkmal ist bei anderen Qualitätsmerkmalen ein Preis zu zahlen.

ACL

Zugriffskontrolllisten (Access Control Lists, ACL) repräsentieren die Autorisierung eines Principals, um auf eine spezifische Entität zuzugreifen. Eine ACL, die einer Entität zugeordnet ist, ist eine Liste von Principals

zusammen mit ihren Berechtigungen. Viele Dateisysteme - darunter Windows und POSIX - unterstützen die Verwendung von ACLs zur Zugriffskontrolle.

Da ACL sich nicht gut im großen Maßstab skalieren lassen, ist eine rollenbasierte Modellierung der Zugriffskontrolle (RBAC) gängig.

Adapter

Ein Adapter ist ein Entwurfsmuster, das die Nutzung einer vorhandenen Schnittstelle von einer anderen Schnittstelle aus ermöglicht. Er wird häufig dazu verwendet, vorhandene Komponenten ohne Veränderung ihres Quellcodes dazu zu bringen, mit anderen Komponenten zusammenzuarbeiten.

Aggregat

Ein Aggregat ist ein Baustein des Domain-Driven Designs. Aggregate sind komplexe Objektstrukturen, die aus Entitäten und Wertobjekten bestehen. Jedes Aggregat hat eine Root-Entität und wird in Bezug auf Änderungen als Einheit betrachtet. Aggregate stellen die Konsistenz und Integrität ihrer enthaltenen Entitäten mit Invarianten sicher.

Aggregation

Eine Form der Komposition in der objektorientierten Programmierung. Sie unterscheidet sich von der Komposition dadurch, dass sie keinen Besitz impliziert. Wenn das Element vernichtet wird, bleiben die enthaltenen Elemente intakt.

Akkreditierter Schulungsanbieter

Schulungsanbieter mit gültiger Akkreditierung des iSAQB®.

Akkreditierung

Prüfungsverfahren und Zertifizierung durch eine ermächtigte Akkreditierungsstelle (in diesem Fall das iSAQB®) zur Bestätigung, dass der Antragsteller die organisatorischen, technischen und qualitativen Anforderungen für Schulungsanbieter erfüllt.

Akkreditierungsstelle

Der Antrag auf Akkreditierung ist über die vom iSAQB benannte *Akkreditierungsstelle* einzureichen. Die Akkreditierungsstelle ist Ansprechpartner für alle Fragen des Schulungsanbieters während der Akkreditierung. Sie koordiniert das Akkreditierungsverfahren, führt die offizielle Bewertung der eingereichten Unterlagen durch und organisiert die technische Beurteilung in der AUDIT-ARBEITSGRUPPE.

Allgegenwärtige Sprache

Ein Konzept von Domain-driven Design: Eine allgegenwärtige Sprache ist eine Sprache, die um das Domänenmodell strukturiert ist. Sie wird von allen Teammitgliedern zur Verbindung aller Aktivitäten des Teams mit der Software genutzt. Die allgegenwärtige Sprache ist lebendig, entwickelt sich während eines Projekts weiter und verändert sich während des gesamten Lebenszyklus der Software.

Angemessenheit

Eignung für einen bestimmten Zweck.

Angriffsbaum

Formale Möglichkeit zur Beschreibung verschiedener Ansätze eines Angreifers zur Erreichung bestimmter

Ziele. Üblicherweise ist der Baum so aufgebaut, dass sich das Angriffsziel oben befindet und die verschiedenen Ansätze als Kindknoten dargestellt sind. Wahrscheinlich hat jeder Ansatz Abhängigkeiten, die wiederum als Kindknoten aufgeführt sind. Die Möglichkeit einer bestimmten Angriffsweise auf ein IT-System kann durch Zuweisung von zusätzlichen Attributen zu jedem Knoten analysiert werden. Mögliche Beispiele sind die geschätzten Kosten eines Angriffs oder die Frage der Möglichkeit eines Angriffsansatzes mittels Einbeziehung von Gegenmaßnahmen.

Siehe Bruce Schneier zu "Modeling security threats".

arc42

Kostenfreies Open-Source Template zur Kommunikation und Dokumentation von Softwarearchitekturen. arc42 besteht aus 12 (optionalen) Teilen oder Abschnitten.

Architecture Objective

TODO

Architektur

Siehe Softwarearchitektur.

Architektur-Framework

Konventionen, Grundsätze und Praktiken für die Beschreibung von Architekturen, die in einem spezifischen Anwendungsbereich und/oder einer Gemeinschaft von Stakeholdern festgelegt wurden (gemäß Definition in ISO/IEC/IEEE 42010).

Beispiele:

- Generalised Enterprise Reference Architecture and Methodologies (GERAM) [ISO 15704] ist ein Architektur-Framework.
- Reference Model of Open Distributed Processing (RM-ODP) [ISO/IEC 10746] ist ein Architektur-Framework.

Architektur-Qualitätsanforderung

Siehe Architekturziel.

Architekturbegründung

Die Architekturbegründung enthält Erläuterungen, Rechtfertigungen oder Argumentationen zu getroffenen Architekturentscheidungen. Die Begründung einer Entscheidung kann die Entscheidungsgrundlage, berücksichtige Alternativen und Kompromisse, mögliche Folgen der Entscheidung und Quellenangaben für zusätzliche Informationen enthalten (gemäß Definition in ISO/IEC/IEEE 42010).

Architekturbeschreibung

Arbeitsergebnis, das genutzt wird, um eine Architektur zum Ausdruck zu bringen (gemäß Definition in ISO/IEC/IEEE 42010).

Architekturbeschreibungselement

Ein Architekturbeschreibungselement ist ein beliebiges Konstrukt in einer Architekturbeschreibung. Architekturbeschreibungselemente sind die grundlegendsten Konstrukte, die in ISO/IEC/IEEE 42010 behandelt werden. Bei allen in ISO/IEC/IEEE 42010 definierten Begriffen handelt es sich um eine

Spezialisierung des Konzepts eines Architekturbeschreibungselements (gemäß Definition in ISO/IEC/IEEE 42010).

Architekturbeschreibungssprache

Architekturbeschreibungssprachen (ADL) sind jegliche Ausdrucksformen zur Verwendung in Architekturbeschreibungen (gemäß Definition in ISO/IEC/IEEE 42010).

Beispiele sind Rapide, Wright, SysML, ArchiMate und die Sprachen der verschiedenen Blickwinkel in RM-ODP [ISO 10746].

Architekturbewertung

Quantitative oder qualitative Beurteilung einer (Software- oder System-) Architektur. Erlaubt es, festzustellen, ob eine Architektur ihre Zieleigenschaften oder Qualitätsmerkmale erreichen kann.

Siehe Beurteilung

Anmerkung (Gernot Starke)

Ich halte die Begriffe *Architekturanalyse* oder *Architekturbeurteilung* für passender, da in *Bewertung Wert* mitschwingt und eine numerische Beurteilung oder Kennzahlen impliziert werden, was üblicherweise nur ein *Teil* dessen ist, was im Rahmen einer Architekturanalyse gemacht werden sollte.

Architekturblickwinkel

Arbeitsergebnis zur Festlegung der Konventionen für den Aufbau, die Interpretation und die Nutzung von Architektursichten für spezifische Systembelange (gemäß Definition in ISO/IEC/IEEE 42010).

Architekturentscheidung

Entscheidung mit nachhaltiger oder wesentlicher Auswirkung auf die Architektur.

Beispiel: Entscheidung über Datenbanktechnologie oder technische Grundlagen der Benutzeroberfläche.

Gemäß ISO/IEC/IEEE 42010 bezieht sich eine Architekturentscheidung auf Systembelange. Jedoch gibt es häufig kein einfaches Mapping zwischen den beiden. Eine Entscheidung kann sich auf verschiedene Weise auf die Architektur auswirken. Dies kann in der Architekturbeschreibung dargestellt werden (gemäß Definition in ISO/IEC/IEEE 42010).

Architekturmodell

Eine Architektursicht besteht aus einem oder mehreren Architekturmodellen. Ein Architekturmodell verwendet für die betreffenden Belange geeignete Modellierungskonventionen. Diese Konventionen sind in der Modellart für dieses Modell festgelegt. In einer Architekturbeschreibung kann ein Architekturmodell Teil von mehr als einer Architektursicht sein (gemäß Definition in ISO/IEC/IEEE 42010).

Architekturmuster

Ein Architekturmuster beschreibt eines grundlegendes strukturelles Organisationsschema für Softwaresysteme. Es liefert eine Reihe von vordefinierten Teilsystemen, spezifiziert ihre Verantwortlichkeiten und enthält Richtlinien für die Organisation der Beziehungen zwischen ihnen ([Buschmann+1996], Seite 12). Vergleichbar mit Architekturstil.

Beispiele:

- Layers
- Pipes und Filter
- Microservices
- CQRS

Architektursicht

Eine Darstellung eines Systems aus einer spezifischen Perspektive. Wichtige und bekannte Sichten:

- Kontextabgrenzung
- Bausteinsicht
- Laufzeitsicht
- Verteilungssicht

[Bass+2021] und [Rozanski+2011] erörtern dieses Konzept ausführlich.

Laut ISO/IEC/IEEE 42010 ist eine Architektursicht ein Arbeitsergebnis, das die Architektur eines Systems aus der Perspektive spezifischer Systembelange darstellt (gemäß Definition in ISO/IEC/IEEE 42010).

Architekturstil

Beschreibung von Element- und Beziehungstypen zusammen mit Einschränkungen ihrer Nutzungsweise. Häufig Architekturmuster genannt. Beispiele: Pipes und Filter, Model-View-Controller, Schichten.

Architekturtaktik

TODO

Architekturziel

(Syn.: Architektur-Qualitätsziel, Architektur-Qualitätsanforderung): Ein Qualitätsmerkmal, das ein System erreichen muss und bei dem es sich um eine Architekturfrage handelt.

Daher ist die Architektur so zu entwerfen, dass das Architekturziel erfüllt wird. Diese Ziele sind im Gegensatz zu (kurzfristigen) Projektzielen häufig *langfristig*.

Artefakt

Greifbares Nebenprodukt, das während der Softwareentwicklung erstellt oder erzeugt wird. Beispiele für Artefakte sind Anwendungsfälle, alle Arten von Diagrammen, UML-Modelle, Anforderungs- und Entwurfsunterlagen, Quellcode, Testfälle, Klassendateien, Archive.

Asset

"In der Informationssicherheit, Computersicherheit und Netzwerksicherheit sind Assets jegliche Daten, Geräte oder sonstigen Komponenten der Umgebung, die Aktivitäten in Zusammenhang mit Informationen unterstützen. Assets umfassen im Allgemeinen Hardware (z.B. Server und Switches), Software (z.B. missionskritische Anwendungen und Supportsysteme) und vertrauliche Informationen."

(Übersetztes englisches Zitat aus Wikipedia)

Assoziation

Definiert eine Beziehung zwischen Objekten (im Allgemeinen zwischen Modulen). Jede Assoziation lässt sich durch Kardinalitäten und (Rollen-)Namen im Detail beschreiben.

Siehe Kopplung, Abhängigkeit und Beziehung

Asymmetrische Kryptografie

Algorithmen der asymmetrischen Kryptografie sind so ausgelegt, dass zur Verschlüsselung und zur Entschlüsselung unterschiedliche Schlüssel verwendet werden. Der Schlüssel für die Verschlüsselung wird "öffentlicher Schlüssel" genannt und der Schlüssel für die Entschlüsselung "privater Schlüssel". Der öffentliche Schlüssel kann veröffentlicht werden und von jedem zur Verschlüsselung von Informationen verwendet werden; diese können nur von der Partei, die im Besitz des privaten Schlüssels für die Entschlüsselung ist, gelesen werden. Siehe [Schneier 1996].

Asymmetrische Kryptografie ist die Grundlage für PKI und digitale Signaturen.

ATAM

Architecture Tradeoff Analysis Method. Qualitative Architekturbewertungsmethode, basierend auf einem (hierarchischen) Qualitätsbaum und konkreten Qualitätsszenarien. Grundidee: Vergleich feinkörniger Qualitätsszenarien ("Qualitätsanforderungen") mit den entsprechenden Architekturansätzen zur Identifizierung von Risiken und Kompromissen.

Audit-Arbeitsgruppe:

Die *Audit-Arbeitsgruppe* ist für die technische Beurteilung der Schulungsunterlagen sowie für die Überwachung und Beurteilung der Schulungen zuständig. Die vom iSAQB® ermächtigten Mitglieder der Audit-Arbeitsgruppe sind von den Schulungsanbietern unabhängig. Der Schulungsanbieter wird von der Akkreditierungsstelle über das Ergebnis der Beurteilungen (die jeweilige Akkreditierungsempfehlung der Audit-Arbeitsgruppe) informiert.

Authentifizierung

Authentifizierung ist der Vorgang der Bestätigung der Identitätsbehauptung einer gegebenen Entität. Dies geschieht üblicherweise durch Überprüfung von mindestens einem der dem System bekannten Authentifizierungsfaktoren:

- Wissen (z.B. Passwort)
- Besitz (z.B. Sicherheitstoken)
- Inhärenz (z.B. Biometrie)

Für eine stärkere Authentifizierung können mehrere Faktoren oder mindestens Faktoren aus zwei Kategorien verlangt werden.

Autorisierung

"Autorisierung ist der Vorgang der Spezifizierung von Zugriffsrechten für Ressourcen in Zusammenhang mit Informationssicherheit und Computersicherheit im Allgemeinen und mit Zugriffskontrolle im Besonderen. Förmlicher bezeichnet "autorisieren" die Festlegung einer Zugrissrichtlinie."

(Übersetztes englisches Zitat aus Wikipedia)

Azyklischer Abhängigkeitsgrundsatz (ADP)

Ein Grundsatz für die Gestaltung der Struktur von Softwaresystemen (siehe auch Packaging-Prinzipien). Er besagt, dass der Abhängigkeitsgraph eines Systems keine Zyklen enthalten darf, was auch eine Notwendigkeit für die hierarchische Zerlegung ist.

Die Vermeidung von Abhängigkeitszyklen ist für lose Kopplung und Wartbarkeit entscheidend, da *alle* Komponenten in einem Abhängigkeitszyklus effektiv, auch wenn mittelbar, voneinander abhängen, wodurch es schwierig ist, einen Teil des Zyklus isoliert zu verstehen, zu ändern oder zu ersetzen (siehe auch [Lilienthal 2019]).

Auch wenn Robert C. Martin ([Martin 2003]) sich auf große Komponenten objektorientierter Software bezog, ist ADP ein *universeller* Grundsatz. Er geht (mindestens) auf einen der Ursprünge der Softwarearchitektur zurück, den Klassiker von 1972 "On the Criteria To Be Used in Decomposing Systems into Modules" ([Parnas 1972]), der zu dem Ergebnis gelangt, dass eine hierarchische Struktur zusammen mit einer "sauberen" Zerlegung wünschenswerte Eigenschaften eines jeden Systems sind.

Es kann argumentiert werden, dass ein Abhängigkeitszyklus, selbst vor Berücksichtigung seiner verschiedenen praktischen Probleme, logisch bereits so fehlerhaft ist wie ein Zirkelargument oder eine Zirkeldefinition. Als struktureller Widerspruch kann ein Zyklus weder ein *angemessenes* noch ein aussagekräftiges Modell der inhärenten Natur und des Zwecks eines Systems sein. Alleine diese konzeptuelle Abweichung führt geradezu mit Sicherheit zur Entstehung von Problemen. Und genau das soll durch einen *Prinzip*-Ansatz verhindert werden.

B

Baustein

Allgemeiner oder abstrakter Begriff für alle Arten von Artefakten, aus denen Software aufgebaut ist. Teil der statischen Struktur (Bausteinsicht) von Softwarearchitektur.

Bausteine können hierarchisch strukturiert sein, sie können andere (kleinere) Bausteine enthalten.

Einige Beispiele für alternative (konkrete) Bezeichnungen von Bausteinen:
Komponente, Modul, Paket, Namensraum, Klasse, Datei, Programm, Teilsystem, Funktion, Konfiguration, Datendefinition.

Bausteinsicht

Zeigt die statische Struktur des Systems, die Organisationsweise des Quellcodes. Üblicherweise hierarchisch, ausgehend von der [Kontextabgrenzung]. Ergänzt durch ein oder mehrere [Laufzeitsichten].

Begründung

Erläuterung der Argumentation oder Argumente, die einer Architekturentscheidung zugrunde liegen.

Belang

Belange in Bezug auf eine Architektur sind Anforderungen, Ziele, Einschränkungen, Absichten oder Bestrebungen eines Stakeholders für diese Architektur. ([Rozanski+2011], Kapitel 8)

Gemäß ISO/IEC/IEEE 42010 ist Belang definiert als (System-)Interesse an einem System, das für einen oder mehrere Stakeholder relevant ist (gemäß Definition in ISO/IEC/IEEE 42010).

Belange beziehen sich auf jegliche Einflüsse auf ein System in seiner Umgebung, wie Entwicklungs-,

Geschäfts- und Betriebseinflüsse sowie technologische, organisatorische, politische, wirtschaftliche, rechtlichen, regulatorische, ökologische und soziale Einflüsse.

Beobachter / Observer

(Entwurfsmuster) "… in dem ein Objekt eine Liste seiner abhängigen Strukturen, Observer genannt, führt und sie automatisch, in der Regel durch Aufruf einer ihrer Methoden, über Zustandsänderungen benachrichtigt." (Übersetztes englisches Zitat aus Wikipedia)

Das Beobachtermuster ist ein wesentlicher Bestandteil des MVC-Architekturmusters.

Betrachtetes System

Betrachtetes System (oder einfach System) bezieht sich auf das System, dessen Architektur bei der Erstellung der Architekturbeschreibung betrachtet wird (gemäß Definition in ISO/IEC/IEEE 42010).

Beurteilung

Siehe auch Bewertung.

Zusammenstellung von Informationen über Status, Risiken oder Schwächen eines Systems. Die Beurteilung kann potenziell alle Aspekte (Entwicklung, Organisation, Architektur, Code usw.) betreffen.

Beziehung

Allgemeiner Begriff zur Bezeichnung einer Art von Abhängigkeit zwischen Elementen einer Architektur. In Architekturen werden unterschiedliche Arten von Beziehungen verwendet, z.B. Aufruf, Benachrichtigung, Besitz, Containment, Erzeugung oder Vererbung.

Blackboard

Architekturmuster, welches oft für Probleme benutzt wird, die keine deterministische Lösungsstrategien erlauben. Mehrere spezialisierte Teilsysteme fügen im Blackboard ihr Wissen zusammen, um eine möglicherweise partielle oder ungefähre Lösung zu erarbeiten. (Zitiert aus [Buschmann+1996])

Blackbox

Sicht auf einen Baustein (oder eine Komponente), die die interne Struktur verbirgt. Blackboxen achten das *Geheimnisprinzip*. Sie müssen klar definierte Ein- und Ausgabeschnittstellen sowie eine präzise formulierte *Verantwortlichkeit* oder ein präzise formuliertes *Ziel* haben. Optional definiert eine Blackbox einige Qualitätsmerkmale, wie beispielsweise zeitliches Verhalten, Durchsatz oder Sicherheitsaspekte.

