1

iSAQB
Arbeitsgruppen

Glossar der Software-Architektur-Terminologie

Auflage 2020

iSAQB®
International Software Architecture Qualification Board

Leanpub

iSAQB Glossar der Softwarearchitektur-Terminologie

Gernot Starke, Ulrich Becker, Carola Lilienthal, Michael Mahlberg, Simon Kölsch, Roger Rhoades, Sebastian Fichtner, Andreas Rausch, Phillip Ghadir, Mahbouba Gharbi, Matthias Bohlen und Mirko Hillert

Dieses Buch steht unter http://leanpub.com/isaqbglossary zum Verkauf. Die vorliegende

Version wurde am 04.02.2020 veröffentlicht.

Dieses Buch tweeten!

Bitte helfen Sie Gernot Starke, Ulrich Becker, Carola Lilienthal, Michael Mahlberg, Simon Kölsch, Roger Rhoades, Sebastian Fichtner, Andreas Rausch, Phillip Ghadir, Mahbouba Gharbi, Matthias Bohlen und Mirko Hillert, indem Sie dieses Buch auf Twitter bekannt machen!

Vorgeschlagenes Hashtag für dieses Buch: #isaqbglossary.

Finden Sie heraus, was andere über dieses Buch sagen, indem Sie über diesen Link auf Twitter nach diesem Hashtag suchen.

#isaqbglossary

Inhaltsverzeichnis

Einleitung

Dieses Buch enthält ein Glossar der *Softwarearchitektur-Terminologie*.

Es kann als Referenz für die Vorbereitung auf die iSAQB e.V. Prüfung *Certified Professional for Software Architecture – Foundation Level*© dienen.

Bitte beachten Sie: Dieses Glossar ist **nicht** als Grundlagen- oder Lehrbuch zu Softwarearchitektur, sondern als Sammlung von Definitionen (und Links zu weiteren Informationen) gedacht.

Außerdem finden Sie Übersetzungsvorschläge für die iSAQB-Terminologie, derzeit vom Englischen ins Deutsche (und umgekehrt).

Und schließlich enthält dieses Buch zahlreiche Verweise auf Bücher und andere Quellen, die häufig in den Definitionen zitiert werden.

Dieses Buch wird laufend überarbeitet.

Fehler oder Lücken können auch in unserem Issue-Tracker auf Github[1] gemeldet werden, wo die Autoren die Originalquellen für dieses Buch verwahren.

Persönliche Anmerkungen

Mehrere Begriffe in diesem Buch wurden von einem oder mehreren Autoren kommentiert:

Anmerkung (Gernot Starke)

Manche Begriffe sind gegebenenfalls besonders wichtig, oder manchmal gilt es, subtilere Aspekte zu berücksichtigen. Diese Anmerkungen stellen die persönliche Meinung ihres Verfassers dar und spiegeln **nicht** notwendigerweise die des iSAQB wider.

[1] https://github.com/isaqb-org/glossary/issues

Verweise auf Begriffe

Alle Begriffe in diesem Glossar haben eine eindeutige URL zur (kostenlosen) Onlineversion des Buchs, daher kann sowohl in Online- als auch in Printdokumenten weltweit auf sie verwiesen werden.

Der Aufbau unserer URL ist recht einfach:

- Die Basis-URL lautet https://leanpub.com/isaqbglossary/read
- Wir stellen einfach dem entsprechenden Begriff das Präfix #term- voran, dann folgt der Begriff an sich, mit Bindestrichen („-“) anstelle von Leerzeichen.

  Für unsere Beschreibung des Begriffs *Software Architecture* kann beispielsweise ein Verweis (Hyperlink) gesetzt werden mit https://leanpub.com/isaqbglossary/read#term-software-architecture

Nahezu alle Begriffe sind mit ihrer vollständigen Bezeichnung und nur einige wenige mit ihren (gängigen) Abkürzungen, wie UML oder DDD, verlinkt.

Lizenz

[2] https://creativecommons.org/licenses/by/4.0/

Danksagung

Verschiedene Teile dieses Glossars wurden von den folgenden Freiwilligen oder Förderern beigesteuert:

- Die Definitionen von rund 120 Begriffen sind eine Schenkung von Gernot Starke und wurden ursprünglich für eines seiner Bücher[3] zusammengestellt.
- Eine Reihe von Definitionen im Bereich Systemverbesserung und -entwicklung wurden vom Open-Source-Projekt aim42[4] beigesteuert.