Bottom-up-Ansatz

Arbeitsrichtung (oder Bearbeitungsstrategie) für Modellierung und Entwurf. Ausgehend von detaillierten oder konkreten Aspekten wird auf etwas Allgemeineres oder Abstrakteres hingearbeitet.

"Beim Bottom-up-Ansatz werden zunächst die einzelnen Grundelemente des Systems mit hohem Detailgrad spezifiziert. Diese Elemente werden dann miteinander zu größeren Teilsystemen verknüpft." (Übersetztes englisches Zitat aus Wikipedia)

Broker

Ein Architekturmuster zur Strukturierung von verteilten Softwaresystemen mit entkoppelten Komponenten, die über (üblicherweise Remote-) Serviceaufrufe interagieren.

Broker sind für die Koordinierung der Kommunikation, wie die Weiterleitung von Anfragen, sowie die Übermittlung von Ergebnissen und Ausnahmen zuständig.

Brücke

Entwurfsmuster, bei dem eine Abstraktion von ihrer Implementierung entkoppelt ist, sodass beide unabhängig voneinander variieren können. Wenn Ihnen das (wie den meisten Menschen) unverständlich erscheint – sehen Sie hier nach.

C

C4 Model

Das C4 Model for Software Architecture Documentation wurde von Simon Brown entwickelt. Es besteht aus einer hierarchischen verknüpften Menge an Softwarearchitekturdiagrammen für Kontext, Container, Komponenten und Code.

Die Hierarchie der C4 Diagramme stellt verschiedene Abstraktionslevel bereit, wobei jedes ist für andere Leser relevant ist.

CA

Ein Zertifizierungsstelle (Certificate Authority, CA) stellt digitale Zertifikate für ein gegebenes Subjekt in einer PKI aus. Üblicherweise besteht Vertrauen in diese Stelle, das zum gleichen Maß an Vertrauen in die ausgestellten Zertifikate führt.

Ein Beispiel ist die weitverbreitete TLS-PKI, bei der jeder Browser die Wurzelzertifikate einer festgelegten Liste von CA enthält. Diese CA überprüfen dann die Identität eines gegebenen Internet-Domaininhabers und signieren sein Zertifikat digital für die Verwendung mit TLS.

CIA-Triade

Siehe Schutzziele

Clean Architektur

See Ports und Adapter.

Closure of Operation

Siehe Kombinator.

Cloud

"Cloudcomputing ist ein Modell zur Ermöglichung eines allgegenwärtigen, bequemen, auf Abruf verfügbaren Netzzugriffs auf einen gemeinsamen Pool konfigurierbarer Rechenressourcen (z.B. Netzwerke, Server, Speicher, Anwendungen und Dienste), der schnell bereitgestellt und mit geringfügigem Verwaltungsaufwand bzw. minimalen Eingriffen durch den Dienstanbieter freigegeben werden kann."

Übersetztes englisches Zitat von NIST (National Institute of Standards and Technology).

Die NIST-Definition enthält die folgenden fünf Eigenschaften (die ebenfalls von der oben genannten NIST-Quelle stammen, jedoch verkürzt wurden):

- On-Demand Self-Service: Ein Kunde kann einseitig Rechenkapazitäten, wie Serverzeit und Netzwerkspeicher, anfordern, ohne dass eine menschliche Interaktion mit jedem Dienstanbieter

erforderlich ist.

- Broad Network Access: Die Leistungen sind mittels Standardmechanismen, die durch heterogene Client-Plattformen die Nutzung fördern, über das Netzwerk zugänglich.
- Resource Pooling: Die Rechenressourcen des Anbieters werden gebündelt, sodass mehrere Kunden mit einem mandantenfähigen Modell bedient werden können, wobei die verschiedenen physischen und virtuellen Ressourcen gemäß Kundenanforderung dynamisch zugewiesen und neu zugewiesen werden. Dies geschieht ortsunabhängig, wobei der Kunde in der Regel keine Kontrolle oder Kenntnis über den genauen Standort der bereitgestellten Ressourcen hat, jedoch gegebenenfalls den Standort auf einer höheren Abstraktionsebene spezifizieren kann (z.B. Land, Bundesstaat oder Rechenzentrum).
  Beispiele für Ressourcen sind Speicher, Verarbeitung, Arbeitsspeicher und Netzwerkbandbreite.
- Rapid Elasticity: Die Dienste können flexibel und in manchen Fällen automatisch bereitgestellt und freigegeben werden, um sich schnell an den Bedarf anzupassen. Für den Kunden erscheinen die verfügbaren Kapazitäten oft unbegrenzt und jederzeit in beliebiger Menge verfügbar.
- Measured Service: Cloud-Systeme können durch den Einsatz von Messverfahren auf einer der Art des Dienstes (z.B. Speicher, Verarbeitung, Bandbreite und aktive Benutzerkonten) angemessenen Abstraktionsebene die Ressourcennutzung automatisch steuern und optimieren. Die Ressourcennutzung kann überwacht, gesteuert und berichtet werden, was Transparenz für den Anbieter sowie den Kunden des betreffenden Dienstes schafft.

Common-Closure-Prinzip

Ein Grundsatz für die Gestaltung der Struktur von Softwaresystemen (siehe auch Packaging-Prinzipien). Er ist eine direkte und ausdrückliche Neuformulierung des Single-Responsibility-Prinzips für größere Komponenten.

Die Unterkomponenten einer Komponente sollen idealerweise genau dieselben Änderungsgründe haben. Ein Änderungsantrag, der sich auf eine von ihnen auswirkt, soll sich auf sie alle, aber auf **nichts** außerhalb ihrer enthaltenden Komponente auswirken.

Dadurch würde sich jeder erwartete Änderungsantrag auf eine minimale Zahl an Komponenten auswirken. Oder anders gesagt: Jede Komponente wäre gegenüber einer maximalen Zahl an erwarteten Änderungsanträgen geschlossen. Der Begriff **erwartet** hat an dieser Stelle einige bedeutende Auswirkungen:

1. Die inhärenten Konzepte/Verantwortlichkeiten eines Systems gehen tiefer als einer Beschreibung seines Verhaltens auf Oberflächenebene.
2. Die tieferen Konzepte/Verantwortlichkeiten eines Systems sind nicht vollständig objektiv, sondern können auf unterschiedliche Weise modelliert werden.
3. Die Festlegung der Konzepte/Verantwortlichkeiten eines Systems ist nicht nur eine passive Beschreibung, sondern aktive **Strategieentwicklung**.

Dieses Prinzip führt zu Komponenten mit starker Kohäsion. Es geht auch mit lose gekoppelten Komponenten einher, da verbundene Konzepte, die sich zusammen **ändern**, in dieselbe Komponente **gebündelt** werden. Wenn jedes einzelne Konzept von einer einzigen Komponente ausgedrückt wird, gibt es keine unnötigen Kopplungen zwischen Komponenten.

Common-Reuse-Prinzip

Ein Grundsatz für die Gestaltung der Struktur von Softwaresystemen (siehe auch Packaging-Prinzipien).

Die Unterkomponenten (Klassen) einer Komponente sollen genau die sein, die zusammen (wieder)verwendet werden. Oder andersherum: Komponenten, die zusammen (wieder)verwendet werden, sollen in eine größere Komponente gepackt werden. Dies bedeutet auch, dass Unterkomponenten, die **nicht** häufig zusammen mit anderen Unterkomponenten verwendet werden, **nicht** in der entsprechenden Komponente sein sollen.

Diese Perspektive hilft bei der Entscheidung, was in eine Komponente gehört und was nicht. Sie zielt auf eine Systemzerlegung mit lose gekoppelten und stark kohärenten Komponenten ab.

Dies steht natürlich im engen Zusammenhang mit dem Single-Responsibility-Prinzip. Außerdem besteht ein Zusammenhang zum Schnittstellenaufteilungsprinzip, da das Prinzip sicherstellt, dass Clients nicht gezwungen werden, von Konzepten abzuhängen, die für sie bedeutungslos sind.

CPSA®

Certified Professional for Software Architecture® — die gängige Bezeichnung für die verschiedenen Zertifizierungsstufen des iSAQB. Die bekanntesten Zertifizierungen sind das Foundation Level (CPSA-F) und das Advanced Level (CPSA-A).

CQRS

(Command-Query-Responsibility-Segregation): Trennt die Elemente, die Daten manipulieren (**Befehl**) von denen, die Daten nur lesen (**Abfrage**). Diese Trennung ermöglicht verschiedene Optimierungsstrategien für das Lesen und Schreiben von Daten (beispielsweise ist es wesentlich leichter, schreibgeschützte Daten zu cachen, als Daten, die auch abgeändert werden können).

Es gibt ein interessantes eBook von Mark Nijhof zu diesem Thema.

D

Dienst

Ein Dienst oder Service beschreibt eine technische, autarke Einheit, die zusammenhängende Funktionalitäten, typischerweise zu einem Themenkomplex, bündelt und über eine klar definierte Schnittstelle zur Nutzung durch andere Bausteine zur Verfügung stellt.

Idealerweise abstrahiert ein Dienst die interne, technische Funktion soweit, dass es für die Nutzung des Dienstes nicht notwendig ist, interne Implementierungsdetails zu kennen oder gar zu verstehen.

Typische Beispiele für extern erreichbare Dienste sind Webservices, Netzwerkdienste, Systemdienste oder auch Telekommunikationsdienste.

(basierend auf Wikipedia)

Dokument

Ein (üblicherweise schriftliches) Artefakt zur Informationsvermittlung.

Dokumentation

Systematisch geordnete Sammlung von Dokumenten und sonstigen Materialien aller Art, die die Nutzung oder Beurteilung erleichtern. Beispiele für "sonstige Materialien": Präsentationen, Videos, Audios, Webseiten, Bilder usw.

Dokumentationserstellung

Automatischer Prozess, mit dem Artefakte zu einer konsistenten Dokumentation zusammengestellt werden.

Domain-Driven Design (DDD)

"Domain-Driven Design (DDD) ist ein Ansatz zur Softwareentwicklung für komplexe Anforderungen durch tiefreichende Verbindung der Implementierung mit einem sich evolvierenden Modell der Kerngeschäftskonzepte." (Übersetztes englisches Zitat von DDDCommunity). Siehe [Evans 2004].

Siehe auch:

- Entität
- Wertobjekt
- Aggregat
- Service
- Fabrik
- Ablage
- Allgegenwärtige Sprache

Domänenmodell

Das Domänenmodell ist ein Konzept von Domain-Driven Design. Das Domänenmodell ist ein System aus Abstraktionen zur Beschreibung ausgewählter Aspekte einer Fachdomäne und kann zur Lösung von Problemen in Zusammenhang mit dieser Domäne verwendet werden.

E

Eingebettete Systeme

In ein größeres mechanisches oder elektrisches System *eingebettetes* System. Eingebettete Systeme haben häufig Echtzeit-Recheneinschränkungen. Typische Eigenschaften von eingebetteten Systemen sind niedriger Stromverbrauch, begrenzter Speicher und begrenzte Verarbeitungsressourcen sowie geringe Größe.

Einschränkung

Eine Einschränkung des Freiheitsgrads bei der Erstellung, dem Entwurf, der Implementierung oder der sonstigen Bereitstellung einer Lösung. Einschränkungen sind häufig *globale Anforderungen*, wie begrenzte Entwicklungsressourcen oder eine Entscheidung der Geschäftsleitung, die einschränkt, wie ein System geplant, entworfen, entwickelt oder betrieben wird.

Gestützt auf eine Definition von Scott Ambler

Entität

Eine Entität ist ein Baustein des Domain-Driven Designs. Eine Entität ist ein Kernobjekt einer Geschäftsdomäne mit einer unveränderlichen Identität und einem klar definierten Lebenszyklus. Entitäten mappen ihren Zustand auf Wertobjekte und sind fast immer persistent.

Entropie

In der Informationstheorie definiert als "Menge an Informationen" in einer Nachricht oder

"Unvorhersehbarkeit des Informationsgehalts". Die Entropie eines Kryptosystems wird anhand der Größe des Schlüsselraums gemessen. Größere Schlüsselräume haben eine höhere Entropie und sind, wenn sie nicht durch den Algorithmus selbst fehlerhaft sind, schwerer zu knacken als kleinere. Für sichere kryptografische Vorgänge sind nicht nur zufällige Werte als Input vorgeschrieben, sondern sie sollten auch eine hohe Entropie aufweisen. Die Schaffung von hoher Entropie in einem Computersystem ist nicht trivial und kann die Systemleistung beeinträchtigen.

Siehe [Schneier 1996] und Whitewood Inc. zu "Understanding and Managing Entropy" oder SANS "Randomness and Entropy - An Introduction".

Entwurfsbegründung

Eine explizite Dokumentation der Gründe für Entscheidungen, die beim Entwurf eines Architekturelements getroffen wurden.

Entwurfsmuster

Allgemeine oder generische wiederverwendbare Lösung für ein gängiges Problem in einem gegebenen Kontext beim Entwurf. Das ursprünglich von dem berühmten Architekten Christopher Alexander erdachte Konzept von *Entwurfsmustern* wurde von Softwareentwicklern übernommen.

Unserer Ansicht nach sollte jeder ernsthafte Softwareentwickler zumindest einige Muster aus dem bahnbrechenden Buch Gang-of-Four von Erich Gamma ([GoF: Design-Patterns]) und seinen drei Verbündeten kennen.

Entwurfsprinzip

Eine Reihe von Richtlinien, die Softwareentwicklern hilft, bessere Lösungen zu entwerfen und zu implementieren, wobei "besser" bedeutet, die folgenden **schlechten Eigenschaften** zu vermeiden:

- **Rigidität**: Ein System oder Element ist schwer zu ändern, weil jede Änderung sich möglicherweise auf zahlreiche andere Elemente auswirkt.
- **Fragilität**: Wenn Elemente geändert werden, treten unerwartete Ergebnisse, Fehler oder sonstige negative Folgen bei anderen Elementen auf.
- **Immobilität**: Ein Element ist schwer wiederzuverwenden, weil es sich nicht aus dem übrigen System herauslösen lässt.

Gleichzeitig sollten folgende *gute Eigenschaften* angestrebt werden:

- Lose Kopplung
- Hohe Kohäsion
- Separation of Concerns oder das Einhalten des Single-Responsibility-Prinzips.

Diese Eigenschaften wurden von Robert Martin formuliert und stammen von OODesign.com

Event Sourcing

Architekturmuster, bei dem Änderungen am Zustand der Anwendung als eine Serie von unveränderlichen Ereignissen erfasst werden. Anstatt nur den aktuellen Zustand der Anwendung zu speichern, wird jede Zustandsänderung als Ereignis in einem "append-only"-Log festgehalten.

F

Fabrik

(Entwurfsmuster) In der klassenbasierten oder objektorientierten Programmierung ist das Entwurfsmuster Fabrikmethode ein Erzeugungsmuster, das Fabrikmethoden oder Fabrikkomponenten zur Erzeugung von Objekten nutzt, ohne die exakte Klasse des zu erzeugenden Objekts spezifizieren zu müssen.

Im Domain-Driven Design: Eine Fabrik kapselt die Erzeugung von Aggregaten, Entitäten und Wertobjekten. Fabriken arbeiten ausschließlich in der Domäne und haben keinen Zugriff auf technische Bausteine (z.B. eine Datenbank).

Fachlicher Kontext

Zeigt das vollständige System als eine Blackbox innerhalb seiner Umgebung aus fachlicher Perspektive und enthält eine Spezifikation aller Kommunikationspartner (Benutzer, IT-Systeme, usw.) mit Erläuterungen zu den domänenspezifischen Ein- und Ausgaben oder Schnittstellen. Zu beachten ist, dass die spezifischen technischen Lösungen für die Interaktion mit externen Akteuren in der Regel nicht im fachlichen Kontext dargestellt werden sollten, da sie zum technischen Kontext) gehören.

Siehe Kontextabgrenzung.

Fassade

Strukturentwurfsmuster. Eine Fassade bietet eine einfache Schnittstelle zu einem komplexen oder komplizierten Baustein (dem *Provider*) ohne Modifikationen am Provider.

Filter

Teil des "Pipes und Filter"-Architekturpatterns, welches genutzt wird, um Daten zu erzeugen oder transformieren. Filter werden üblicherweise unabhängig von anderen Filtern ausgeführt. Siehe Pipes und Filter

Fitnessfunktion

"Eine architektonische Fitnessfunktion bietet eine objektive Integritätsbewertung einiger architektonischer Merkmale." ([Ford+2017]).

Eine Fitnessfunktion wird aus manuellen Bewertungen und automatisierten Tests abgeleitet und zeigt, inwieweit die Architektur- oder Qualitätsanforderungen erfüllt werden.

Fundamental Modeling Concepts (FMC)

Fundamental Modeling Concepts (FMC)

Grafische Notation für die Modellierung und Dokumentation von Softwaresystemen. Von ihrer Website: "FMC bietet einen Rahmen für die umfassende Beschreibung von softwareintensiven Systemen. Es basiert auf einer präzisen Terminologie und wird durch eine leicht verständliche grafische Notation unterstützt."

Funktionssignatur

(Synonym: Typ- oder Methodensignatur) definiert die Eingabe und Ausgabe von Funktionen oder Methoden.

Eine Signatur kann enthalten:

- Parameter und ihre Typen

- Rückgabewert und Typ
- geworfene Ausnahmen oder ausgelöste Fehler

G

Gateway

Ein (Entwurfs- oder Architektur-)Muster: Elemente, die den Zugriff auf (üblicherweise externe) Systeme oder Ressourcen kapseln. Siehe auch Wrapper, Adapter.

Geschäftsarchitektur

Ein Plan des Unternehmens, der eine gemeinsame Verständnisgrundlage der Organisation bildet und zur Abstimmung von strategischen Zielen und taktischen Anforderungen genutzt wird.

Globale Analyse

Systematischer Ansatz zur Erreichung der gewünschten Qualitätsmerkmale. Entwickelt und dokumentiert von Christine Hofmeister (Siemens Corporate Research). Die globale Analyse wird in [Hofmeister+2000] beschrieben.

H

Heterogener Architekturstil

Siehe hybrider Architekturstil.

Heuristik

Informelle Regel, Faustformel. Möglichkeit zur Problemlösung, die nicht mit Sicherheit optimal, aber in gewisser Weise ausreichend ist. Beispiele aus dem objektorientierten Entwurf oder Benutzeroberflächenentwurf.

Hexagonale Architektur

See Ports und Adapter.

Hybrider Architekturstil

Kombination aus zwei oder mehreren existierenden Architekturstilen oder -mustern. Beispielsweise ein in eine Schichtstruktur eingebettetes MVC-Konstrukt.

I

IEEE-1471

Norm *Recommended Practice for Architectural Description of Software-Intensive Systems*, definiert als ISO/IEC 42010:2007. Legt einen (abstrakten) Rahmen für die Beschreibung von Softwarearchitekturen fest.

Inkrementelle Entwicklung

Siehe iterative und inkrementelle Entwicklung.

Integrität

Verschiedene Bedeutungen:

Eines der grundlegenden Schutzziele, das die Aufrechterhaltung und Gewährleistung der Richtigkeit und Vollständigkeit der Daten bezeichnet. Dies wird üblicherweise durch den Einsatz von kryptografischen Algorithmen zur Erstellung einer digitalen Signatur erreicht.

Daten- oder Verhaltensintegrität:

- Maß, in dem Clients (einer Datenbank) bei identischen Abfragen identische Ergebnisse erhalten (z.B. Monotonic-Read-Consistency, Monotonic-Write-Consistency, Read-Your-Writes-Consistency etc.)
- Maß, in dem ein System sich kohärent, reproduzierbar und vernünftig verhält.

Siehe auch Konsistenz.

Interpreter

Entwurfsmuster, das Domänenobjekte oder domänenspezifische Sprachen als Syntax darstellt. Eine Interpreterfunktion liefert dann eine semantische Interpretation der Domänenobjekte getrennt von den Objekten.

iSAQB®

international Software Architecture Qualification Board – eine international aktive Organisation zur Förderung der Entwicklung der Softwarearchitektur-Ausbildung. Siehe auch die Diskussion im Anhang.

ISO 25010

Systems and Software Engineering - Systems and Software Quality Requirements and Evaluation (SQuaRE)

Es handelt sich um eine Norm zur Beschreibung der *Softwareproduktqualität*.

ISO schlägt ein hierarchisches Modell der Produktqualität vor, mit derzeit (Standardversion 2011) 8 Top-Level-Attributen.

image::1_architecture/ISO-25010-EN.png

Die zukünftige Version des ISO-Standards (die Stand 2022 noch im Draft-Status ist) enthält 9 dieser Attribute.

Eine Liste der in der ISO 25010 Norm definierten Qualitätsmerkmale findet sich unter [ISO-25010].

ISO 9126

(Veraltete) Norm zu Beschreibung (und Bewertung) von *Softwareproduktqualität*. Inzwischen abgelöst durch ISO 25010.

Iterative Entwicklung

"Entwicklungsansatz, bei dem Entwicklungsphasen von der Zusammenstellung der Anforderungen bis zur Bereitstellung der Funktionalität in einem funktionierenden Release in *Zyklen* durchlaufen werden." (Übersetztes englisches Zitat von c2-wiki).

Diese Zyklen werden zur Verbesserung von Funktionalität, Qualität oder beidem wiederholt.

Gegensätzlich zur Wasserfall-Entwicklung.

Iterative und inkrementelle Entwicklung

Kombination von iterativen und inkrementellen Ansätzen zur Softwareentwicklung. Sie sind wesentliche Bestandteile verschiedener *agiler* Entwicklungsansätze, z.B. Scrum und XP.

K

Kapselung

Kapselung bezeichnet zwei leicht unterschiedliche Konzepte und manchmal eine Kombination der beiden:

- Einschränkung des Zugriffs auf einige Komponenten des Objekts
- Bündelung von Daten mit Methoden oder Funktionen, die auf diese Daten angewandt werden

Kapselung ist ein Mechanismus zum Verbergen von Informationen.

Kardinalität

Beschreibt die quantitative Bewertung einer Assoziation oder Beziehung. Sie gibt die Zahl der Beteiligten (Objekte, Instanzen, Module usw.) der Assoziation an.

Kerckhoffs'sches Prinzip

Eines der sechs kryptografischen Axiome, die 1883 von dem niederländischen Kryptografen und Linguisten Auguste Kerckhoffs in dem Artikel "La cryptographie militaire" beschrieben wurde. Dieses Axiom ist heute noch relevant und wird daher als "Kerckhoffs’sches Prinzip" bezeichnet.

Es schildert, dass eine kryptografische Methode nicht geheim gehalten werden muss, um die verschlüsselte Botschaft zu schützen.

"Der Feind kennt das System" ist ein weiterer Ausdruck, den der Mathematiker Claude Shannon als Shannons Maxime geprägt hat.

Siehe Bruce Schneiers Crypto-Gram, May 15, 2002

Knoten (in UML)

Eine Verarbeitungsressource (Ausführungsumgebung, Prozessor, Maschine, virtuelle Maschine, Anwendungsserver) zur Verteilung und Ausführung von Artefakten.

Kohäsion

Maß, in dem Elemente eines Bausteins, einer Komponente oder eines Moduls zusammengehören wird Kohäsion genannt. Sie misst die Stärke der Beziehung zwischen Teilen einer Funktionalität in einer gegebenen Komponente. In kohärenten Systemen ist Funktionalität stark verbunden. Sie wird in der Regel als *starke Kohäsion* oder *schwache Kohäsion* charakterisiert. Ziel sollte starke Kohäsion sein, da diese oft mit Wiederverwendbarkeit, loser Kopplung und Verständlichkeit einhergeht.

Kombinator

Entwurfsmuster zum Aufbau komplexer Funktionen oder Objekte durch die Kombination einfacherer Funktionen oder Objekte. Gegeben ein Domänenobjekt vom Typ T, suche nach Operationen, die als Ein- und Ausgabe ebenfalls den Typ T haben. Auch bekannt als *closure of operation*. Siehe [Yorgey 2012] und [Maguire 2019].

Kommando

Entwurfsmuster, bei dem ein Objekt zur Kapselung einer Aktion genutzt wird. Diese Aktion kann später aufgerufen oder ausgeführt werden.

Komplexität

"Komplexität wird im Allgemeinen zur Charakterisierung eines Systems o.Ä. mit vielen Teilen, in dem diese Teile auf unterschiedliche Weise miteinander interagieren, verwendet." (Übersetztes englisches Zitat aus Wikipedia.)

- *Essenzielle* Komplexität ist der Kern des Problems, das es zu lösen gilt, und besteht aus den Teilen der Software, die wirklich schwierige Probleme sind. Den meisten Softwareproblemen wohnt eine gewisse Komplexität inne.
- *Akzidentelle* Komplexität ist alles, was sich nicht notwendigerweise direkt auf die Lösung bezieht, mit dem wir uns aber dennoch befassen müssen.

(Übersetztes englisches Zitat von Mark Needham)

Architekten haben sich um eine Verringerung der akzidentellen Komplexität zu bemühen.

Komponente

Siehe Baustein. Strukturelement einer Architektur.

Komposition

Kombination von einfacheren Elementen (z.B. Funktionen, Datentypen, Bausteinen) zu komplizierteren, leistungsstärkeren oder stärker verantwortlichen Elementen.

In UML: Wenn das enthaltende Element vernichtet wird, werden auch die enthaltenen Elemente vernichtet.

Konsistenz

Ein konsistentes System enthält keine Widersprüche.

- Identische Probleme werden mit identischen (oder zumindest gleichartigen) Ansätzen gelöst.
- Maß, in dem Daten und ihre Beziehungen Validierungsregeln entsprechen.
- Clients (einer Datenbank) erhalten bei identischen Abfragen identische Ergebnisse (z.B. Monotonic-Read-Consistency, Monotonic-Write-Consistency, Read-Your-Writes-Consistency etc.)
- In Bezug auf Verhalten: Maß, in dem ein System sich kohärent, reproduzierbar und vernünftig verhält.

Kontext (eines Systems)

"Definiert die Beziehungen, Abhängigkeiten und Interaktionen zwischen dem System und seiner Umgebung: Menschen, Systeme und externe Entitäten, mit denen es interagiert." (Übersetztes englisches Zitat aus Rozanski-Woods)

Eine weitere Definition von arc42: "System Scope und Kontext - wie der Name schon sagt - grenzen dein System (d.h. deinen Scope) von all seinen Kommunikationspartnern (benachbarte Systeme und Benutzer, d.h. den Kontext deines Systems) ab. Er legt damit die externen Schnittstellen fest." (zitiert aus docs.arc42.org)

Unterscheide zwischen *geschäftlichem* und *technischem* Kontext:

- Der **geschäftliche** Kontext (früher *logischer* Kontext genannt) zeigt die externen Beziehungen aus einer geschäftlichen oder nicht-technischen Perspektive. Er abstrahiert von technischen, Hardware- oder Implementierungsdetails. Input-/Output-Beziehungen werden nach ihrer *geschäftlichen Bedeutung* benannt und nicht nach ihren technischen Eigenschaften.
- Der **technische** Kontext zeigt technische Details, wie Übertragungskanäle, technische Protokolle, IP-Adressen, Busse oder ähnliche Hardware-Details. Eingebettete Systeme zum Beispiel interessieren sich oft schon sehr früh in der Entwicklung für hardwarebezogene Informationen.

Kontextabgrenzung

Auch als Kontextsicht bezeichnet. Zeigt das vollständige System als eine Blackbox in seiner Umgebung. Dies kann entweder aus fachlicher Perspektive (*fachlicher Kontext*) und/oder aus technischer oder Verteilungsperspektive (*technischer Kontext*) erfolgen. Die Kontextabgrenzung (oder Kontextdiagramm) zeigt die Grenzen zwischen einem System und seiner Umgebung und stellt die Entitäten in seiner Umgebung (seine Nachbarn), mit denen es interagiert, dar.

Nachbarn können andere Software, Hardware (wie Sensoren), Menschen, Benutzerrollen oder sogar Organisationen, die das System nutzen, sein.

Siehe Kontext.

Kontextgrenze

Kontextgrenze ist ein Prinzip des Strategieentwurfs von Domain-Driven Design. "Eine Kontextgrenze definiert ausdrücklich den Kontext, in dem ein Domänenmodell für ein Softwaresystem gilt. Idealerweise wäre ein einziges, einheitliches Modell für alle Systeme in derselben Domäne am besten. Dies ist zwar ein ehrenwertes Ziel, aber in Wirklichkeit ist es normalerweise in mehrere Modelle zerstückelt. Es ist sinnvoll, dies so hinzunehmen und damit zu arbeiten." (Übersetztes englisches Zitat aus Wikipedia)

"Bei sämtlichen großen Projekten gibt es mehrere Domänenmodelle. Doch wenn auf unterschiedlichen Modellen basierender Code miteinander kombiniert wird, wird die Software fehlerhaft, unzuverlässig und schwer verständlich. Die Kommunikation der Teammitglieder wird verwirrend. Es ist häufig unklar, in welchem Kontext ein Modell nicht angewandt werden sollte. Daher gilt: Legen Sie in Bezug auf Teamorganisation, Verwendung in spezifischen Teilen der Anwendung und physische Manifestationen, wie Codebasen oder Datenbankschemata, ausdrücklich Grenzen fest. Sorgen Sie dafür, dass das Modell exakt mit diesen Grenzen konsistent ist, aber lassen Sie sich nicht von Themen außerhalb ablenken oder verwirren." (Übersetztes englisches Zitat aus Wikipedia)

Konzept

Plan, Prinzip(ien) oder Regel(n), wie ein spezifisches Problem zu lösen ist.

Konzepte sind häufig **querschnittlich**, in dem Sinne, dass mehrere Architekturelemente von einem einzigen Konzept betroffen sein können. Das heißt, dass Implementierer von z.B. Implementierungseinheiten (Bausteinen) das entsprechende Konzept einhalten sollen.

Konzepte bilden die Basis für konzeptionelle Integrität.

Konzeptionelle Integrität

Konzepte, Regeln, Muster und ähnliche Lösungsansätze werden im gesamten System auf einheitliche (homogene, ähnliche) Weise angewendet. Ähnliche Probleme werden auf ähnliche oder identische Weise

gelöst.

Kopplung

Kopplung ist die Art und der Grad der *Interdependenz* zwischen Software-Bausteinen; ein Maß dafür, wie eng zwei Komponenten verbunden sind.

Ziel sollte immer eine *lose* Kopplung sein. Kopplung steht in der Regel im Gegensatz zu Kohäsion. Lose Kopplung korreliert häufig mit starker Kohäsion. Lose Kopplung ist oft ein Zeichen für ein gut strukturiertes System. Zusammen mit starker Kohäsion unterstützt sie Verständlichkeit und Wartbarkeit.

Korrespondenz

Korrespondenz definiert eine Beziehung zwischen Architekturbeschreibungselementen. Korrespondenzen werden genutzt, um relevante Architekturbeziehungen innerhalb einer Architekturbeschreibung (oder zwischen Architekturbeschreibungen) auszudrücken (gemäß Definition in ISO/IEC/IEEE 42010).

Korrespondenzregel

Korrespondenzen können Korrespondenzregeln unterliegen. Korrespondenzregeln werden genutzt, um Beziehungen innerhalb einer Architekturbeschreibung (oder zwischen Architekturbeschreibungen) durchzusetzen (gemäß Definition in ISO/IEC/IEEE 42010).

Synonym: Integrität, Homogenität, konzeptionelle Integrität.

L

Latenz

Latenz ist die zeitliche Verzögerung zwischen der Ursache und der Wirkung einer Veränderung in einem System.

In Computernetzwerken beschreibt die Latenz die Zeit, die eine Datenmenge (*Paket*) braucht, um von einem bestimmten Ort zu einem anderen zu gelangen.

In interaktiven Systemen ist die Latenz die Zeitspanne zwischen einer Eingabe in das System und der audiovisuellen Reaktion. Oft gibt es eine Verzögerung, die oft durch Netzwerkverzögerungen verursacht wird.

Laufzeitsicht

Zeigt die Zusammenarbeit von Bausteinen (beziehungsweise ihrer Instanzen) zur Laufzeit in konkreten Szenarien. Sollte auf Elemente der Bausteinsicht verweisen. Kann beispielsweise (muss aber nicht) als UML-Sequenz oder Aktivitätsdiagramm ausgedrückt werden.

Lehrplan

Der Lernprozess, der von einer Schule angeboten wird (hier: iSAQB® als Institution, die für die Softwarearchitekturausbildung). Er umfasst den Inhalt der Kurse (den Lehrplan), die angewandten Methoden und andere Aspekte wie Normen und Werte, die sich auf die Art und Weise beziehen, wie die Ausbildung einschließlich Zertifizierung und Prüfung organisiert ist.

Liskovsches Substitutionsprinzip

Bezieht sich auf die objektorientierte Programmierung: Wenn Vererbung genutzt wird, dann richtig: Instanzen von abgeleiteten Typen (Unterklassen) müssen vollständig an die Stelle ihrer Basistypen treten

können. Wenn der Code Basisklassen verwendet, können diese Referenzen durch jede beliebige Instanz einer abgeleiteten Klasse ersetzt werden, ohne dass dies die Funktionalität des Codes beeinträchtigt.

M

Mal-/Zeichenprogramm

Programm zur Erstellung von Zeichnungen, die in der Architekturdokumentation verwendet werden können. Beispiele: MS Visio, OmniGraffle, PowerPoint, etc. Mal-/Zeichenprogramme behandeln jede Zeichnung als separate Sache, was bei der Aktualisierung eines Elements der Architektur, das in mehreren Diagrammen erscheint, zu höheren Wartungskosten führt (anders als ein Modellierungswerkzeug).

MFA

Für Multi-Faktor-Authentifizierung siehe Authentifizierung.

Microservice

Architekturstil, der die Unterteilung von großen Systemen in kleine Einheiten vorschlägt. "Microservices müssen als virtuelle Maschinen, als leichtere Alternativen, wie Docker-Container, oder als individuelle Prozesse implementiert werden. Dadurch können sie leicht einzeln in Produktion genommen werden." (Übersetztes englisches Zitat aus dem (kostenlosen) LeanPub booklet on Microservices von Eberhard Wolff.

Model-View-Controller (MVC)

Architekturmuster, das häufig zur Implementierung von Benutzeroberflächen verwendet wird. Unterteilt ein System in drei miteinander verbundene Teile (Modell / model, Präsentation / view und Steuerung / controller), um die folgenden Verantwortlichkeiten zu trennen:

- Das Modell verwaltet Daten und Logik des Systems. Die "Wahrheit", die von einer oder vielen Präsentationen gezeigt oder angezeigt wird. Das Modell kennt seine Präsentationen nicht (und ist nicht von ihnen abhängig).
- Die Präsentation kann eine Reihe von (beliebigen) Output-Darstellungen der (Modell-)Informationen sein. Mehrere Präsentationen desselben Modells sind möglich.
- Die Steuerung akzeptiert (Benutzer-)Eingaben und wandelt diese in Befehle für das Modell oder die Präsentation um.

Model-View-Update (MVU)

Architekturmuster, das häufig zur Implementierung von Benutzeroberflächen verwendet wird. Es beruht auf unveränderbaren Daten und unidirektionalem Datenfluss. Es besteht aus drei Teilen:

- Model repräsentiert den Zustand der Anwendung als unveränderbare Datenstruktur.
- View ist eine Funktion ohne Seiteneffekte, die das Model in der UI darstellt.
- Update ist eine Funktion, die Aktualisierungen des Models verarbeitet, indem sie eine neue Instanz des Models erzeugt.

Model-View-ViewModel (MVVM)

Architekturmuster, das häufig zur Implementierung von Benutzeroberflächen verwendet wird. Es unterteilt eine Anwendung in drei Teile (Model, View, und ViewModel):

- Model verwaltet die Daten und die Domänenlogik des Systems. Hängt nicht von View und ViewModel ab.
- View ist die sichtbare Benutzeroberfläche der Anwendung (oder Teilen davon).
- ViewModel dient als Vermittler zwischen View und Model und enthält die UI-Logik. Kann vom Model abhängen, aber nicht vom View.

Modellart

Konventionen für einen Modellierungstyp (gemäß Definition in ISO/IEC/IEEE 42010).

Beispiele für Modellarten sind Datenflussdiagramme, Klassendiagramme, Petri-Netze, Bilanzen, Organigramme und Zustandsübergangsmodelle.

Modellgetriebene Architektur

Modellgetriebene Architektur / Model Driven Architecture (MDA) ist ein OMG-Standard für die modellbasierte Softwareentwicklung. Definition: Ein Ansatz zur IT-Systemspezifikation, bei dem die Spezifikation der Funktionalität von der Spezifikation der Implementierung dieser Funktionalität auf einer spezifischen Technologieplattform getrennt wird.

Modellgetriebene Softwareentwicklung / Model-driven software development (MDSD)

Die zugrunde liegende Idee besteht darin, Code aus abstrakteren Anforderungsmodellen oder der Domäne zu generieren.

Modellierungswerkzeug

Ein Werkzeug, das Modelle erstellt (z.B. UML- oder BPMN-Modelle). Kann zur Erstellung von konsistenten Diagrammen zur Dokumentation verwendet werden, da es den Vorteil hat, dass jedes Modellelement nur einmal vorhanden ist, aber in vielen Diagrammen konsistent angezeigt wird (anders als bei einem einfachen Mal-/Zeichenprogramm).