Beiträge

Beiträge sind willkommen

Wenn Sie Fehler, Lücken oder Rechtschreibfehler melden oder zusätzliche Inhalte beisteuern möchten – können Sie dies gerne auf einem der folgenden Wege tun:

1. Legen Sie ein Issue in unserer Github-Repository [a] an.
2. Forken Sie das Repository und erstellen Sie einen Pull-Request.
3. Diskutieren Sie Themen auf der Leanpub-Feedbackseite[b] des Buches.
4. Schreiben Sie eine E-Mail an die Autoren, ebenfalls über die Website[c] des Buches.

a https://github.com/isaqb-org/glossary/issues
b https://leanpub.com/isaqbglossary/feedback
c https://leanpub.com/isaqbglossary/email_author/new

Die Autoren wissen Ihre Mitwirkung sehr zu schätzen.

Grundsätzliches Problem “Maß”:

[3] https://leanpub.com/esa42/
[4] http://aim42.github.io/

Glossar

Abstraktion

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

Kategorie: Entwurfsprinzip

Abstraktheit

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

Kategorie: Kennzahl

Einheit?

Qualitätsmerkmal Zugänglichkeit

unterschiedlichen

Maß, in dem ein Produkt oder System von Personen ~~mit den unterschiedlichsten Eigenschaften und Fähigkeiten~~ zur Erreichung eines spezifizierten Ziels in einem spezifizierten Nutzungskontext genutzt werden kann. Teilmerkmal von: Benutzerfreundlichkeit. Vgl. Website von ISO 25010[5].

Kategorie: Qualität, ISO 25010

ACL

Was ist das?

Was ist das?

Zugriffskontrolllisten (Access Control Lists, ACL) sind eine Möglichkeit zur Organisation und Speicherung von Berechtigungen eines Principals für eine spezifische Entität. Neben Implementierungen auf Anwendungsebene ist ein typisches Beispiel für eine ACL die Verwaltung von Dateiberechtigungen in Unix-basierten Betriebssystemen.

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

Kategorie: Sicherheit

Unix (POSIX zumindest)

Qualitätsmerkmal Verantwortlichkeit

Maß, in dem Aktionen einer Entität eindeutig zu der Entität zurückverfolgt werden können. Teilmerkmal von: Sicherheit. Vgl. Website von ISO 25010[6].

Kategorie: Qualität, ISO 25010

[5] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010
[6] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010

Typografie falsch

Akkreditierung

Prüfungsverfahren und Zertifizierung durch eine ermächtigte Akkreditierungsstelle (in diesem Fall das iSAQB(R)) 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.

Akkreditierter Schulungsanbieter

Schulungsanbieter mit gültiger Akkreditierung des iSAQB(R).

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[7] für die *hierarchische* Zerlegung[8] 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 objektorie Software bezog, ist ADP ein *universeller* Grundsatz. Er geht (mindestens) auf ein 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.

Die Italics machen keinen Sinn

keine Italics

n argumentiert werden, dass ein Abhängigkeitszyklus, selbst vor Berücksichtigung seiner verschiedenen praktischen Probleme, logisch bereits so fehlerhaft ist wie ein Zirkelargument[9] oder eine Zirkeldefinition[10]. 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.

Was ist ein Prinzip-Ansatz?

Kategorie: Entwurfsprinzip

[7] https://en.wikipedia.org/wiki/Directed_acyclic_graph
[8] https://en.wikipedia.org/wiki/Functional_decomposition
[9] https://en.wikipedia.org/wiki/Circular_reasoning
[10] https://en.wikipedia.org/wiki/Fallacies_of_definition#Circularity

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. Vgl. Website von ISO 25010[11] .

Kategorie: Qualität, ISO 25010

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.

Kategorie: Entwurfsmuster, Foundation.

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.

Kategorie: DDD

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.

Kategorie: Foundation

Qualitätsmerkmal Analysierbarkeit

Maß der Effektivität und Effizienz, mit dem die Auswirkung einer 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. Vgl. Website von ISO 25010[12] .

Kategorie: Qualität, ISO 25010

Angemessenheit

Eignung für einen bestimmten Zweck.

[11] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010
[12] http://iso25000.com/index.php/en/iso-25000-standards/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. Vgl. Website von ISO 25010[13].

Kategorie: Qualität, ISO 25010

arc42

Kostenloses Template[14] zur Kommunikation und Dokumentation von Softwarearchitekturen. arc42 besteht aus 12 (optionalen) Teilen oder Abschnitten.