Modul

(Siehe auch Modulare Programmierung)

1. Strukturelement oder Baustein, üblicherweise als *Blackbox* angesehen, mit einer klar definierten Verantwortlichkeit. Kapselt Daten und Code und bietet öffentliche Schnittstellen, sodass Clients auf seine Funktionalität zugreifen können. Diese Bedeutung wurde erstmals in einem bahnbrechenden Grundlagenpapier von David L. Parnas beschrieben: On the Criteria to be Used in Decomposing Software into Modules
2. In mehreren Programmiersprachen ist ein *Modul* ein Konstrukt zur Zusammenstellung kleinerer Programmiereinheiten, z.B. in Python. In anderen Sprachen (wie Java) werden Module *Pakete* genannt.
3. Das CPSA®-Advanced Level ist derzeit in mehrere Module unterteilt, die getrennt und in beliebiger Reihenfolge gelernt oder unterrichtet werden können. Die genauen Beziehungen zwischen diesen Modulen und die Inhalte dieser Module sind in den jeweiligen Lehrplänen festgelegt.

Modulare Programmierung

"Softwareentwurfstechnik, die die Funktionalität eines Programms in unabhängige, austauschbare *Module* unterteilt, sodass jedes Modul alles enthält, was zur Ausführung von nur einem Aspekt der gewünschten Funktionalität erforderlich ist.

Module haben *Schnittstellen*, die die vom Modul bereitgestellten und benötigten Elemente angeben. Die in der Schnittstelle definierten Elemente können von anderen Modulen erkannt werden." (Übersetztes englisches Zitat aus Wikipedia)

Muster

Wiederverwendbare oder wiederholbare Lösung für ein gängiges Problem beim Softwareentwurf oder in der Softwarearchitektur.

Siehe Architekturmuster oder Entwurfsmuster.

N

Nebenläufigkeit

Nebenläufigkeit ist die Möglichkeit, verschiedene Teile oder Einheiten eines Programms, Algorithmus oder Problems in beliebiger Reihenfolge oder in Teilreihenfolge auszuführen, ohne das Endergebnis zu beeinflussen. Aus Nebenläufigkeit folgt dabei nicht zwangsläufig Parallelität. (Übersetzung nach Wikipedia)

Netz des Vertrauens (EN: Web of Trust)

Da eine einzelne CA ein leichtes Ziel für einen Angreifer sein könnte, delegiert ein Netz des Vertrauens die Begründung des Vertrauens an den Benutzer. Jeder Benutzer entscheidet, in der Regel durch Überprüfung eines Fingerprints eines Schlüssels, welchem Identitätsnachweis anderer Nutzer er vertraut. Dieses Vertrauen wird durch die Signatur des Schlüssels des anderen Benutzers, der ihn dann mit der zusätzlichen Signatur veröffentlichen kann, ausgedrückt. Ein dritter Benutzer kann dann diese Signatur überprüfen und entscheiden, ob er der Identität vertraut oder nicht.

Die E-Mail-Verschlüsselung PGP ist ein Beispiel für eine auf einem Netz des Vertrauens basierende PKI.

Nichtfunktionale Anforderung

Anforderungen, die *die Lösung einschränken*. Nichtfunktionale Anforderungen werden auch als Qualitätsanforderungen bezeichnet. Der Begriff nichtfunktional ist eigentlich irreführend, da viele der betreffenden *Eigenschaften* sich direkt auf spezifische System*funktionen* beziehen, weshalb sie im modernen Anforderungsmanagements gerne als *vorgegebene Randbedingungen* bezeichnet werden.

Node (Node.js)

In der modernen Webentwicklung: Kurz für die quelloffene JavaScript-Laufzeitumgebung Node.js®, die auf V8 JavaScript von Chrome aufbaut. Node.js ist für sein ereignisgesteuertes, nicht blockierendes E/A-Modell und sein großes Ökosystem unterstützender Bibliotheken bekannt.

Notation

Ein System aus Zeichen, Symbolen, Bildern oder Schriftzeichen zur Darstellung von Informationen. Beispiele: Fließtexte, Tabellen, Stichpunktlisten, nummeriere Listen, UML, BPMN.

Nutzungsbeziehung

Abhängigkeit zwischen zwei Bausteinen. Wenn A B nutzt, dann hängt die Ausführung von A von der Anwesenheit einer korrekten Implementierung von B ab.

O

Onion Architektur

See Ports und Adapter.

Open-Close-Prinzip (OCP)

Softwareentitäten (Klassen, Module, Funktionen usw.) sollten für Erweiterungen offen, aber für Modifikationen geschlossen sein (Bertrand Meyer, 1998). Einfach gesagt: Um eine Funktionalität zu einem System *hinzuzufügen* (Erweiterung), sollte *keine Modifikation* des vorhandenen Codes erforderlich sein. Teil der "SOLID-"Prinzipien von Robert Martin für objektorientierte Systeme. Kann in objektorientierten Sprachen durch Schnittstellenvererbung, allgemeiner als *Plugins*, implementiert werden.

OWASP

Das **Open Web Application Security Project** ist eine 2001 gegründete, weltweite, gemeinnützige Onlineorganisation zur Verbesserung der Softwaresicherheit. Es ist eine reichhaltige Quelle für Informationen und beste Praktiken im Bereich Websicherheit. Siehe https://www.owasp.org/.

Die OWASP-Top-10 ist eine häufig angeführte Liste von Angriffskategorien basierend auf der Datenerhebung des Projekts.

P

Packaging-Prinzipien

Grundsätze für die Gestaltung der Struktur von Softwaresystemen ([Martin 2003]):

- Reuse-Release-Equivalence-Prinzip (REP)
- Common-Reuse-Prinzip (CRP)
- Common-Closure-Prinzip (CCP)
- Acyclic-Dependencies-Prinzip (ADP)
- Stable-Dependencies-Prinzip (SDP)
- Stable-Abstractions-Prinzip (SAP)

Robert C. Martin, der das Akronym "SOLID" geprägt hat, hat auch die Packaging-Prinzipien eingeführt und häufig beide zusammen angeführt. Während die SOLID-Prinzipien auf die Klassenebene abzielen, beziehen sich die Packaging-Prinzipien auf die Ebene größerer Komponenten, die mehrere Klassen enthalten und eventuell unabhängig verteilt werden.

Package- und SOLID-Prinzipien haben beide das ausdrückliche Ziel, Software wartbar zu halten und die Anzeichen von schlechtem Design, Rigidität, Fragilität, Immobilität und Viskosität zu vermeiden.

Martin hat die Packaging-Prinzipien zwar bezogen auf große Komponenten formuliert, sie gelten jedoch auch für alle anderen Größen. Ihr Kern sind universelle Prinzipien, wie lose Kopplung, eindeutige Verantwortung, hierarchische (azyklische) Zerlegung und die Erkenntnis, dass sinnvolle Abhängigkeiten von spezifischen/instabilen Konzepten zu abstrakteren/stabileren verlaufen (was sich im DIP wiederfindet).

Perfect Forward Secrecy / Perfekte vorwärts gerichtete Geheimhaltung

Eigenschaft eines kryptografischen Protokolls, die darin besteht, dass ein Angreifer durch Kompromittierung von Langzeitschlüsseln keine Informationen über Kurzzeit-Sitzungsschlüssel erhalten

kann.

Beispiele für Protokolle mit perfekter vorwärts gerichteter Geheimhaltung sind TLS und OTR. Wenn diese Funktion für TLS aktiviert ist und ein Angreifer Zugriff auf den privaten Schlüssel erhält, können früher aufgezeichnete Kommunikationssitzungen dennoch nicht entschlüsselt werden.

Perspektive

Eine Perspektive dient der Berücksichtigung einer Reihe von zusammenhängenden Qualitätseigenschaften und Belangen eines Systems.

Architekten wenden Perspektiven iterativ auf die *Architektursichten* eines Systems an, um die Auswirkungen von *Architekturentwurfsentscheidungen* über mehrere *Blickwinkel* und *Architektursichten* hinweg zu beurteilen.

[Rozanski+2011] verbindet mit dem Begriff *Perspektive* auch Aktivitäten, Taktiken und Richtlinien, die zu berücksichtigen sind, wenn ein System eine Reihe von zusammenhängenden Qualitätseigenschaften erfüllen soll, und schlägt folgende Perspektiven vor:

- Zugänglichkeit
- Verfügbarkeit und Resilienz
- Entwicklungsressource
- Weiterentwicklung
- Internationalisierung
- Standort
- Performance und Skalierbarkeit
- Regulierung
- Sicherheit
- Benutzerfreundlichkeit

Pikachu

Eine gelbliche mausähnliche Figur aus der (recht berühmten) Pokémon-Welt. Das brauchen Sie eigentlich nicht zu wissen. Aber es schadet auch nicht – und vielleicht beeindrucken Sie Ihre Kinder mit diesem Wissen…

Pipe

Verbindung im "Pipes und Filter"-Architekturstil, die Datenströme oder -blöcke von der Ausgabe eines Filters zur Eingabe eines anderen Filters überträgt, ohne Werte oder die Datenreihenfolge zu verändern. Siehe Pipes und Filter

Pipes und Filter

Das Architekturmuster "Pipes und Filter" bietet eine Struktur für Systeme, die Datenströme verarbeiten. Jeder Verarbeitungsschritt wird als eine Filterkomponente gekapselt, dabei werden die Daten zwischen benachbarten Filtern durch Pipes geleitet. Andere Kombinationen der Filter führt zu einer Variente des Systems. (Zitiert aus [Buschmann+1996], siehe auch Pipe und Filter.)

PKI

Abkürzung von **Public-Key-Infrastruktur**. Ein Konzept zum Management von digitalen Zertifikaten, das üblicherweise asymmetrische Kryptografie nutzt. Der Begriff "public" (öffentlich) bezieht sich zumeist auf die Art des verwendeten Kryptografieschlüssels und nicht notwendigerweise auf eine öffentlich zugängliche Infrastruktur. Zur Vermeidung von Begriffsverwirrungen kann "offene PKI" oder "geschlossene PKI" verwendet werden, vgl. [Anderson 2008] Kapitel 21.4.5.7 PLI, Seite 672.

Eine PKI basiert in der Regel auf einer CA oder einem Netz des Vertrauens.

Plugin

Architekturmuster, das die Erweiterung der Funktionalität einer Anwendung ermöglicht, ohne den Kern der Anwendung zu verändern. Dies wird durch externe Module, sogenannte Plugins, erreicht, welche Funktionen hinzuzufügen oder mit der Kernanwendung interagieren.

Port

UML-Konstrukt, das in Komponentendiagrammen verwendet wird. Eine Schnittstelle, die einen Punkt, an dem eine Komponente mit ihrer Umgebung interagiert, definiert.

Ports und Adapter (PAC)

Architekturmuster, das die Domänenlogik im Zentrum des Systems hält und Verbindungen zur Außenwelt nur über Ports herstellt. Ports sind unabhängig von einer bestimmten Technologie. An die Ports können verschiedene Adapter angeschlossen werden, um Anfragen und Antworten für eine bestimmte Technologie bereitzustellen. Dieser Ansatz ermöglicht es, eine Anwendung von verschiedenen Agenten (z. B. Benutzer, Programme, automatisierte Tests) steuern zu lassen und isoliert von ihrer Produktionsumgebung zu entwickeln und testen. Siehe [Cockburn 2005], [Lange 2021], [Hombergs 2024].

Auch bekannt als Onion Architecture, Hexagonale Architektur, Clean Architecture.

POSA

Pattern-oriented Software Architecture. Buchreihe zu Softwarearchitekturmustern.

Presentation-Abstraction-Control (PAC)

Architekturmuster, welches ein interaktives Softwaresystemen in eine Hierarchie von kooperierenden Agenten aufteilt. Jeder Agent besteht aus drei verschiedene Komponenten:

- Presentation ist die Benutzeroberfläche des Agenten.
- Abstraction beinhaltet die Domänenlogik und die Daten.
- Control behandelt Interaktionen zwischen Presentation und Abstraction.

Principal

Im Sicherheitskontext sind Principals Entitäten, die authentifiziert wurden und denen Berechtigungen zugewiesen werden können. Principals können Benutzer, aber auch andere Dienste oder ein auf einem System laufender Prozess sein. Der Begriff wird in der Java-Umgebung und in verschiedenen Authentifizierungsprotokollen verwendet (siehe GSSAPI RFC2744 oder Kerberos RFC4121).

Produkt

Produktdaten sind solche, deren Werte mehrere, feststehende Attribute haben. Auch als **Records**, **Structs**, **Tupel** oder **Und-Daten** bezeichnet.

Beispiel:

Eine **Adresse** hat die folgenden Attribute:

- Name und
- Straße und
- Hausnummer und
- Stadt und
- Postleitzahl

Siehe auch [Sperber+2024].

Pseudo-Zufälligkeit

Häufig in Verbindung mit Pseudozufallszahlengeneratoren verwendet. Die Erzeugung von Zufälligkeit mit hoher Entropie ist ressourcenintensiv und, abgesehen von Kryptografie, nicht für viele Anwendungen erforderlich. Zur Behebung dieses Problems werden Pseudozufallszahlengeneratoren mit einen Daten-Startwert initialisiert und erzeugen basierend auf diesem Startwert zufällige Werte. Die Daten werden zufällig erzeugt, aber sind immer gleich, wenn der Generator mit dem gleichen Startwert initialisiert wird. Dies wird als Pseudo-Zufälligkeit bezeichnet und ist weniger leistungsintensiv.

Q

Qualitative Bewertung

Erkennung von Risiken bezüglich der gewünschten Qualitätsmerkmale eines Systems. Analyse oder Beurteilung, ob ein System oder seine Architektur die gewünschten oder geforderten Qualitätsziele erreichen kann.

Statt mit der Berechnung oder Messung bestimmter Eigenschaften von Systemen oder Architekturen befasst sich die qualitative Bewertung mit Risiken, Kompromissen und Sensitivitätspunkten.

Siehe auch Beurteilung.

Qualität

Siehe Softwarequalität und Qualitätsmerkmale.

Qualitätsanforderung

Eigenschaft oder Merkmal einer Komponente eines Systems. Beispiele sind Laufzeitleistung, Schutz, Sicherheit, Zuverlässigkeit oder Wartbarkeit. Siehe auch Softwarequalität.

Qualitätsbaum

(Syn.: Qualitätsattributbaum). Ein hierarchisches Modell zur Beschreibung von Produktqualität: Die Wurzel "Qualität" wird hierarchisch in *Bereiche* oder Themen verfeinert, welche wiederum verfeinert werden. Qualitätsszenarien bilden die Blätter dieses Baums.

- Standards zu Produktqualität, wie ISO 25010, enthalten Vorschläge von *allgemeinen* Qualitätsbäumen.
- Die Qualität eines spezifischen Systems kann mit einem *spezifischen* Qualitätsbaum beschrieben werden (siehe nachfolgendes Beispiel).

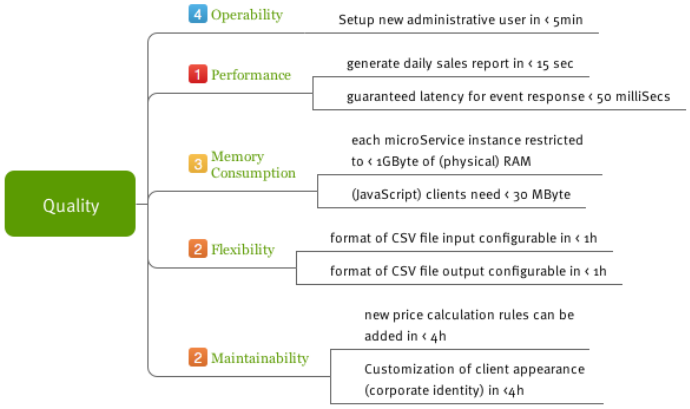

Figure 1. Beispiel eines Qualitätsbaums

Qualitätseigenschaft

Synonym: Qualitätsmerkmal.

Qualitätsmerkmal

Die Softwarequalität ist das Maß, in dem ein System die gewünschte Kombination von *Merkmalen* besitzt. (Siehe: Softwarequalität).

In der ISO-25010 Norm sind folgende Qualitätsmerkmale definiert:

- Funktionale Eignung
  - Funktionale Vollständigkeit,
  - Funktionale Korrektheit
  - Funktionale Angemessenheit
- Leistungseffizienz
  - Zeitverhalten
  - Ressourcenverbrauch
  - Kapazität
- Kompatibilität
  - Koexistenz
  - Interoperabilität
- Benutzerfreundlichkeit
  - Erkennbarkeit der Brauchbarkeit
  - Erlernbarkeit
  - Bedienbarkeit
  - Schutz vor Fehlbedienung
  - Ästhetik der Benutzeroberfläche
  - Zugänglichkeit

- Zuverlässigkeit
  - Verfügbarkeit
  - Fehlertoleranz
  - Wiederherstellbarkeit
- Sicherheit
  - Vertraulichkeit
  - Integrität
  - Nichtabstreitbarkeit
  - Verantwortlichkeit
  - Authentifizierbarkeit
- Wartbarkeit
  - Modularität
  - Wiederverwendbarkeit
  - Analysierbarkeit
  - Modifizierbarkeit
  - Testbarkeit
- Portierbarkeit
  - Adaptierbarkeit
  - Installierbarkeit
  - Austauschbarkeit

Es kann hilfreich sein, zwischen folgenden Typen von Merkmalen zu unterscheiden:

- *Laufzeitqualitätsmerkmale*, die während der Ausführungszeit des Systems beobachtet werden können,
- *Nicht-Laufzeitqualitätsmerkmale*, die während der Ausführung des Systems nicht beobachtet werden können, und
- *Geschäftsqualitätsmerkmalen*, wie Kosten, Zeitplan, Marktfähigkeit, Eignung für Unternehmen.

Beispiele für Laufzeitqualitätsmerkmale sind Leistungseffizienz, Sicherheit, Zuverlässigkeit, Benutzungsfreundlichkeit und Robustheit.

Beispiele für Nicht-Laufzeitqualitätsmerkmale sind Modifizierbarkeit, Portierbarkeit, Wiederverwendbarkeit und Testbarkeit.

Qualitätsmerkmal Adaptierbarkeit

Maß, in dem sich ein Produkt oder System effektiv und effizient an unterschiedliche oder sich weiterentwickelnde Hardware, Software oder sonstige Betriebs- oder Nutzungsumgebungen anpassen lässt. Teilmerkmal von: Portierbarkeit.

Siehe auch [ISO-25010].

Qualitätsmerkmal Analysierbarkeit

Maß der Effektivität und Effizienz, mit dem die Auswirkung eine geplanten Änderung an einem oder mehreren seiner Teile auf ein Produkt oder System beurteilt, die Mängel oder Fehlerursachen eines Produkts diagnostiziert oder zu modifizierende Teile identifiziert werden können. Teilmerkmal von: Wartbarkeit.

Siehe auch [ISO-25010].

Qualitätsmerkmal Austauschbarkeit

Maß, in dem ein Produkt ein anderes spezifiziertes Softwareprodukt für den gleichen Zweck in der gleichen Umgebung ersetzen kann. Teilmerkmal von: Portierbarkeit.

Siehe auch [ISO-25010].