Kategorie: Kommunikation, Dokumentation

Architektur

Siehe Softwarearchitektur

Kategorie: 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).

Kategorie: ISO-IEC-IEEE-42010

Architekturbeschreibung

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

Kategorie: 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).

Kategorie: ISO-IEC-IEEE-42010

[13] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010
[14] http://arc42.org

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].

Kategorie: ISO-IEC-IEEE-42010

Architekturbewertung

Quantitative oder qualitative Beurteilung einer (Software- oder System-) Architektur. Ermittlung, 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.

keine Italics

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.

Kategorie: ISO-IEC-IEEE-42010

Architekturziel

“Merkmal” vs. “Frage” … häh?

(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*.

Kategorie: Grundlegend

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).

Kategorie: ISO-IEC-IEEE-42010

[Begriff entfernt, da auf Dt. kein Synonym]

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:

- Model-View-Controller
- Schichten
- Pipes und Filter
- CQRS

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).

Kategorie: 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.

Architektursicht

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

- Kontextabgrenzung,
- Bausteinsicht
- Laufzeitsicht
- Verteilungssicht

[Bass+2012] und [Rozanski+11] 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).

Kategorie: ISO-IEC-IEEE-42010

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).

Kategorie: ISO-IEC-IEEE-42010

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.

Was ist das?

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[15])

Kategorie: Sicherheit

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.

[15] https://en.wikipedia.org/w/index.php?title=Asset_(computer_security)&oldid=694606042

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

Kategorie: Foundation

Asymmetrische Kryptographie

Algorithmen der asymmetrischen Kryptographie 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, Public-Key Algorithms, Seite 17.

Asymmetrische Kryptographie ist die Grundlage für PKI und digitale Signaturen.

Kategorie: Sicherheit

ordentliche Referenz, den Link sieht man im Druck nicht

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.

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“[16].

Kategorie: Sicherheit

“Wahrscheinlich”? Ist gemeint: “Jeder Ansatz kann Abhängigkeiten haben …”

Audit-Arbeitsgruppe:

Die *Audit-Arbeitsgruppe* ist für die technische Beurteilung der Schulungsunterlagen sowie für die Überwachung und Beurteilung der Schulungen zuständig. Die vom iSAQB(R) 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.

[16] https://www.schneier.com/academic/archives/1999/12/attack_trees.html

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.

Kategorie: Sicherheit

Qualitätsmerkmal Authentifizierbarkeit

Maß, inwieweit nachgewiesen werden kann, dass die Identität eines Subjekts oder einer Ressource der Identitätsbehauptung entspricht. Teilmerkmal von: Sicherheit. Vgl. Website von ISO 25010[17].

Kategorie: Qualität, ISO 25010

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.“ ← “Zugriffsrichtlinie”

(Übersetztes englisches Zitat aus Wikipedia[18])

Kategorie: Sicherheit

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.

Kategorie: Sicherheit

Qualitätsmerkmal Verfügbarkeit

Maß, in dem ein System, ein Produkt oder eine Komponente einsatzfähig und zugänglich sind, wenn sie benötigt werden. Teilmerkmal von: Zuverlässigkeit. Vgl. Website von ISO 25010[19].

Kategorie: Qualität, ISO 25010, Sicherheit

[17] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010
[18] https://en.wikipedia.org/w/index.php?title=Authorization&oldid=739777234
[19] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010

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.

Kategorie: Foundation

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)

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)

Kategorie: DDD

Brücke

Entwurfsmuster, bei dem eine Abstraktion von ihrer Implementierung entkoppelt ist, so dass beide unabhängig voneinander variieren können. Wenn Ihnen das (wie den meisten Menschen) unverständlich erscheint – sehen hier[20] nach.

Kategorie: Entwurfsmuster

[20] http://www.cs.sjsu.edu/~pearce/modules/patterns/platform/bridge/index.htm

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.

Kategorie: Architekturmuster

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 Laufzeitszenarien.

Geschäftsarchitektur

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

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.

im

zu

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.

Kategorie: Sicherheit

Qualitätsmerkmal Kapazität

Maß, in dem die Höchstgrenzen eines Produkt- oder Systemparameters den Anforderungen entsprechen. Teilmerkmal von: Leistungseffizienz. Vgl. Website von ISO 25010[21] .

Kategorie: Qualität, ISO 25010

[21] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010

Kardinalität

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

Zertifizierungsprogramm

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

Die urheberrechtliche geschützte Abkürzung CPSA(R) steht für *Certified Professional for Software Architecture*.