Qualitätsmerkmal Authentifizierbarkeit

Maß, inwieweit nachgewiesen werden kann, dass die Identität eines Subjekts oder einer Ressource der Identitätsbehauptung entspricht. Teilmerkmal von: Sicherheit.

Siehe auch [ISO-25010].

Qualitätsmerkmal Bedienbarkeit

Maß, in dem ein Produkt oder System Eigenschaften aufweist, die es einfach bedien- und steuerbar machen. Teilmerkmal von: Benutzerfreundlichkeit.

Siehe auch [ISO-25010].

Qualitätsmerkmal Benutzerfreundlichkeit

Maß, in dem ein Produkt oder System von spezifizierten Benutzern effektiv, effizient, und zufriedenstellend zur Erreichung von spezifizierten Zielen in einem spezifizierten Nutzungskontext genutzt werden kann. Es besteht aus folgenden Teilmerkmalen: Erkennbarkeit der Brauchbarkeit, Erlernbarkeit, Bedienbarkeit, Schutz vor Fehlbedienung, Ästhetik der Benutzeroberfläche, Zugänglichkeit.

Siehe auch [ISO-25010].

Qualitätsmerkmal Erkennbarkeit der Brauchbarkeit

Maß, in dem Benutzer erkennen können, ob ein Produkt oder System für ihre Bedürfnisse geeignet ist. Teilmerkmal von: Benutzerfreundlichkeit.

Siehe auch [ISO-25010].

Qualitätsmerkmal Erlernbarkeit

Maß, in dem ein Produkt oder System von spezifizierten Benutzern verwendet werden kann, um spezifizierte Lernziele zur Nutzung des Produkts oder Systems in einem spezifizierten Nutzungskontext effektiv, effizient, risikofrei und zufriedenstellend zu erreichen. Teilmerkmal von: Benutzerfreundlichkeit.

Siehe auch [ISO-25010].

Qualitätsmerkmal Fehlertoleranz

Maß, in dem ein System, ein Produkt oder eine Komponente trotz Hardware- oder Softwarefehlern wie vorgesehen funktioniert. Teilmerkmal von: Zuverlässigkeit.

Siehe auch [ISO-25010].

Qualitätsmerkmal funktionale Angemessenheit

Maß, in dem die Funktionen die Erfüllung von spezifizierten Aufgaben und Zielen ermöglichen. Teilmerkmal von: Funktionale Eignung.

Siehe auch [ISO-25010].

Qualitätsmerkmal funktionale Eignung

Maß, in dem ein Produkt oder System bei Nutzung unter spezifizierten Bedingungen Funktionen liefert, die festgelegte und vorausgesetzte Erfordernisse erfüllen. Es besteht aus folgenden Teilmerkmalen: funktionale Vollständigkeit, funktionale Korrektheit, funktionale Angemessenheit.

Siehe auch [ISO-25010].

Qualitätsmerkmal funktionale Korrektheit

Maß, in dem ein Produkt oder System die korrekten Ergebnisse mit dem benötigten Grad an Präzision liefert. Teilmerkmal von: Funktionale Eignung.

Siehe auch [ISO-25010].

Qualitätsmerkmal funktionale Vollständigkeit

Maß, in dem der Satz von Funktionen alle spezifizierten Aufgaben und Benutzerziele abdeckt. Teilmerkmal von: Funktionale Eignung.

Siehe auch [ISO-25010].

Qualitätsmerkmal Installierbarkeit

Maß der Effektivität und Effizienz, mit dem ein Produkt oder ein System in einer spezifizierten Umgebung erfolgreich installiert und/oder deinstalliert werden kann. Teilmerkmal von: Portierbarkeit.

Siehe auch [ISO-25010].

Qualitätsmerkmal Integrität

Maß, in dem ein System, ein Produkt oder eine Komponente den unbefugten Zugriff auf Computerprogramme oder Daten oder deren unbefugte Abänderung verhindert.

Teilmerkmal von: Sicherheit.

Siehe auch [ISO-25010].

Qualitätsmerkmal Interoperabilität

Maß, in dem zwei oder mehr Systeme, Produkte oder Komponenten Informationen austauschen und die ausgetauschten Informationen nutzen können. Teilmerkmal von: Kompatibilität.

Siehe auch [ISO-25010].

Qualitätsmerkmal Kapazität

Maß, in dem die Höchstgrenzen eines Produkt- oder Systemparameters den Anforderungen entsprechen. Teilmerkmal von: Leistungseffizienz.

Siehe auch [ISO-25010].

Qualitätsmerkmal Koexistenz

Maß, in dem ein Produkt, während es sich eine gemeinsame Umgebung und Ressourcen mit anderen Produkten teilt, ohne nachteilige Auswirkungen auf andere Produkte seine geforderten Funktionen effizient erfüllen kann. Teilmerkmal von: Kompatibilität.

Siehe auch [ISO-25010].

Qualitätsmerkmal Kompatibilität

Maß, in dem ein Produkt, ein System oder eine Komponente Informationen mit anderen Produkten, Systemen oder Komponenten austauschen und/oder ihre geforderten Funktionen erfüllen können, während sie sich eine Hardware- oder Softwareumgebung teilen. Es besteht aus folgenden Teilmerkmalen: Koexistenz Interoperabilität.

Siehe auch [ISO-25010].

Qualitätsmerkmal Leistungseffizienz

Leistung im Verhältnis zur Menge der unter angegebenen Bedingungen genutzten Ressourcen.

Ressourcen können andere Softwareprodukte, die Software- und Hardwarekonfiguration des Systems und Materialien (z.B. Druckerpapier, Speichermedien) umfassen.

Es besteht aus folgenden Teilmerkmalen: Zeitverhalten, Ressourcenverbrauch, Kapazität.

Siehe auch [ISO-25010].

Qualitätsmerkmal Modifizierbarkeit

Maß, in dem ein Produkt oder System effektiv und effizient modifiziert werden kann, ohne dass Fehler eingebracht werden oder die bestehende Produktqualität beeinträchtigt wird. Teilmerkmal von: Wartbarkeit.

Siehe auch [ISO-25010].

Qualitätsmerkmal Modularität

Maß, inwieweit ein System oder Computerprogramm aus diskreten Komponenten besteht, sodass eine Änderung an einer Komponente minimale Auswirkungen auf andere Komponenten hat. Teilmerkmal von: Wartbarkeit.

Siehe auch [ISO-25010].

Qualitätsmerkmal Nichtabstreitbarkeit

Maß, in dem nachgewiesen werden kann, dass Maßnahmen oder Ereignisse stattgefunden haben, so dass sie später nicht bestritten werden können. Teilmerkmal von: Sicherheit.

Siehe auch [ISO-25010].

Qualitätsmerkmal Portierbarkeit

Maß der Effektivität und Effizienz, mit dem ein System, ein Produkt oder eine Komponente von einer Hardware-, Software- oder sonstigen Betriebs- oder Nutzungsumgebung in eine andere übertragen werden kann. Es besteht aus folgenden Teilmerkmalen: Adaptierbarkeit, Installierbarkeit, Austauschbarkeit.

Siehe auch [ISO-25010].

Qualitätsmerkmal Reifegrad

Maß, in dem ein System, ein Produkt oder eine Komponente im Normalbetrieb die Zuverlässigkeitsanforderungen erfüllt. Teilmerkmal von: Zuverlässigkeit.

Siehe auch [ISO-25010].

Qualitätsmerkmal Ressourcenverbrauch

Maß, in dem die von einem Produkt oder einem System bei der Erfüllung seiner Funktionen verbrauchten Mengen und Arten von Ressourcen den Anforderungen entsprechen. Teilmerkmal von: Leistungseffizienz.

Siehe auch [ISO-25010].

Qualitätsmerkmal Schutz vor Fehlbedienung

Maß, in dem ein System Benutzer:innen davor schützt, Fehler zu machen. Teilmerkmal von: Benutzerfreundlichkeit.

Siehe auch [ISO-25010].

Qualitätsmerkmal Sicherheit

Maß, in dem ein Produkt oder System Informationen und Daten schützt, sodass Personen oder andere Produkte oder Systeme den ihren Berechtigungsarten oder -stufen entsprechenden Datenzugriffsgrad haben. Es besteht aus folgenden Teilmerkmalen: Vertraulichkeit, Integrität, Nichtabstreitbarkeit, Verantwortlichkeit, Authentifizierbarkeit.

Siehe auch [ISO-25010].

Qualitätsmerkmal Testbarkeit

Maß an Effektivität und Effizienz, mit welchem Testkriterien für ein System, ein Produkt oder eine Komponente festgelegt und Tests durchgeführt werden können, um zu ermitteln, ob diese Kriterien erfüllt sind. Teilmerkmal von: Wartbarkeit.

Siehe auch [ISO-25010].

Qualitätsmerkmal Verantwortlichkeit

Maß, in dem Aktionen einer Entität eindeutig zu der Entität zurückverfolgt werden können. Teilmerkmal von: Sicherheit.

Siehe auch [ISO-25010].

Qualitätsmerkmal Verfügbarkeit

Maß, in dem ein System, ein Produkt oder eine Komponente einsatzfähig und zugänglich sind, wenn sie

benötigt werden. Teilmerkmal von: Zuverlässigkeit.

Siehe auch [ISO-25010].

Qualitätsmerkmal Vertraulichkeit

Maß, in dem ein Produkt oder System sicherstellt, dass nur zugriffsberechtigte Zugriff auf die Daten haben. Teilmerkmal von: Sicherheit.

Siehe auch [ISO-25010].

Qualitätsmerkmal Wartbarkeit

Grad an Effektivität und Effizienz, mit dem ein Produkt modifiziert werden kann, um es zu verbessern, zu korrigieren oder an Veränderungen der Umgebung oder der Anforderungen anzupassen. Es besteht aus folgenden Teilmerkmalen: Modularität, Wiederverwendbarkeit, Analysierbarkeit, Modifizierbarkeit, Testbarkeit.

Siehe auch [ISO-25010].

Qualitätsmerkmal Wiederherstellbarkeit

Maß, in dem ein Produkt oder System im Falle einer Unterbrechung oder eines Fehlers die direkt betroffenen Daten und den gewünschten Systemstatus wiederherstellen kann.

Teilmerkmal von: Zuverlässigkeit. Siehe auch [ISO-25010].

Qualitätsmerkmal Wiederverwendbarkeit

Maß, in dem ein Asset in mehr als einem System oder zum Aufbau anderer Assets genutzt werden kann. Teilmerkmal von: Wartbarkeit.

Siehe auch [ISO-25010].

Qualitätsmerkmal Zeitverhalten

Maß, in dem die Antwort- und Verarbeitungszeiten und Durchsatzgeschwindigkeiten eines Produkts oder System bei der Erfüllung seiner Funktionen den Anforderungen entsprechen. Teilmerkmal von: Leistungseffizienz.

Siehe auch [ISO-25010].

Qualitätsmerkmal Zugänglichkeit

Maß, in dem ein Produkt oder System von Personen mit einer großen Bandbreite von Eigenschaften und Fähigkeiten zur Erreichung eines spezifizierten Ziels in einem spezifizierten Nutzungskontext genutzt werden kann. Teilmerkmal von: Benutzerfreundlichkeit.

Siehe auch [ISO-25010].

Qualitätsmerkmal Zuverlässigkeit

Maß, in dem ein System, ein Produkt oder eine Komponente unter spezifizierten Bedingungen für eine spezifizierte Zeitdauer spezifizierte Funktionen erfüllt. Es besteht aus folgenden Teilmerkmalen: Reifegrad, Verfügbarkeit, Fehlertoleranz, Wiederherstellbarkeit.

Siehe auch [ISO-25010].

Qualitätsmerkmal Ästhetik der Benutzeroberfläche

Maß, in dem eine Benutzeroberfläche dem Benutzer eine angenehme und zufriedenstellende Interaktion ermöglicht. Teilmerkmal von: Benutzerfreundlichkeit.

Siehe auch [ISO-25010].

Qualitätsmodell

(Aus ISO 25010) Ein Modell, das sich auf die statischen Eigenschaften von Software und die dynamischen Eigenschaften von Computersystemen und Softwareprodukten beziehende Qualitätseigenschaften definiert. Das Qualitätsmodell liefert eine konsistente Terminologie zur Spezifikation, Messung und Bewertung der System- und Softwareproduktqualität.

Der Anwendungsumfang von Qualitätsmodellen umfasst die Unterstützung der Spezifikation und Bewertung von Software und softwareintensiven Computersystemen aus unterschiedlichen Perspektiven durch an ihrem Erwerb, ihren Anforderungen, ihrer Entwicklung, ihrer Nutzung, ihrer Bewertung, ihrem Support, ihrer Wartung, ihrer Qualitätssicherung und -kontrolle sowie ihrem Audit beteiligte Personen.

Kommentar (Gernot Starke)

Ein Qualitätsmodell (wie ISO-25010) liefert **nur** eine Taxonomie von Begriffen, aber **keine** Mittel zur Spezifikation oder Bewertung von Qualität. Ich stimme der obigen Formulierung "einheitliche Terminologie" zu, lehne aber "Messung und Bewertung" entschieden ab. Zum Messen und Bewerten braucht man definitiv zusätzliche Werkzeuge und/oder Methoden, das reine Qualitätsmodell hilft da nicht weiter.

Quantitative Bewertung

(Syn.: quantitative Analyse): Messung oder Zählung von Werten von Softwareartefakten, z.B. Kopplung, zyklomatische Komplexität, Größe, Testabdeckung. Kennzahlen wie diese helfen bei der Identifizierung von kritischen Teilen oder Elementen von Systemen.

Querschnittsbelang

Funktionalität der Architektur oder des Systems, die mehrere Elemente betrifft. Beispiele für diese Belange sind Logging, Transaktionen, Sicherheit, Ausnahmebehandlung, Caching etc.

Siehe auch Konzept.

Querschnittskonzept

Siehe Konzept

Synonym: Prinzip, Regel.

R

RBAC (Role Based Access Control / Rollenbasierte Zugriffskontrolle)

Eine Rolle ist ein fester Satz an Berechtigungen, der üblicherweise einer Gruppe von Principals zugewiesen wird. So kann eine rollenbasierte Zugriffskontrolle zumeist effizienter umgesetzt werden als ein ACL -basiertes System und ermöglicht beispielsweise Vertreterreglungen.

Redesign

Die Veränderung von Softwareeinheiten, sodass sie den gleichen Zweck wie zuvor erfüllen, jedoch auf andere Weise und gegebenenfalls mit anderen Mitteln. Häufig fälschlicherweise Refactoring genannt.

Refactoring

Begriff zur Bezeichnung der Verbesserung von Softwareeinheiten durch Veränderung ihrer internen Struktur ohne Veränderung des Verhaltens. (vgl.: "Refactoring ist der Prozess der Änderung eines Softwaresystems, sodass sich das externe Verhalten des Codes nicht verändert, aber die interne Struktur verbessert wird." – Refactoring, Martin Fowler, 1999)

Nicht mit Re**design** zu verwechseln.

Registry

"Sehr bekanntes Objekt, das andere Objekte nutzen können, um gemeinsame Objekte und Dienste zu finden." (Übersetztes englisches Zitat PoEAA). Häufig als Singleton (auch ein bekanntes Entwurfsmuster) implementiert.

Remote Procedure Call (RPC)

Mechanismus zur Kommunikation zwischen zwei Systemen, der es einem Programm ermöglicht, eine Prozedur in einem anderen Adressraum auszuführen (in der Regel auf einer anderen Maschine). Die Kommunikation erfolgt dabei so, also ob es sich um einen lokalen Prozeduraufruf handelt. Dabei wird die Komplexität der Netzwerkkommunikation versteckt. RPC wird häufig in verteilten Systemen und Netzwerkanwendungen eingesetzt.

Reuse-Release-Equivalence-Prinzip

Ein Grundsatz für die Gestaltung der Struktur von Softwaresystemen (siehe auch Packaging-Prinzipien). Es verlangt einen "Release" und eine Versionskontrolle von großen Komponenten, insbesondere wenn das System sie von mehreren Punkten aus nutzt. Auch wenn sie nicht öffentlich herausgegeben werden, sollten diese Komponenten aus dem System extrahiert werden und durch einen externen Dependency Manager eine ordnungsgemäße Versionskontrolle erhalten.

Das REP enthält zwei unterschiedliche Erkenntnisse:

1. Im großen Maßstab erfordern Modularität und lose Kopplung mehr als Typentrennung.
2. Die Wiederverwendbarkeit von Komponenten (auch wenn die gesamte "Wiederverwendung" intern erfolgt) führt zu allgemeiner Wartbarkeit.

Risiko

Einfach gesagt ist ein Risiko die Möglichkeit, dass ein Problem auftritt. Ein Risiko beinhaltet *Ungewissheit* über die Auswirkungen, Folgen oder Implikationen einer Aktivität oder Entscheidung, meist mit einer negativen Konnotation in Bezug auf einen bestimmten Wert (wie Gesundheit, Geld oder Eigenschaften eines Systems wie Verfügbarkeit oder Sicherheit).

Um ein Risiko zu quantifizieren, wird die Eintrittswahrscheinlichkeit mit dem potenziellen Wert multipliziert, der normalerweise ein Verlust ist – andernfalls wäre das Risiko eine Chance, die angesichts der Ungewissheit bei einigen Risiken ein mögliches Ergebnis sein könnte.

RM/ODP

Reference Model for Open Distributed Processing

(Abstraktes) Metamodell zur Dokumentation von Informationssystemen. Definiert in ISO/IEC 10746.

Round-Trip-Engineering

"Konzept, gemäß dem an einem Modell sowie am aus diesem Modell generierten Code alle Arten von Änderungen vorgenommen werden können. Die Änderungen werden immer in beide Richtungen propagiert und beide Artefakte sind immer konsistent." (Übersetztes englisches Zitat aus Wikipedia).

Anmerkung (Gernot Starke)

Meiner persönlichen Meinung nach funktioniert dies in der Praxis nicht, sondern nur in "Hello-World"-ähnlichen Szenarien, da die umgekehrte Abstraktion (von Quellcode niedriger Ebene zu Architekturelementen höherer Ebene) in der Regel Entwurfsentscheidungen erfordert und realistischerweise nicht automatisiert werden kann.

Anmerkung (Matthias Bohlen)

Vor Kurzem habe ich aus DDD stammenden Code gesehen, bei dem Reverse Engineering tatsächlich funktioniert hat.

Ruby

Eine großartige Programmiersprache.

S

S.O.L.I.D.-Prinzipien

SOLID (Single-Responsibility, Open-Closed, Liskovsche Substitution, Interface-Segregation und Dependency-Inversion) ist ein (von Robert C. Martin) geprägtes Akronym für einige Prinzipien zur Verbesserung der objektorientierten Programmierung und des objektorientierten Entwurfs. Diese Prinzipien erhöhen die Wahrscheinlichkeit, dass ein Entwickler leicht zu wartenden und im Laufe der Zeit erweiterbaren Code schreibt.

Weitere Quellen: siehe [SOLID-principles].

Schicht

Zusammenstellung von Bausteinen oder Komponenten die (zusammen) anderen Schichten einen kohärenten Satz an Services bieten. Die Beziehung zwischen Schichten wird durch die geordnete Beziehung *erlaubt zu nutzen* geregelt. Zwei häufigen Arten von Schichten sind *Abstraktionschichten* zum Verstecken von Details (Beispiel: ISO/OSI-Netzwerkschichten oder „Hardware-Abstraktionsschicht“, siehe https://en.wikipedia.org/wiki/Hardware_abstraction) und Schichten, die (physikalisch) Funktionen oder Verantwortlichkeiten trennen (siehe https://de.wikipedia.org/wiki/Schichtenarchitektur).

Schnittstelle

Mehrere Bedeutungen, je nach Kontext:

1. Grenze, über die zwei Bausteine hinweg interagieren oder miteinander kommunizieren.
2. Entwurfskonstrukt, welches eine Abstraktion des Verhaltens konkreter Komponenten bereitstellt und mögliche Interaktionen sowie Einschränkungen für die Interaktionen mit diesen Komponenten deklariert.

Ein Beispiel für die zweite Bedeutung ist das Programmiersprachenkonstrukt Interface aus der objektorientierten Sprache Java(tm):

```
/* File name : Animal.java */
interface Animal {
   public void eat();
   public void move();
}

/* File name : Horse.java */
public class Horse implements Animal {

   public void eat() {
      System.out.println("Horse eats");
   }

   public void move() {
      System.out.println("Horse moves");
   }
}
```

Schnittstellenaufteilungsprinzip (Interface Secration Principle, ISP)

Bausteine (Klassen, Komponenten) sollen nicht gezwungen werden, von Methoden abzuhängen, die sie nicht nutzen. Nach dem ISP werden größere Schnittstellen in kleinere und (client)spezifischere Schnittstellen aufgeteilt, sodass Clients nur Methoden kennen müssen, die sie tatsächlich nutzen.

Schulungsanbieter

Eine Organisation oder Person, die über die Rechte zur Nutzung von akkreditierten Schulungsunterlagen verfügt oder die eine Akkreditierung für Schulungsunterlagen erworben hat, Schulungsleiter und Infrastruktur zur Verfügung stellt und Schulungen durchführt.

Schulungsleiter:in

Ein:e Trainer:in ist eine Person, die eine Schulung selbst leitet, mit der Maßgabe, dass diese im Rahmen einer einem akkreditierten Schulungsanbieter gewährten Akkreditierung durchgeführt wird. Entsprechend dürfen akkreditierte Schulungsanbieter nur CPSA®-Schulungen mit akkreditierten Schulungsleiter:innen organisieren und durchführen. Nur akkreditierte Schulungsanbieter können Akkreditierungen von Schulungsleiter:innen beantragen.

Schulungslevel

Das iSAQB® CPSA® Schulungsprogramm ist (derzeit) in zwei Schulungslevel gegliedert:

1. **Foundation Level** und
2. **Advanced Level**.

Die Schulungslevel sollten aufeinander aufbauendes Wissen enthalten. Die genauen Beziehungen untereinander und die Inhalte dieser Schulungslevel sind in den jeweiligen Lehrplänen festgelegt.

Schutzziele

Die Ziele sind der Hauptpunkt von Informationssicherheit. Sie sind ein Basissatz an Informationseigenschaften, die abhängig von der Architektur und den Prozessen eines Systems erfüllt werden können oder nicht.

Die gängigste Gruppe von vereinbarten Schutzzielen ist die sogenannte "CIA-Triade":

- Vertraulichkeit (Confidentiality)
- Integrität (Integrity)
- Verfügbarkeit (Availability)

Das "Reference Model of Information Assurance and Security" (RMIAS) erweitert diese Liste um Verantwortlichkeit, Prüfbarkeit, Authentifizierbarkeit/Vertrauenswürdigkeit, Nichtabstreitbarkeit und Datenschutz.

Dies sind die typischen Beispiele für nichtfunktionale Anforderungen in Zusammenhang mit Sicherheit.

Siehe [Anderson 2008] Seite 11 oder [Cherdantseva+2013].

SCS (Self Contained System)

Ein Architekturstil, ähnlich wie Microservices. Auf scs-architecture.org heißt es hierzu (übersetzt aus dem Englischen):

"Der Ansatz Self-contained System (SCS) ist eine Architektur, die sich auf eine Trennung der Funktionalität in zahlreiche unabhängige Systeme konzentriert, sodass das vollständige System eine Zusammenarbeit vieler kleinerer Softwaresysteme ist. Dies verhindert das Problem großer Monolithen, die stetig wachsen und irgendwann nicht mehr wartbar sind."

SDL

Ein **Secure-Development-Lifecycle** ist der übliche Softwareentwicklungsprozess eines Unternehmens mit zusätzlichen Praktiken zur Entwicklung von sicherer Software. Er umfasst beispielsweise Code-Reviews, Architektur-Risikoanalysen, Blackbox/Whitebox und Penetrationstests und zahlreiche weitere Ergänzungen. Der SDL sollte den gesamten Lebenszyklus einer Anwendung, von den ersten Anforderungsmanagementaufgaben bis zum Feedback aus dem Betrieb der herausgegebenen Software durch Behebung von Sicherheitsproblemen, abdecken.

Siehe [McGraw 2006], Seite 239.

Sensitivitätspunkt

(In der qualitativen Bewertung, wie ATAM): Element des Architektursystems, das mehrere Qualitätsmerkmale beeinflusst. Wenn beispielsweise eine Komponente *sowohl* für die Laufzeitleistung *als auch* die Robustheit verantwortlich ist, ist diese Komponente ein Sensitivitätspunkt.

Salopp gesagt, wenn man einen Sensitivitätspunkt in den Sand setzt, hat man zumeist mehr als ein Problem.

Separation of Concerns (SoC)

Jedes Element einer Architektur sollte über Exklusivität und Einzigartigkeit von Verantwortlichkeit und Zweck verfügen: Kein Element sollte die Verantwortlichkeiten eines anderen Elements teilen oder unverbundene Verantwortlichkeiten enthalten.

Eine weitere Definition lautet: Aufteilung eines Systems in Elemente, die sich möglichst wenig überschneiden.

Der berühmte Edgar Dijkstra sagte 1974: "Separation of concerns … ist, auch wenn es nicht perfekt möglich ist, die einzig verfügbare Technik zur effektiven Ordnung der eigenen Gedanken."

Ähnlich wie das Single-Responsibility-Prinzip.

Sequenzdiagramm

Diagrammart zur Illustration, wie Elemente einer Architektur interagieren, um ein bestimmtes Szenario zu erreichen. Es zeigt die Sequenz (Abfolge) von Mitteilungen zwischen Elementen. Die parallelen vertikalen Linien stellen die Lebensspanne von Objekten oder Komponenten dar, und die horizontalen Linien zeigen die Interaktionen zwischen diesen Komponenten. Siehe folgendes Beispiel.

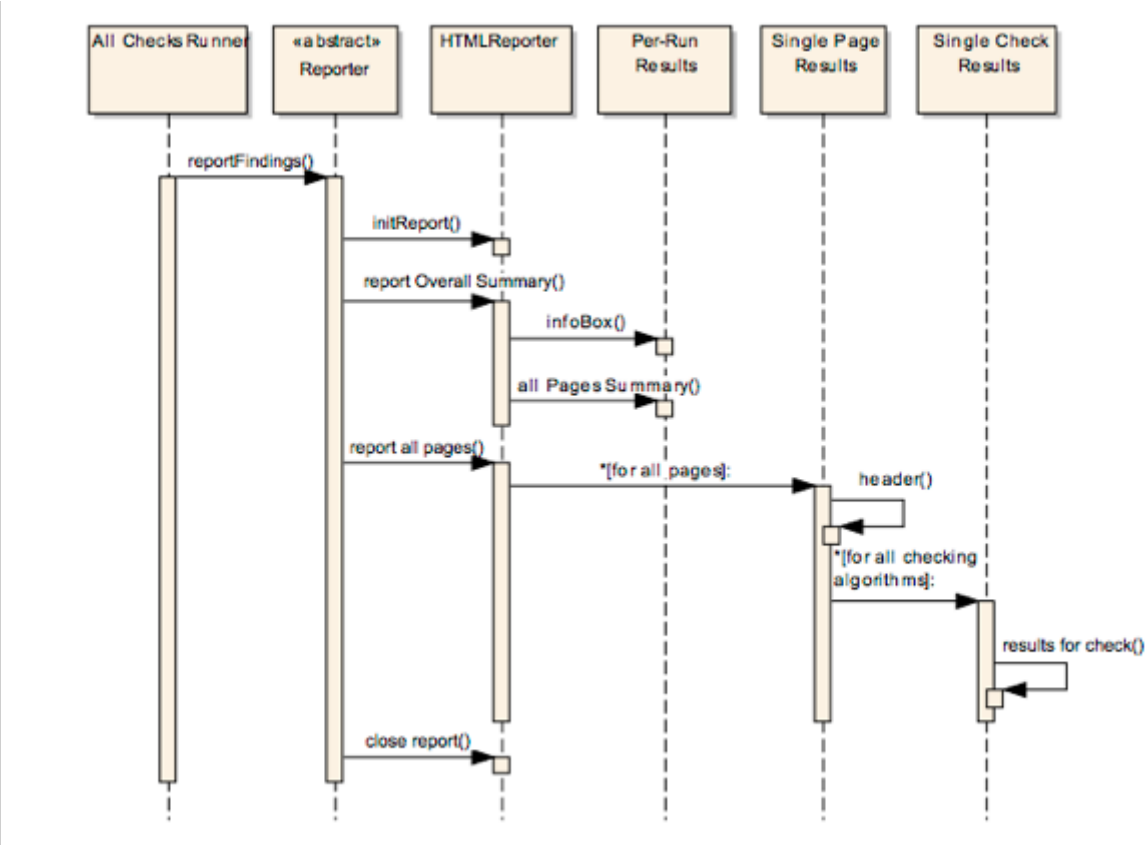

Figure 2. Beispiel eines Sequenzdiagramms

Service (DDD)

Ein Service ist ein Baustein des Domain-driven Design. Services implementieren eine Logik oder Prozesse der Geschäftsdomäne, die nicht von Entitäten alleine ausgeführt werden. Ein Service ist zustandslos, und die Parameter und Rückgabewerte seiner Operationen sind Entitäten, Aggregate und Wertobjekte.

Serviceorientierte Architektur

Architekturparadigma, das sich auf die Bereitstellung dokumentierter Schnittstellen mit austauschbaren Implementierungen konzentriert. Services ermöglichen Wiederverwendung über Organisationsgrenzen hinweg. Siehe Dienst.

Sicht

Siehe Architektursicht.

Signatur

Signatur einer Funktion oder Methode: Siehe Funktionssignatur.

Digitale Signatur: Methode zur Überprüfung der Authentizität von Daten oder Dokumenten.

Single-Responsibility-Prinzip (SRP)

Jedes Element in einem System oder einer Architektur sollte eine einzige Verantwortlichkeit haben, und alle seine Funktionen oder Dienste sollten auf diese Verantwortlichkeit abgestimmt sein.

Kohäsion wird manchmal als gleichbedeutend mit SRP angesehen.

Singleton

"Entwurfsmuster, das die Instanziierung einer Klasse auf ein Objekt beschränkt. Dies ist sinnvoll, wenn genau ein Objekt benötigt wird, um Aktionen im System zu koordinieren." (Übersetztes englisches Zitat aus Wikipedia.)

Softwarearchitektur

Es gibt mehrere(!) gültige und plausible Definitionen des Begriffs *Softwarearchitektur*. Die IEEE 1471 Norm schlägt folgende Definition vor:

> Softwarearchitektur: die grundlegende Organisation eines Systems, wie sie sich in dessen Komponenten, deren Beziehungen zueinander und zur Umgebung sowie den Grundsätzen für Entwurf, Entwicklung und Evolution widerspiegelt.

In der neuen Norm ISO/IEC/IEEE 42010:2011 wurden die Definitionen folgendermaßen übernommen und überarbeitet:

> Architecture: (system) fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution.

Die Schlüsselbegriffe dieser Definition bedürfen einer Erläuterung:

- Komponenten: Teilsysteme, Module, Klassen, Funktionen oder allgemeiner gesagt Bausteine: Strukturelemente von Software. Komponenten werden üblicherweise in einer Programmiersprache implementiert, können aber auch andere Artefakte sein, die (zusammen) *das System bilden*.
- Beziehungen: Schnittstellen, Abhängigkeiten, Assoziationen — verschiedene Bezeichnungen für dieselbe Funktion: Komponenten müssen mit anderen Komponenten interagieren, um Separation of Concerns zu ermöglichen.
- Umgebung: Jedes System hat Beziehungen zu seiner Umgebung. Daten, Kontrollflüsse oder Ereignisse werden an möglicherweise unterschiedliche Arten von Nachbarn und von diesen übertragen.
- Prinzipien: Regeln oder Konventionen, die für ein System oder mehrere Teile eines Systems gelten. Entscheidung oder Definition, die in der Regel für mehrere Elemente des Systems gültig ist. Häufig Konzepte oder sogar *Lösungsmuster* genannt. Prinzipien (Konzepte) bilden die Grundlage für

konzeptionelle Integrität.

Das *Software Engineering Institute* führt eine Sammlung weiterer Definitionen.

Der Begriff bezieht sich sowohl auf die *Softwarearchitektur eines IT-Systems*, wie auch benutzt, um sich auf *Softwarearchitektur als Ingenieursdisziplin* zu beziehen (also als Aufgabe oder Rolle in Entwicklungsprojekten oder -organisationen).

Softwarequalität

(Aus der IEEE-Norm 1061): Die Softwarequalität ist das Maß, in dem eine Software eine gewünschte Kombination von Merkmalen besitzt. Die gewünschte Kombination von Eigenschaften muss klar definiert sein; ansonsten bleibt die Beurteilung der Qualität der Intuition überlassen.

(Aus der ISO/IEC-Norm 25010): Die Qualität eines Systems ist das Maß, in dem das System die festgelegten und vorausgesetzten Anforderungen seiner verschiedenen Stakeholder erfüllt und somit Wert bietet. Diese festgelegten und vorausgesetzten Anforderungen sind in den ISO 25000 Qualitätsmodellen dargestellt, die Produktqualität in Eigenschaften, welche in manchen Fällen weiter in Untereigenschaften unterteilt werden, einteilt.

Sparsamkeit

Wirtschaftlich, einfach, schlank oder mit relativ geringem Aufwand machbar sein.

Stable-Abstractions-Prinzip

Ein Grundsatz für die Gestaltung der Struktur von Softwaresystemen (siehe auch Packaging-Prinzipien). Er fordert, dass die Abstraktheit von Komponenten proportional zu ihrer Stabilität ist. Das eng damit verbundene SDP erklärt auch den Begriff **Stabilität** in diesem Zusammenhang.

Wir wollen, dass Komponenten, die abstrakte Konzepte und Verantwortlichkeiten repräsentieren, wenig oder keine Änderungen benötigen, weil zahlreiche konzeptionell spezifischere (konkrete) Komponenten von ihnen abhängen. Und wir wollen, dass Komponenten, die nicht einfach geändert werden können oder sollten, mindestens so abstrakt sind, dass wir sie erweitern können. Dies steht mit dem OCP in Zusammenhang.

Das SAP kann wie ein Zirkelargument klingen, bis die zugrunde liegende Idee zu Tage tritt: **Konkrete** Dinge und Konzepte sind natürlich volatiler, spezifischer, willkürlicher und zahlreicher als **abstrakte**. Die Komponentenstruktur eines Systems sollte dies einfach widerspiegeln. Die allgemeine Logik, die physischen Artefakte des Systems sowie seine funktionalen und technischen Konzepte sollten alle Deckungsgleich sein.

Das SAP ist eng mit dem SDP verbunden. Ihre Kombination ergibt eine allgemeinere und wohl tiefergehende Version des DIP: Spezifische Konzepte hängen natürlich von **abstrakteren** ab, da sie aus universaleren Bausteinen bestehen oder davon abgeleitet sind. Und abhängige Konzepte sind natürlich **spezifischer**, weil sie durch mehr Informationen als ihre Abhängigkeiten definiert sind (vorausgesetzt es gibt keine Abhängigkeitszyklen).

Stable-Dependencies-Prinzip

Ein Grundsatz für die Gestaltung der Struktur von Softwaresystemen (siehe auch Packaging-Prinzipien). Er fordert, dass sich häufig ändernde Komponenten von stabileren abhängen.

Ein Teil der Volatilität einer Komponente wird erwartet und von ihrer speziellen Verantwortlichkeit

logischerweise impliziert.

Aber in diesem Kontext hängt Stabilität auch von ein- und ausgehenden Abhängigkeiten ab. Eine Komponente, von der andere stark abhängen, ist schwieriger zu ändern und gilt als stabiler. Eine Komponente, die stark von anderen abhängt, hat mehr Änderungsgründe und gilt als weniger stabil.

In Bezug auf Abhängigkeit sollte also eine Komponente mit vielen Clients nicht von einer Komponente mit vielen Abhängigkeiten abhängen. Eine einzelne Komponente, die diese beiden Eigenschaften auf sich vereint, ist ebenfalls eine Red Flag. Eine solche Komponente hat viele Gründe für eine Änderung, ist aber gleichzeitig schwer zu ändern.

Ursprüngliche Definitionen des SDP (wie [Martin 2003]) beinhalten eine Kennzahl I der Instabilität. Leider erfasst diese Kennzahl beabsichtigte/inhärente Volatilität, transitive Abhängigkeit oder Fälle, wie die oben genannte Red Flag, nicht. Aber wir wissen das Konzept des SDP zu schätzen, unabhängig davon, wie es sich messen lässt.

Das SDP ist eng mit dem SAP verbunden. Ihre Kombination ergibt eine allgemeinere Version des DIP (mehr dazu unter SAP).

Stakeholder

Person oder Organisation, die von einem System, seiner Entwicklung oder Ausführung betroffen sein kann oder ein Interesse (*stake*) daran hat.

Beispiele sind Benutzer, Beschäftigte, Eigner, Administratoren, Entwickler, Entwerfer, Manager, Product Owner, Projektmanager.

Gemäß ISO/IEC/IEEE 42010 sind Stakeholder (System) eine Einzelperson, ein Team, eine Organisation oder Klassen davon, die ein Interesse an einem System haben (gemäß Definition in ISO/IEC/IEEE 42010).