CIA-Triade

Siehe Schutzziele

Kategorie: Sicherheit

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[22] (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, unnötiger Umbruch
  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, so dass 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.

[22] http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf

- 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.

Kohäsion

Maß, in dem Elemente eines Bausteins, einer Komponente oder eines Moduls zusammengehören. 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.

Kommando

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

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.

“Änderungswunsch”?

wirklich “soll” - wäre nicht “darf” besser?

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

“kleine”

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:

“großen”

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.

Kategorie: Entwurfsprinzip

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 anders herum: 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.

Kategorie: Entwurfsprinzip

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. Vgl. Website von ISO 25010[23].

Kategorie: Qualität, ISO 25010

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~~.

“die wir also zusätzlich erzeugen”

(Übersetztes englisches Zitat von Mark Needham[24])

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

Komponente

Siehe Baustein. Strukturelement einer Architektur.

[23] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010
[24] http://codebetter.com/markneedham/2010/03/18/essential-and-accidental-complexity/

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.

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*.

Belang

Belange in Bezug auf eine Architektur sind Anforderungen, Ziele, Einschränkungen, Absichten oder Bestrebungen eines Stakeholders für diese Architektur. ([Rozanski+11], 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.

Kategorie: ISO-IEC-IEEE-42010

Vertraulichkeit

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

Kategorie: Sicherheit

Qualitätsmerkmal Vertraulichkeit

Maß, in dem ein Produkt oder System sicherstellt, dass nur Zugriffsberechtigte Zugriff auf die Daten haben. Teilmerkmal von: Sicherheit. Vgl. Website von ISO 25010[25].

Kategorie: Qualität, ISO 25010

[25] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010

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[26]

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[27])

Kontextabgrenzung

Zeigt das vollständige System als eine Blackbox in seiner Umgebung, entweder aus Geschäftsperspektive (*Geschäftskontext*) oder aus technischer oder Verteilungsperspektive (*technischer Kontext*). 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.

Einheit? (Das Problem kommt noch öfter.)

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. Vgl. Website von ISO 25010[28].

Kategorie: Qualität, ISO 25010

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).

Kategorie: ISO-IEC-IEEE-42010

[26] http://agilemodeling.com/artifacts/constraint.htm
[27] http://www.viewpoints-and-perspectives.info/home/viewpoints/context/
[28] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010

Korrespondenzregel

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

Kategorie: ISO-IEC-IEEE-42010

Konsistenz

Ein konsistentes System enthält keine Widersprüche.

Einheit?

- 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, Montonic-Write-Consistency, Read-Your-Writes-Consistency etc.)
- In Bezug auf Verhalten: Maß, in dem ein System sich kohärent, reproduzierbar und vernünftig verhält.

Einheit?

Synonym: Integrität, Homogenität, konzeptionelle Integrität.

Kopplung

Was ist mit “Grad” gemeint? Einheit?

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.

CPSA(R)

Certified Professional for Software Architecture(R) – 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[29] zu diesem Thema.

[29] https://leanpub.com/cqrs

Querschnittskonzept

Siehe Konzept.

Synonym: Prinzip, Regel.

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.

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.

Kategorie: Kennzahl.

geschlechtergerechte Sprache wäre gut

zyklomatische

Zerlegung

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

Abhängigkeit

Siehe Kopplung.

Die persönliche Ansprache gibt's sonst nicht

Abhängigkeitsinjektion / Dependency Injection (DI)

Statt dass Ihre Objekte oder eine Fabrik eine Abhängigkeit erzeugen, übergeben Sie die benötigten Abhängigkeiten an den Konstruktor oder über Eigenschaft-Setter. Damit machen Sie die Erzeugung von spezifischen Abhängigkeiten zum *Problem anderer Leute*.

unklar … "im Objekt speichern"?

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[30] anschaulich erläutert, und das eng mit dem SDP und SAP zusammenhängt.

[30] http://martinfowler.com/articles/dipInTheWild.html

ausschreiben

ausschreiben

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[31])

Entwurfsmuster

Allgemeine oder generische wiederverwendbare Lösung für ein gängiges Problem in einem gegebenen Kontext beim Entwurf. Das ursprünglich von dem berühmten Architekten Christopher Alexander[32] 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[33] von Erich Gamma (Gamma+1994) und seinen drei Verbündeten kennen.

Was soll der “Unserer Ansicht” … “ernsthaft” … Quatsch?

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.

Diese Eigenschaften wurden von Robert Martin formuliert und stammen von OODesign.com[34]

Dokument

Ein (üblicherweise schriftliches) Artefakt zur Informationsvermittlung.

[31] http://www.viewpoints-and-perspectives.info/home/viewpoints/deployment/
[32] https://en.wikipedia.org/wiki/Christopher_Alexander [33] https://en.wikipedia.org/wiki/Design_Patterns
[34] http://www.oodesign.com/design-principles.html

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[35]). 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.

Mal-/Zeichenprogramm

Im Ernst?

Programm zur Erstellung von Zeichnungen, die in der Architekturdokumentation verwendet werden können. Beispiel: 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).

Sparsamkeit

Hier scheint was zu fehlen?

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.

[35] http://dddcommunity.org/learning-ddd/what_is_ddd/

Kapselung

2 oder 1?

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.

Das hier scheint mir nur das erste zu 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.

Entität

Kategorie: DDD

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.

Was bedeutet das?

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 kryptographische 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 11.1 Information Theory of Schneier-1996 und Whitewood Inc. zu „Understanding and Managing Entropy“[36] oder SANS „Randomness and Entropy - An Introduction“[37].

Kategorie: Sicherheit

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.

Kategorie: ISO-IEC-IEEE-42010

[36] https://www.blackhat.com/docs/us-15/materials/us-15-Potter-Understanding-And-Managing-Entropy-Usage-wp.pdf
[37] https://www.sans.org/reading-room/whitepapers/vpns/randomness-entropy-introduction-874

Fassade

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

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).

Einheit?

Qualitätsmerkmal Fehlertoleranz

Maß, in dem ein System, ein Produkt oder eine Komponente trotz Hardware- oder Softwarefehlern wie vorgesehen funktioniert. Teilmerkmal von: Zuverlässigkeit. Vgl. Website von ISO 25010[38].

Kategorie: Qualität, ISO 25010

Was heißt das?

Filter

Teil des „Pipes und Filter“-Architekturstils, der Daten erzeugt oder transformiert. Filter werden üblicherweise unabhängig von anderen Filtern ausgeführt.

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.

Einheit?

Qualitätsmerkmal funktionale Angemessenheit

Maß, in dem die Funktionen die Erfüllung von spezifizierten Aufgaben und Zielen ermöglichen. Teilmerkmal von: Funktionale Eignung. Vgl. Website von ISO 25010[39].

Kategorie: Qualität, ISO 25010

Einheit?

Qualitätsmerkmal funktionale Vollständigkeit

Maß, in dem der Satz von Funktionen alle spezifizierten Aufgaben und Benutzerziele abdeckt. Teilmerkmal von: Funktionale Eignung. Vgl. Website von ISO 25010[40].

Kategorie: Qualität, ISO 25010

[38] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010
[39] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010
[40] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010

Einheit?

Qualitätsmerkmal funktionale Korrektheit

Maß, in dem ein Produkt oder System die korrekten Ergebnisse mit dem benötigten Grad an Präzision liefert. Teilmerkmal von: Funktionale Eignung. Vgl. Website von ISO 25010[41] .

Kategorie: Qualität, ISO 25010

Einheit?

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. Vgl. Website von ISO 25010[42].

Kategorie: Qualität, ISO 25010

Gateway

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

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.

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[43] oder Benutzeroberflächenentwurf[44].

“gibt es im”

Hybrider Architekturstil

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

[41] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010
[42] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010
[43] http://www.vincehuston.org/ood/oo_design_heuristics.html
[44] https://www.nngroup.com/articles/ten-usability-heuristics/

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*.

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.

Einheit?

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. Vgl. Website von ISO 25010[45].

Kategorie: Qualität, ISO 25010

Integrität

Verschiedene Bedeutungen:

sieht aus wie ein AsciiDoc-Fehler

Kategorie: Sicherheit

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

Kategorie: Grundlegend

Einheit?

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

Siehe auch Konsistenz.

[45] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010

Einheit?

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. Vgl. Website von ISO 25010[46].

Kategorie: Qualität, ISO 25010, Sicherheit

Schnittstelle

Grenze, über die zwei Bausteine hinweg miteinander interagieren oder kommunizieren.

Schnittstellenaufteilungsprinzip (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, so dass Clients nur Methoden kennen müssen, die sie tatsächlich nutzen.

Einheit?