Stellvertreter / Proxy

(Entwurfsmuster) "Ein Wrapper oder Stellvertreterobjekt, das vom Client aufgerufen wird, um auf das reale Serving-Objekt im Hintergrund zuzugreifen. Die Funktion des Stellvertreters kann einfach in der Weiterleitung an das reale Objekt oder die Bereitstellung zusätzlicher Logik sein. Im Stellvertreter kann eine zusätzliche Funktionalität bereitgestellt werden, beispielsweise Caching, wenn die Operationen des realen Objekts ressourcenintensiv sind, oder Überprüfung von Voraussetzungen vor dem Aufruf von Operationen des realen Objekts. Für den Client ist die Verwendung eines Stellvertreterobjekts mit der Verwendung des realen Objekts vergleichbar, da beide die gleiche Schnittstelle implementieren." (Übersetztes englisches Zitat aus Wikipedia)

Strategy

Entwurfsmuster für die Definition einer Familie von Algorithmen, wobei jeder einzelne Algorithmus gekapselt und austauschbar ist. Mit Strategy kann ein konkreter Algorithmus unabhängig vom Nutzer geändert werden. (Zitiert aus [Gang-of-Four, short: GoF].)

Struktur

Anordnung, Ordnung oder Organisation von zusammenhängenden Elementen in einem System. Strukturen bestehen aus Bausteinen (Strukturelementen) und ihren Beziehungen (Abhängigkeiten).

In der Softwarearchitektur werden Strukturen häufig in Architektursichten, z.B. der Bausteinsicht, verwendet. Ein Dokumentationstemplate (z.B. arc42) ist auch eine Art Struktur.

Strukturelement

Siehe Baustein oder Komponente

Summe

Summendaten sind solche, bei denen ein Wert zu einem von mehreren verschiedenen aber festehenden Typen gehört. Auch als **direkte Summe** oder **gemischte Daten** oder **Oder_Daten** bezeichnet.

Beispiel:

Eine **geometrische Figur** ist eine der folgenden:

- ein Quadrat oder
- ein Kreis oder
- ein Dreieck

Siehe auch [Sperber+2024].

Symmetrische Kryptografie

Symmetrische Kryptografie basiert auf einem identischen Schlüssel für die Verschlüsselung und Entschlüsselung von Daten. Sender und Empfänger vereinbaren einen Schlüssel für die Kommunikation. Siehe [Schneier 1996], Seite 17.

System

Sammlung von Elementen (Bausteinen, Komponenten usw.), die zu einem gemeinsamen Zweck organisiert sind. In den ISO/IEC/IEEE-Normen gibt es eine Reihe von Systemdefinitionen:

- Systeme gemäß Beschreibung in [ISO/IEC 15288]: Systeme, die vom Menschen geschaffen wurden und mit einem oder mehreren der folgenden Aspekte konfiguriert werden können: Hardware, Software, Daten, Menschen, Prozesse (z.B. Prozesse zur Bereitstellung eines Dienstes für Benutzer), Verfahren (z.B. Bedieneranweisungen), Anlagen, Material und natürlich vorkommende Entitäten.
- Softwareprodukte und Dienste gemäß Beschreibung in [ISO/IEC 12207].
- Software-intensive Systeme gemäß Beschreibung in [IEEE Std 1471:2000]: jegliche Systeme, in denen Software wesentliche Einflüsse zum Entwurf, zur Entwicklung, Verbreitung und Weiterentwicklung des Systems als Ganzes beisteuert, um individuelle Anwendungen, Systeme im herkömmlichen Sinne, Teilsysteme, Systemverbünde, Produktlinien, Produktfamilien, ganze Unternehmen und sonstige Interessensvereinigungen zu umspannen.

Szenario

Qualitätsszenarien dokumentieren die vorgegebenen Qualitätsmerkmale. Sie helfen bei der Beschreibung der vorgegebenen oder gewünschten Eigenschaften eines Systems auf pragmatische und informelle Weise und machen dennoch das abstrakte Konzept "Qualität" konkret und greifbar.

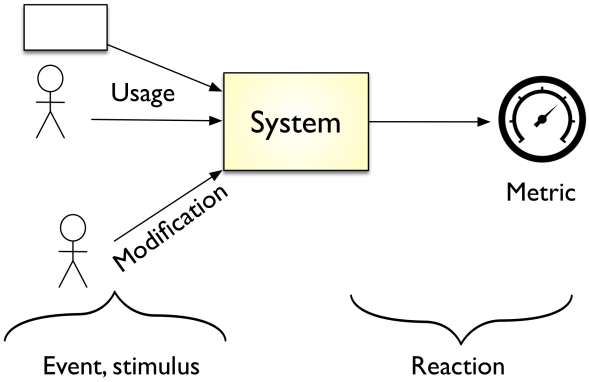

Figure 3. Allgemeine Form eines (Qualitäts-)Szenarios

- Ereignis/Stimulus: Jegliche Bedingungen oder Ereignisse, die das System erreichen
- System (oder ein Teil des Systems) wird durch Ereignis stimuliert.
- Antwort: Nach Eintreffen des Stimulus durchgeführte Aktivität.
- Kennzahl (Antwortmaß): Die Antwort sollte auf irgendeine Weise gemessen werden.

Üblicherweise werden Qualitätsszenarien in drei Kategorien unterteilt

- Verwendungsszenarien (Anwendungsszenarien)
- Änderungsszenarien (Szenarien der Veränderung oder des Wachstums)
- Fehlerszenarien (Stressszenarien oder explorative Szenarien)

T

Technischer Kontext

Zeigt das vollständige System als eine Blackbox innerhalb seiner Umgebung aus einer technischen bzw. Verteilungsperspektive. Dazu gehören insbesondere technische Schnittstellen und Kommunikationskanäle sowie die relevanten technischen Details der Nachbarsysteme. Damit ergänzt der technische Kontext den fachlichen Kontext um die Zuordnung der fachlichen Interaktionen mit Nachbarsystemen zu z. B. spezifischen Kommunikationskanälen und technischen Protokollen.

Siehe Kontextabgrenzung.

Template (zur Dokumentation)

Standardisierte Zusammenstellung von Artefakten, die in der Softwareentwicklung verwendet werden. Templates können dabei helfen, andere Dateien, insbesondere Dokumente, in eine vordefinierte Struktur einzubetten, ohne den Inhalt dieser einzelnen Dateien vorzugeben.

Ein sehr bekanntes Template ist arc42

Template Method

Entwurfsmuster, das das Skelett eines Algorithmus in einer Operation definiert, wobei einige Schritte in Unterklassen verschoben werden. Mit Template Method können Unterklassen bestimmte Schritte eines Algorithmus neu definieren, ohne die Struktur des Algorithmus zu verändern. (Zitiert aus [Gang-of-Four, short: GoF].)

TLS

Transport-**L**ayer-**S**ecurity (Transportschichtsicherheit) bezeichnet eine Reihe von Protokollen zum kryptografischen Schutz der Kommunikation von zwei Parteien mit den Mitteln der CIA-Triade. Es wird sehr häufig zur sicheren Kommunikation im Internet genutzt und bildet die Grundlage für HTTPS.

TLS begann als Update seines Vorgängers SSL (Secure Socket Layer) Version 3.0 und sollte nun statt SSL verwendet werden siehe RFC7568 "Deprecating Secure Sockets Layer Version 3.0".

TOGAF

The Open Group Architecture Framework

Konzeptioneller Rahmen für die Planung und Wartung von Unternehmens-IT-Architekturen.

Top-Down

"Arbeitsrichtung" oder "Kommunikationsabfolge": Ausgehend von einem abstrakten oder allgemeinen Konstrukt hin zu einer konkreteren, spezielleren oder detaillierteren Darstellung.

U

Umgebung

(System) Kontext, der das Setting und die Umstände aller Einflüsse auf ein System bestimmt (gemäß Definition in ISO/IEC/IEEE 42010).

Die Umgebung eines Systems schließt Entwicklungs-, Geschäfts- und Betriebseinflüsse sowie technologische, organisatorische, politische, wirtschaftliche, rechtliche, regulatorische, ökologische und soziale Einflüsse ein.

Unified Modeling Language (UML)

Unified Modeling Language / Vereinheitlichte Modellierungssprache (UML)

Grafische Sprache zur Visualisierung, Spezifizierung und Dokumentation der Artefakte und Strukturen eines Softwaresystems.

- Verwenden Sie für Bausteinsichten oder die Kontextabgrenzung Komponentendiagramme mit Komponenten, Paketen oder Klassen zur Bezeichnung von Bausteinen.
- Für Laufzeitsichten verwenden Sie Sequenz- oder Aktivitätsdiagramme (mit Schwimmbahnen). Objektdiagramme können theoretisch verwendet werden, sind aber praktisch nicht zu empfehlen, da sie auch bei kleinen Szenarien überhäuft erscheinen.
- Verwenden Sie für Verteilungssichten Verteilungsdiagramme mit Knotensymbolen.

Unit Test

Test der kleinsten testbaren Teile eines Systems, um festzustellen, ob sie einsatzfähig sind.

Je nach Implementierungstechnologie kann eine *Unit* eine Methode, Funktion, Schnittstelle oder ein ähnliches Element sein.

Unternehmens-IT-Architektur

Synonym: Unternehmensarchitektur.

Strukturen und Konzepte für den IT-Support eines gesamten Unternehmens. Die kleinsten betrachteten Einheiten der Unternehmensarchitektur sind einzelne Softwaresysteme, auch "Anwendungen" genannt.

V

Verbergen von Informationen

Ein grundlegendes Prinzip im Softwareentwurf: Entwurfs- oder Implementierungsentscheidungen, die sich wahrscheinlich ändern, werden **verborgen** gehalten, so dass andere Teile des Systems vor Modifizierungen geschützt sind, wenn diese Entscheidungen oder Implementierungen geändert werden. Eine der wichtigen Eigenschaften von Blackboxen. Trennt Schnittstelle von Implementierung.

Der Begriff Kapselung wird häufig austauschbar mit Verbergen von Informationen verwendet.

Verfolgbarkeit

(Genauer gesagt: **Anforderungs**verfolgbarkeit): Dokumentation, dass

1. alle Anforderungen durch Elemente des Systems abgedeckt sind (Vorwärtsverfolgbarkeit) und
2. alle Elemente des Systems durch mindestens eine Anforderung begründet sind (Rückverfolgbarkeit).

Meine persönliche Meinung: Verfolgbarkeit sollte nach Möglichkeit vermieden werden, da sie einen erheblichen Dokumentationsaufwand verursacht.

Verfügbarkeit

Eines der grundlegenden Schutzziele, das ein System beschreibt, das die gewünschten Informationen bei Bedarf bereitstellen kann. Aus einer Sicherheitsperspektive können beispielsweise Denial-of-Service-Angriffe die Verfügbarkeit verhindern.

Verteilung

Einbringen der Software in ihre Ausführungsumgebung (Hardware, Prozessor usw.). Inbetriebnahme der Software.

Verteilungssicht

Architektursicht, die die technische Infrastruktur, in der ein System oder Artefakte verteilt und ausgeführt werden, zeigt.

"Diese Sicht definiert die physische Umgebung, in der das System laufen soll, einschließlich der Hardwareumgebung, die Ihr System benötigt (z.B. Verarbeitungsknoten, Netzwerkverbindungen und Speicherkapazitäten), der technischen Umgebungsanforderungen für jeden Knoten (oder Knotentyp) im System und des Mappings Ihrer Softwareelemente in Bezug auf die Laufzeitumgebung, die sie ausführt." (Übersetztes englisches Zitat von Rozanski+2011)

Vertraulichkeit

Eines der grundlegenden Schutzziele, das ein System beschreibt, welches Informationen nur Befugten offenlegt und bereitstellt.

Visitor

Entwurfsmuster, um Operationen zu modellieren, die auf den Elementen einer Objektstruktur durchgeführt werden. Visitor erlaubt die Definitione neuer Operation, ohne bestehende Klassen zu ändern. (Nach [Gang-

of-Four, short: GoF].)

W

Wasserfall-Entwicklung

Entwicklungsansatz, "bei dem man alle Anforderungen vorab zusammenträgt, den gesamten erforderlichen Entwurf bis runter auf Detailebene macht und dann die Spezifikationen an die Coder, die den Code schreiben, weitergibt; dann werden Tests durchgeführt (eventuell mit einem Abstecher in die Integrationshölle) und schließlich wird das Ganze mit einem großen abschließenden Release geliefert. Alles ist groß, auch die Gefahr des Scheiterns." (Übersetztes englisches Zitat aus C2 wiki.

Siehe auch iterativen Entwicklung

Wertobjekt

Ein Wertobjekt ist ein Baustein des Domain-driven Designs. Wertobjekte haben keine eigene konzeptionelle Identität und sollten als unveränderlich behandelt werden. Sie werden zur Beschreibung des Zustands von Entitäten genutzt und können aus anderen Wertobjekten, aber niemals aus Entitäten bestehen.

Whitebox

Zeigt die interne Struktur eines aus Blackboxes bestehenden Systems oder Bausteins und die internen/externen Beziehungen/Schnittstellen.

Siehe auch Blackbox.

Workflow-Management-System (WFMS)

"Bietet eine Infrastruktur für die Einrichtung, Durchführung und Überwachung einer festgelegten Abfolge von Aufgaben in Form eines Workflows." (Übersetztes englisches Zitat aus Wikipedia)

Wrapper

(Syn.: Decorator, Adapter, Gateway) Muster zum abstrahieren der konkreten Schnittstelle oder Implementierung oder Komponente. Wrapper fügen zusätzliche Verantwortlichkeiten dynamisch zu einem Objekt hinzu.

Anmerkung (Gernot Starke)

Die winzigen Unterschiede, die sich in der Literatur zu diesem Begriff finden, spielen im realen Leben häufig keine Rolle. Das *Wrapping* einer Komponente oder eines Bausteins muss in einem einzelnen Softwaresystem eine klare Bedeutung haben.

Z

Zeitliche Kopplung

Es gibt unterschiedliche Interpretationen aus verschiedenen Quellen. Zeitliche Kopplung

- bedeutet, dass Prozesse, die miteinander kommunizieren, beide aktiv sein müssen. Siehe [Tanenbaum+2016].
- wenn du oft verschiedene Komponenten gleichzeitig eincheckst (*modifizierst*). Siehe [Tornhill 2015].
- wenn es eine implizite Beziehung zwischen zwei oder mehr Mitgliedern einer Klasse gibt, die es

erforderlich macht, dass die Clients das eine Mitglied vor dem anderen aufrufen. Mark Seemann, siehe Design Smell Temporal Coupling

- bedeutet, dass ein System auf die Antwort eines anderen Systems warten muss, bevor es mit der Bearbeitung fortfahren kann. Siehe Rest Antipattern

Zerlegung

(Syn.: Faktorisieren) Aufbrechen oder Unterteilen eines komplexen Systems oder Problems in mehrere kleinere Teile, die einfacher zu verstehen, zu implementieren oder zu warten sind.

Zertifizierungsprogramm

Das iSAQB® CPSA®-Zertifizierungsprogramm, einschließlich seiner organisatorischen Komponenten, Dokumente (Schulungsunterlagen, Verträge) und Prozesse.

Die urheberrechtliche geschützte Abkürzung CPSA® steht für **Certified Professional for Software Architecture**.

Zufälligkeit

Siehe Entropie oder Pseudo-Zufälligkeit.

Zyklomatische Komplexität

Quantitatives Maß, Zahl der unabhängigen Pfade durch den Quellcode eines Programms. Sie entspricht grob der Zahl der bedingten Anweisungen (if, while) im Code +1. Eine lineare Abfolge von Anweisungen ohne if oder while hat eine zyklomatische Komplexität von 1. Viele Softwareentwickler sind der Auffassung, dass eine höhere Komplexität mit der Anzahl der Fehler zusammenhängt.

Übersetzungstabellen

TODO

English	German
Accessibility	Barrierefreiheit, Zugänglichkeit
Accountability	Rechenschaft, Verantwortlichkeit
Accreditation contract	Akkreditierungsvertrag
Accreditation fee	Akkreditierungsgebühr
Action	Maßnahme
Adaptability	Adaptierbarkeit
Adaption	Anpassung
Adequacy	Angemessenheit
Analysability	Analysierbarkeit
Approach	Ansatz
Appropriateness	Angemessenheit
Appropriateness Recognizability	Erkennbarkeit der Brauchbarkeit, Verständlichkeit
Architectural objective	Architekturziel
Architectural pattern	Architekturmuster
Architectural view	Architektursicht, Sicht
Architecture assessment	Architekturanalyse, Architekturbewertung
Architecture evaluation	Architekturbewertung, Architekturanalyse
Architecture objective	Architekturziel
Articles of assocation	Satzung des Vereins
Artifact	Artefakt
Aspect	Aspekt, Belang
Assessment	Bewertung, Begutachtung, Einschätzung, Untersuchung
Association	Verein, Beziehung
Attack Tree	Angriffsbäume
Authenticity	Authentifizierbarkeit
Availability	Verfügbarkeit
Bounded Context	Kontextgrenze
Building block	Baustein
Building block view	Bausteinsicht
Business	Fachlichkeit, Domäne
Business architecture	fachliche Architektur, Geschäftsarchitektur
Business context	Fachlicher Kontext
Cabinet (as methaphor for template)	Schrank (als Metapher für Template)
Capacity	Kapazität

Cash audit	Rechnungsprüfung
Cash auditor	Rechnungsprüfer
Certification authority	Zertifizierungsstelle
Certification body	Zertifizierungsstelle
Chairman	Vorsitzender
Channel	Kanal
Co-Existence	Koexistenz
Cohesion	Kohäsion, innerer Zusammenhalt
Commensurability	Angemessenheit, Messbarkeit, Vergleichbarkeit
Compatibility	Kompatibilität
Compliance	Erfüllung, Einhaltung
Component	Baustein, Komponente
Concern	Belang
Confidentiality	Vertraulichkeit
Constraint	Randbedingung, Einschränkung
Context (of a term)	Einordnung (eines Begriffes) in einen Zusammenhang
Context view	Kontextabgrenzung
Coupling	Kopplung, Abhängigkeit
Cross-cutting	Querschnittlich
Curriculum	Lehrplan
Decomposition	Zerlegung
Dependency	Abhängigkeit, Beziehung
Deployment	Verteilung
Deployment unit	Verteilungsartefakt
Deployment view	Verteilungssicht
Deputy chairman	Stellvertretender Vorsitzender
Design	Entwurf
Design approach	Entwurfsansatz, Entwurfsmethodik
Design decision	Entwurfsentscheidung
Design principle	Entwurfsprinzip
Domain	Fachdomäne, Fachlicher Bereich, Geschäftsbereich
Domain event	Fachliches Event
Domain-related architecture	fachliche Architektur
Drawing Tool	Mal-/Zeichenprogramm
Economicalness	Sparsamkeit, Wirtschaftlichkeit
Embedded	Eingebettet
Encapsulation	Kapselung