Qualitätsmerkmal Interoperabilität

Maß, in dem zwei oder mehr Systeme, Produkte oder Komponenten Informationen austauschen und die ausgetauschten Informationen nutzen können. Teilmerkmal von: Kompatibilität. Vgl. Website von ISO 25010[47].

Kategorie: Qualität, ISO 25010

iSAQB

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

ISO 9126

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

ISO 25010

Normen zur Beschreibung (und Bewertung) von *Softwareproduktqualität*. Das Qualitätsmodell legt fest, welche Qualitätseigenschaften bei der Bewertung der Eigenschaften eines Softwareprodukts berücksichtigt werden. (Übersetztes englisches Zitat von der ISO-Website[48])

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

[46] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010
[47] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010
[48] http://iso25000.com/index.php/en/iso-25000-standards/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[49]).

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

Gegensatz 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.

Kerckhoffs'sches Prinzip

Eines der sechs kryptographischen Axiome, die 1883 von dem niederländischen Kryptographen 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 kryptographische 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.

in Italics

Siehe Bruce Schneiers Crypto-Gram, May 15, 2002[50]

Kategorie: Sicherheit

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.

Einheit?

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. Vgl. Website von ISO 25010[51].

Kategorie: Qualität, ISO 25010

[49] http://c2.com/cgi/wiki?IterativeDevelopment
[50] https://www.schneier.com/crypto-gram/archives/2002/0515.html
[51] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010

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.

Einheit?

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. Vgl. Website von ISO 25010[52].

Kategorie: Qualität, ISO 25010

Einheit?

Qualitätsmerkmal Reifegrad

Maß, in dem ein System, ein Produkt oder eine Komponente im Normalbetrieb die Zuverlässigkeitsanforderungen erfüllt. Teilmerkmal von: Zuverlässigkeit. Vgl. Website von ISO 25010[53].

Kategorie: Qualität, ISO 25010

MFA

Multi-Faktor-Authentifizierung, siehe Authentifizierung.

Kategorie: Sicherheit

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[54] von Eberhard Wolff[55].

Modellgetriebene Architektur / Model Driven Architecture (MDA)

Link

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.

[52] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010
[53] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010
[54] https://leanpub.com/microservices-primer
[55] http://microservices-book.com

Modellgetriebene Softwareentwicklung / Model-driven software development (MDSD)

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

Modellart

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

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

Kategorie: ISO-IEC-IEEE-42010

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).

gegenüber was?

Model-View-Controller

in welchem Kontext?

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) Outputdarstellungen 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.

Qualitätsmerkmal Modifizierbarkeit

Einheit?

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. Vgl. Website von ISO 25010[56].

Kategorie: Qualität, ISO 25010

Qualitätsmerkmal Modularität

Einheit?

Maß, inwieweit ein System oder Computerprogramm aus diskreten Komponenten besteht, so dass eine Änderung an einer Komponente minimale Auswirkungen auf andere Komponenten hat. Teilmerkmal von: Wartbarkeit. Vgl. Website von ISO 25010[57].

Kategorie: Qualität, ISO 25010

[56] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010
[57] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010

Modul

Was ist das?

(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, so dass 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[58]
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(R)-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.

Typografie

Und was ist das hier?

Modulare Programmierung

„Softwareentwurfstechnik, die die Funktionalität eines Programms in unabhängige, austauschbare *Module* unterteilt, so dass 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[59])

Was ist das?

Knoten (in UML)

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

Node (Node.js)

In der modernen Webentwicklung: Kurz für die quelloffene JavaScript-Laufzeitumgebung Node.js®[60], 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.

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).

[58] http://www.cs.umd.edu/class/spring2003/cmsc838p/Design/criteria.pdf
[59] https://en.wikipedia.org/wiki/Modular_programming
[60] https://nodejs.org/en/

Einheit?

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. Vgl. Website von ISO 25010[61].

Kategorie: Qualität, ISO 25010

Notation

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

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. Das Beobachtermuster ist ein wesentlicher Bestandteil des MVC-Architekturmusters (model–view–controller).“ (Übersetztes englisches Zitat aus Wikipedia.)

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 ~~Robin Martins~~ für objektorientierte Systeme. Kann in objektorientierten Sprachen durch Schnittstellenvererbung, allgemeiner als *Plugins*, implementiert werden.

“Robert C. Martin”

Einheit?

Qualitätsmerkmal Bedienbarkeit

Maß, in dem ein Produkt oder System Eigenschaften aufweist, die es einfach bedien- und steuerbar machen. Teilmerkmal von: Benutzerfreundlichkeit. Vgl. Website von ISO 25010[63].

Kategorie: Qualität, ISO 25010

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/[64].

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

Kategorie: Sicherheit

[61] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010
[62] https://en.wikipedia.org/wiki/Observer_pattern
[63] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010
[64] https://www.owasp.org/

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[65] und häufig beide zusammen angeführt. Während die SOLID-Prinzipien auf die Klassen-Ebene 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).

Kategorie: Entwurfsprinzip

Muster

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

Siehe Architekturmuster oder Entwurfsmuster.

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.

Kategorie: Sicherheit

[65] [RobertC.Martin](http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod)

Qualitätsmerkmal Leistungseffizienz

kein Satz

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.

Vgl. Website von ISO 25010[66].

Kategorie: Qualität, ISO 25010

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+11] 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[67]. 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.

[66] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010
[67] https://simple.wikipedia.org/wiki/Pikachu

PKI

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

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

Kategorie: Sicherheit

Port

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

Einheit?

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. Vgl. Website von ISO 25010[68].

Kategorie: Qualität, ISO 25010

POSA

Pattern-oriented Software Architecture. Buchreihe zu Softwarearchitekturmustern.

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[69] und in verschiedenen Authentifizierungsprotokollen verwendet (siehe GSSAPI RFC2744[70] oder Kerberos RFC4121[71]).

Kategorie: Sicherheit

[68] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010
[69] https://docs.oracle.com/javase/8/docs/api/java/security/Principal.html
[70] https://tools.ietf.org/html/rfc2744
[71] https://tools.ietf.org/html/rfc4121

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[72])

Pseudo-Zufälligkeit

Häufig in Verbindung mit Pseudozufallszahlengeneratoren verwendet. Die Erzeugung von Zufälligkeit mit hoher Entropie ist ressourcenintensiv und, abgesehen von Kryptographie, 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.

Kategorie: Sicherheit

“chaotisch weitere”

als?

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*.

Link

Qualität

Siehe *Softwarequalität* und Qualitätsmerkmale.

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 ist hilfreich, zwischen folgenden Merkmalen zu unterscheiden:

[72] https://en.wikipedia.org/wiki/Proxy_pattern

- *Laufzeit-Qualitätsmerkmalen* (die während der Ausführungszeit des Systems beobachtet werden können),
- Nicht-Laufzeit-Qualitätsmerkmalen (die während der Ausführung des Systems nicht beobachtet werden können) und
- Geschäftsqualitätsmerkmalen (Kosten, Zeitplan, Marktfähigkeit, Eignung für Unternehmen)

Beispiele für Laufzeit-Qualitätsmerkmale sind funktionale Eignung, Leistungseffizienz, Sicherheit, Zuverlässigkeit, Benutzerfreundlichkeit und Interoperabilität.

Beispiele für Nicht-Laufzeit-Qualitätsmerkmale sind Modifizierbarkeit, Portierbarkeit, Wiederverwendbarkeit, Integrierbarkeit und Testbarkeit.

Qualitätseigenschaft

Synonym: *Qualitätsmerkmal*.

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.

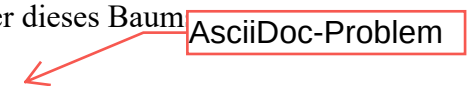
Der Anwendungsumfang von Qualitätsmodellen umfasst die Unterstützung der Spezifikation und Bewertung von Software und softwareintensiven Computersystemen aus unterschiedlichen Perspektiven durch an ihrem Erwerb, ihren Anforderungen, ihrer Entwicklung, ihrer Nutzung, ihrer Bewertung, ihrem Support, ihrer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

Qualitätsanforderung

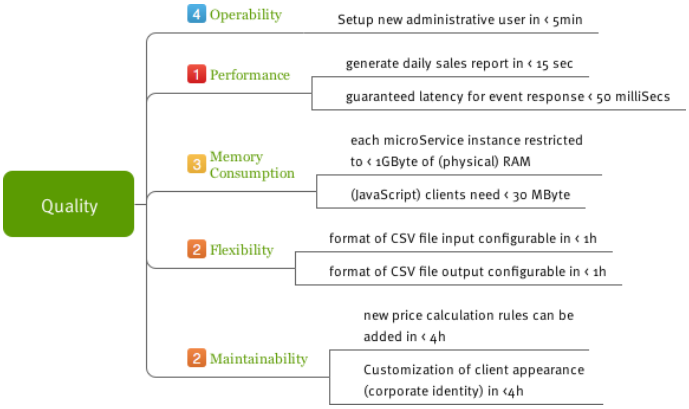
Eigenschaft oder Merkmal einer Komponente eines Systems. Beispiele sind Laufzeitleistung, Schutz, Sicherheit, Zuverlässigkeit oder Wartbarkeit. Siehe auch *Softwarequalität*.