Enterprise IT architecture	Unternehmens-IT-Architektur
Estimation	Schätzung
Evaluation	Bewertung
Examination question	Prüfungsfrage
Examination rules and regulations	Prüfungsordnung
Examination sheet	Prüfungsbogen
Examination task	Prüfungsaufgabe
Examinee	Prüfling
Examiner	Prüfer
Executive board	Vorstand
Fault Tolerance	Fehlertoleranz
Fees rules and regulations	Gebührenordnung
Fitness Function	Fitnessfunktion
Functional Appropriateness	Funktionale Angemessenheit
Functional Completeness	Funktionale Vollständigkeit
Functional Correctness	Funktionale Korrektheit
Functional Suitability	Funktionale Eignung
General meeting	Mitgliederversammlung
Improvement	Verbesserung
Improvement action	Verbesserungsmaßnahme
Influencing Factor	Einflussfaktor
Information hiding principle	Geheimnisprinzip
Installability	Installierbarkeit
Integrity	Integrität
Interdependency (between design decisions)	Abhängigkeit (zwischen Entwurfsentscheidungen)
Interface	Schnittstelle
Interface description	Schnittstellenbeschreibung, Schnittstellendokumentation
Interoperability	Interoperabilität
Learnability	Erlernbarkeit
Learning goal	Lernziel
License fee	Akkreditierungsgebühr
Licensee	Lizenznehmer
Licensing agreement	Lizenzvertrag, Lizenzvereinbarung, Akkreditierungsvertrag
Local court	Amtsgericht
Maintainability	Wartbarkeit
Maturity	Reifegrad
Means for describing	Beschreibungsmittel

Means for documenting	Beschreibungsmittel
Measurability	Messbarkeit
Members' meeting	Mitgliederversammlung
message-driven	Nachrichten-zentrisch
Modeling Tool	Modellierungswerkzeug
Modifiability	Modifizierbarkeit
Modularity	Modularität
Module	Komponente, Modul, Baustein
Node	Knoten
Non-exclusive license	Einfache Lizenz
Non-profit	Gemeinnützig
Non-repudiation	Nichtabstreitbarkeit
Normal case	Normalfall
Notification	Benachrichtigung
Objective	Ziel
Operability	Bedienbarkeit
Operational processes	Betriebsprozesse (von Software)
Pattern	Muster
Pattern language	Mustersprache, Musterfamilie
Performance Efficiency	Leistungseffizienz, Performance
Perspective	Perspektive
Portability	Portierbarkeit
Principle	Prinzip, Konzept
Quality attribute	Qualitätsmerkmal, Qualitätseigenschaft
Quality characteristic	Qualitätsmerkmal, Qualitätseigenschaft
Quality feature	Qualitätsmerkmal, Qualitätseigenschaft
Rationale	Begründung, Erklärung
Real-time system	Echtzeitsystem
Recoverability	Widerherstellbarkeit
Registered trademark	Marke (gesetzlich geschützt)
Relationship	Beziehung
Relationship (kind of)	Beziehungsart
Reliability	Zuverlässigkeit
Replaceability	Austauschbarkeit
Repository	Ablage
Requirement	Anforderung
resilient	unverwüstlich, selbstwiederherstellend
Resolution	Beschluss
Resource Utilization	Ressourcenverbrauch

Responsibility	Verantwortlichkeit
responsive	reaktionsfähig
Reusability	Wiederverwendbarkeit
Rights of use	Nutzungsrecht
Runtime	Laufzeit
Runtime view	Laufzeitsicht
Security	Sicherheit
Security Goals	Schutzziele, Sachziele
Skill	Fähigkeit, Fertigkeit
Specification (of software architecture)	Beschreibung (von Softwarearchitektur)
sponsoring (board) member	materiell förderndes Mitglied
statutory	satzungsgemäß
Structure	Struktur
Task	Aufgabe
Team regulations	Arbeitsgruppenordnung
Technical context	Technischer Kontext
Term	Begriff
Testability	Testbarkeit
Thriftyness	Sparsamkeit, Wirtschaftlichkeit
Time Behaviour	Zeitverhalten
Tools	Arbeitsmittel, Werkzeug
Tools-and-material-approach	Werkzeug-Material-Ansatz
Tradeoff	Kompromiss, Abwägung, Wechselwirkung
Training provider	Schulungsanbieter
Treasurer	Schatzmeister
Ubiquitous language	Allgegenwärtige Sprache
Usability	Benutzbarkeit, Benutzerfreundlichkeit
User Error Protection	Schutz vor Fehlbedienung
User Interface Aesthetics	Ästhetik der Benutzeroberfläche
Uses relationship	Benutzt-Beziehung, Nutzungsbeziehung
View	Sicht, Architektursicht
Workflow management	Ablaufsteuerung
Working environment	Arbeitsumgebung
Working group	Arbeitsgruppe
Working group head	Arbeitsgruppenleiter

TODO

German	English
Abhängigkeit	Coupling, Dependency
Abhängigkeit (zwischen Entwurfsentscheidungen)	Interdependency (between design decisions)
Ablage	Repository
Ablaufsteuerung	Workflow management
Abwägung	Tradeoff
Adaptierbarkeit	Adaptability
Akkreditierungsgebühr	Accreditation fee, License fee
Akkreditierungsvertrag	Accreditation contract, Licensing agreement
Allgegenwärtige Sprache	Ubiquitous language
Amtsgericht	Local court
Analysierbarkeit	Analysability
Anforderung	Requirement
Angemessenheit	Adequacy, Appropriateness, Commensurability
Angriffsbäume	Attack Tree
Anpassung	Adaption
Ansatz	Approach
Arbeitsgruppe	Working group
Arbeitsgruppenleiter	Working group head
Arbeitsgruppenordnung	Team regulations
Arbeitsmittel	Tools
Arbeitsumgebung	Working environment
Architekturanalyse	Architecture assessment, Architecture evaluation
Architekturbewertung	Architecture assessment, Architecture evaluation
Architekturmuster	Architectural pattern
Architektursicht	Architectural view, View
Architekturziel	Architectural objective, Architecture objective
Artefakt	Artifact
Aspekt	Aspect
Aufgabe	Task
Austauschbarkeit	Replaceability
Authentifizierbarkeit	Authenticity
Barrierefreiheit	Accessibility
Baustein	Building block, Component, Module
Bausteinsicht	Building block view
Bedienbarkeit	Operability
Begriff	Term

Begründung	Rationale
Begutachtung	Assessment
Belang	Aspect, Concern
Benachrichtigung	Notification
Benutzbarkeit	Usability
Benutzerfreundlichkeit	Usability
Benutzt-Beziehung	Uses relationship
Beschluss	Resolution
Beschreibung (von Softwarearchitektur)	Specification (of software architecture)
Beschreibungsmittel	Means for describing, Means for documenting
Betriebsprozesse (von Software)	Operational processes
Bewertung	Assessment, Evaluation
Beziehung	Association, Dependency, Relationship
Beziehungsart	Relationship (kind of)
Domäne	Business
Echtzeitsystem	Real-time system
Einfache Lizenz	Non-exclusive license
Einflussfaktor	Influencing Factor
Eingebettet	Embedded
Einhaltung	Compliance
Einordnung (eines Begriffes) in einen Zusammenhang	Context (of a term)
Einschränkung	Constraint
Einschätzung	Assessment
Entwurf	Design
Entwurfsansatz	Design approach
Entwurfsentscheidung	Design decision
Entwurfsmethodik	Design approach
Entwurfsprinzip	Design principle
Erfüllung	Compliance
Erkennbarkeit der Brauchbarkeit	Appropriateness Recognizability
Erklärung	Rationale
Erlernbarkeit	Learnability
Fachdomäne	Domain
fachliche Architektur	Business architecture, Domain-related architecture
Fachlicher Bereich	Domain
Fachlicher Kontext	Business context
Fachliches Event	Domain event
Fachlichkeit	Business

Fehlertoleranz	Fault Tolerance
Fertigkeit	Skill
Fitnessfunktion	Fitness Function
Funktionale Angemessenheit	Functional Appropriateness
Funktionale Eignung	Functional Suitability
Funktionale Korrektheit	Functional Correctness
Funktionale Vollständigkeit	Functional Completeness
Fähigkeit	Skill
Gebührenordnung	Fees rules and regulations
Geheimnisprinzip	Information hiding principle
Gemeinnützig	Non-profit
Geschäftsarchitektur	Business architecture
Geschäftsbereich	Domain
innerer Zusammenhalt	Cohesion
Installierbarkeit	Installability
Integrität	Integrity
Interoperabilität	Interoperability
Kanal	Channel
Kapazität	Capacity
Kapselung	Encapsulation
Knoten	Node
Koexistenz	Co-Existence
Kohäsion	Cohesion
Kompatibilität	Compatibility
Komponente	Component, Module
Kompromiss	Tradeoff
Kontextabgrenzung	Context view
Kontextgrenze	Bounded Context
Konzept	Principle
Kopplung	Coupling
Laufzeit	Runtime
Laufzeitsicht	Runtime view
Lehrplan	Curriculum
Leistungseffizienz	Performance Efficiency
Lernziel	Learning goal
Lizenznehmer	Licensee
Lizenzvereinbarung	Licensing agreement
Lizenzvertrag	Licensing agreement
Mal-/Zeichenprogramm	Drawing Tool

Marke (gesetzlich geschützt)	Registered trademark
materiell förderndes Mitglied	sponsoring (board) member
Maßnahme	Action
Messbarkeit	Commensurability, Measurability
Mitgliederversammlung	General meeting, Members' meeting
Modellierungswerkzeug	Modeling Tool
Modifizierbarkeit	Modifiability
Modul	Module
Modularität	Modularity
Muster	Pattern
Musterfamilie	Pattern language
Mustersprache	Pattern language
Nachrichten-zentrisch	message-driven
Nichtabstreitbarkeit	Non-repudiation
Normalfall	Normal case
Nutzungsbeziehung	Uses relationship
Nutzungsrecht	Rights of use
Performance	Performance Efficiency
Perspektive	Perspective
Portierbarkeit	Portability
Prinzip	Principle
Prüfer	Examiner
Prüfling	Examinee
Prüfungsaufgabe	Examination task
Prüfungsbogen	Examination sheet
Prüfungsfrage	Examination question
Prüfungsordnung	Examination rules and regulations
Qualitätseigenschaft	Quality attribute, Quality characteristic, Quality feature
Qualitätsmerkmal	Quality attribute, Quality characteristic, Quality feature
Querschnittlich	Cross-cutting
Randbedingung	Constraint
reaktionsfähig	responsive
Rechenschaft	Accountability
Rechnungsprüfer	Cash auditor
Rechnungsprüfung	Cash audit
Reifegrad	Maturity
Ressourcenverbrauch	Resource Utilization

Sachziele	Security Goals
Satzung des Vereins	Articles of assocation
satzungsgemäß	statutory
Schatzmeister	Treasurer
Schnittstelle	Interface
Schnittstellenbeschreibung	Interface description
Schnittstellendokumentation	Interface description
Schrank (als Metapher für Template)	Cabinet (as methaphor for template)
Schulungsanbieter	Training provider
Schutz vor Fehlbedienung	User Error Protection
Schutzziele	Security Goals
Schätzung	Estimation
selbstwiederherstellend	resilient
Sicherheit	Security
Sicht	Architectural view, View
Sparsamkeit	Economicalness, Thriftyness
Stellvertretender Vorsitzender	Deputy chairman
Struktur	Structure
Technischer Kontext	Technical context
Testbarkeit	Testability
Unternehmens-IT-Architektur	Enterprise IT architecture
Untersuchung	Assessment
unverwüstlich	resilient
Verantwortlichkeit	Accountability, Responsibility
Verbesserung	Improvement
Verbesserungsmaßnahme	Improvement action
Verein	Association
Verfügbarkeit	Availability
Vergleichbarkeit	Commensurability
Verständlichkeit	Appropriateness Recognizability
Verteilung	Deployment
Verteilungsartefakt	Deployment unit
Verteilungssicht	Deployment view
Vertraulichkeit	Confidentiality
Vorsitzender	Chairman
Vorstand	Executive board
Wartbarkeit	Maintainability
Wechselwirkung	Tradeoff
Werkzeug	Tools

Werkzeug-Material-Ansatz	Tools-and-material-approach
Widerherstellbarkeit	Recoverability
Wiederverwendbarkeit	Reusability
Wirtschaftlichkeit	Economicalness, Thriftyness
Zeitverhalten	Time Behaviour
Zerlegung	Decomposition
Zertifizierungsstelle	Certification authority, Certification body
Ziel	Objective
Zugänglichkeit	Accessibility
Zuverlässigkeit	Reliability
Ästhetik der Benutzeroberfläche	User Interface Aesthetics

Referenzen und Quellen

Dieser Abschnitt enthält Quellenangaben, die ganz oder teilweise im Glossar oder einem Curriculum referenziert werden.

A

- [Anderson 2008] Ross Anderson, Security Engineering - A Guide to Building Dependable Distributed Systems, 2nd edition 2008, John Wiley & Sons. One of the most comprehensive books about information security available.

B

- [Bachmann+2000] F. Bachmann et al., "Software Architecture Documentation in Practice: Documenting Architectural Layers," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA, Tech. Rep. CMU/SEI-2000-SR-004, Mar. 2000. [Online]. Available: https://insights.sei.cmu.edu/documents/5437/2000_003_001_13649.pdf
- [Bass+2021] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 4th ed. Boston, MA, USA: Addison Wesley, 2021. Although the title suggests otherwise, a quite fundamental (and sometimes abstract) book. The authors have a strong background in ultra-large scale (often military) systems - so their advice might sometimes conflict with small or lean kinds of projects.
- [Buschmann+1996] Buschmann, Frank/Meunier, Regine/Rohnert, Hans/Sommerlad, Peter: *A System of Patterns: Pattern-Oriented Software Architecture 1*, 1st edition, 1996, John Wiley & Sons.

Also known as POSA-1. Most likely the most famous and groundbreaking book on architecture patterns.

C

- [Clements+2003] Clements, P., F. Bachmann, L. Bass, D. Garlan, J. Ivers et al.: Documenting Software Architectures – Views and Beyond. Addison Wesley, 2003.
- [Cockburn 2005] Cockburn, Alistair (2005-04-01): Hexagonal architecture, online https://alistair.cockburn.us/hexagonal-architecture/ (retrieved 2024-07-25)

E

- [Evans 2004] Evans, Eric: *Domain-Driven Design: Tackling Complexity in the Heart of Software,* 1st edition, Addison-Wesley, 2004.

F

- [Ford+2017] Neil Ford, Rebecca Parsons, Patrick Kua: Building Evolutionary Architectures: Support Constant Change. OReilly 2017

G

- [GoF: Design-Patterns] Gamma, Erich/Helm, Richard/Johnson, Ralph/Vlissides, John M. *Design Patterns: Elements of Reusable Object-Oriented Software,* 1st edition, 1994, Addison-Wesley, 1994.

A classic on design patterns.

- [Gang-of-Four, short: GoF] See [GoF: Design-Patterns]

H

- [Hargis+2004] Hargis, Gretchen et al.: Quality Technical Information: A Handbook for Writers and Editors. Prentice Hall, IBM Press, 2004.
- [Hofmeister+2000] Hofmeister, Christine/Nord, Robert/Soni, Dilip]]]: *Applied Software Architecture,* 1st edition, Addison-Wesley, 1999
- [Hombergs 2024] Hombergs, Tom: Get Your Hands Dirty on Clean Architecture, Packt, 2nd edition 2024.

I

- [ISO-25010] ISO/IEC 25010:2023(en) Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Product quality model. Terms and definitions online: https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-2:v1:en
- [ISO-25019] ISO/IEC 25019:2023(en) Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Quality-in-use model. Terms and definitions online: https://www.iso.org/obp/ui/#iso:std:iso-iec:25019:ed-1:v1:en

K

- [Kazman+1996] Kazman, R., Abowd, G., Bass, L., & Clements, P.: *Scenario-based analysis of software architecture*, IEEE software, 13(6), 47-55, 1996.
- [Kruchten 1995] Kruchten, P.: Architectural Blueprints – The 4-1 View Model of Architecture. IEEE Software November 1995; 12(6), p. 42-50.

L

- [Lange 2021] Kenneth Lange: The Functional Core, Imperative Shell Pattern, online: https://www.kennethlange.com/functional-core-imperative-shell/
- [Lilienthal 2019] Lilienthal, Carola: *Langlebige Software-Architekturen: Technische Schulden analysieren, begrenzen und abbauen* 3rd edition, dpunkt.verlag, 2019

M

- [Maguire 2019] Sandy Maguire: Algebra-Driven Design: Elegant Solutions from Simple Building Blocks. Leanpub, 2019.
- [Martin 2003] Martin, Robert C.: *Agile Software Development: Principles, Patterns and Practices*, Prentice Hall, 2003
- [SOLID-principles] Martin, Robert: SOLID-principles. S.O.L.I.D is an acronym for the first five object-oriented design(OOD) principles by Robert C. Martin. Some original papers have been moved around onto various locations - see Wikipedia
- [McGraw 2006] Garry McGraw, "Software Security - Building Security In", Addison-Wesley 2006 Covering the whole process of software design from a security perspective by the means of risk management, code reviews, risk analysis, penetration testing, security testing abuse case development.

P

- [Parnas 1972] Parnas, David: *On the criteria to be used in decomposing systems into modules"*, Communications of the ACM, volume 15, issue 12, Dec 1972. One of the most influential articles ever written in software engineering, introducing encapsulation and modularity. Thank you, David!

R

- [Cherdantseva+2013] Yulia Cherdantseva, Jeremy Hilton, A Reference Model of Information Assurance & Security, 2013 Eight International Conference on Availability, Reliability and Security (ARES), DOI: 10.1109/ARES.2013.72, http://users.cs.cf.ac.uk/Y.V.Cherdantseva/RMIAS.pdf Conference Paper of Yulia Cherdantseva and Jeremy Hilton describing the RMIAS.
- [Rozanski+2011] Eoin Woods and Nick Rozanski: Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives. 2nd edition 2011, Addison-Wesley. Presents a set of architectural viewpoints and perspectives.

S

- [Schmidt+2000] D. C. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, *Pattern-Oriented Software Architecture, Volume 2: Patterns for Concurrent and Networked Objects*. New York, NY, USA: Wiley & Sons, 2000.
- [Schneier 1996] B. Schneier, *Applied Cryptography*, 2nd ed. New York, NY, USA: John Wiley & Sons, 1996.
- [Sperber+2024] Michael Sperber, Stefan Wehr: Datenmodellierung mit Summen und Produkten, 2024. https://funktionale-programmierung.de/2024/11/25/sums-products.html. (English translation: Data Modeling with Sums and Products, 2024. https://funktionale-programmierung.de/2024/11/25/sums-products-english.html)
- [Starke 2024] G. Starke, *Effektive Software-Architekturen - Ein praktischer Leitfaden*, 10th ed. Munich, Germany: Carl Hanser Verlag, 2024. Website: https://esabuch.de

T

- [Tanenbaum+2016] Andrew Tanenbaum, Maarten van Steen: Distributed Systems, Principles and Paradigms, 2016. https://www.distributed-systems.net/
- [Tornhill 2015] Adam Tornhill: Your Code as a Crime Scene. Use Forensic Techniques to Arrest Defects, Bottlenecks, and Bad Design in Your Programs. Pragmatic Programmers, 2015. https://www.adamtornhill.com

Y

- [Yorgey 2012] Brent A. Yorgey, Monoids: Theme and Variations. Proceedings of the 2012 Haskell Symposium, September 2012 https://doi.org/10.1145/2364506.2364520

Anhänge