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 Baum



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



Beispiel eines Qualitätsbaums

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.

Zufälligkeit

Siehe Entropie oder Pseudo-Zufälligkeit.

Kategorie: Sicherheit

Begründung

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

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.

Kategorie: Sicherheit

Einheit?

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. Vgl. Website von ISO 25010[73].

Kategorie: Qualität, ISO 25010

[73] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010

Redesign

Die Veränderung von Softwareeinheiten, so dass 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, so dass 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[74]). Häufig als *Singleton* (auch ein bekanntes Entwurfsmuster) implementiert.

Qualitätsmerkmal Zuverlässigkeit

Einheit?

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. Vgl. Website von ISO 25010[75].

Kategorie: Qualität, ISO 25010

Beziehung

“für”

“Abhängigkeiten”

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.

Qualitätsmerkmal Austauschbarkeit

Einheit?

Maß, in dem ein Produkt ein anderes spezifiziertes Softwareprodukt für den gleichen Zweck in der gleichen Umgebung ersetzen kann. Teilmerkmal von: Portierbarkeit. Vgl. Website von ISO 25010[76].

Kategorie: Qualität, ISO 25010

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.

[74] http://martinfowler.com/eaaCatalog/registry.html
[75] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010
[76] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010

Einheit?

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. Vgl. ~~Website von~~ ISO 25010[77]. Kategorie: Qualität, ISO 25010

Einheit?

Qualitätsmerkmal Wiederverwendbarkeit

Maß, in dem ein Asset in mehr als einem System oder zum Aufbau anderer Assets genutzt werden kann. Teilmerkmal von: Wartbarkeit. Vgl. ~~Website von~~ ISO 25010[78].

Kategorie: Qualität, ISO 25010

Reuse-Release-Equivalence-Prinzip

Was ist das? Und warum Anführungszeichen?

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.

Was für Komponenten?

Was ist ein Punkt?

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. 

Kategorie: Entwurfsprinzip

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[79]).

[77] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010
[78] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010
[79] https://en.wikipedia.org/wiki/Model-driven_software_development

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. Kategorie: Programmierung

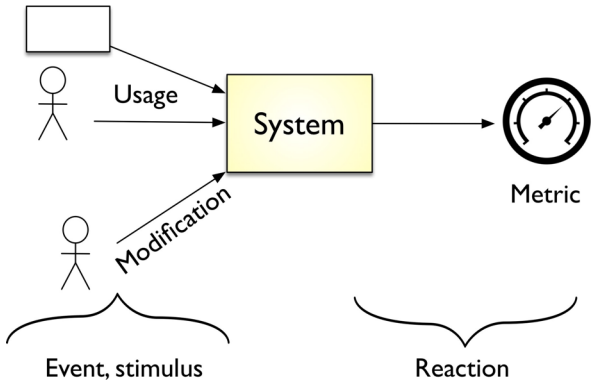
Laufzeitsicht

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.

Szenario

“UML-Sequenz-“ (mit Bindestrich hinten)

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.



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.

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, Black-/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 „An Enterprise Software Security Program“, Seite 2[illegible]

Kategorie: Sicherheit

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“ (RIMAS) erweitert diese Liste um Verantwortlichkeit, Prüfbarkeit, Authentifizierbarkeit/Vertrauenswürdigkeit, Nichtabstreitbarkeit und Datenschutz.

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

Siehe „What is Security Engineering - Definitions“, Seite 11 oder RMIAS.

Kategorie: Sicherheit

Qualitätsmerkmal Sicherheit

Maß, in dem ein Produkt oder System Informationen und Daten schützt, so dass Personen oder andere Produkte oder Systeme den ihren Berechtigungsarten oder -stufen entsprechenden Datenzugriffsgrad haben. Es besteht aus folgenden Teilmerkmalen: Vertraulichkeit, Integrität, Nichtabstreitbarkeit, Verantwortlichkeit, Authentifizierbarkeit. Vgl. Website von ISO 25010[80].

Kategorie: Qualität, ISO 25010

SCS (Self Contained System)

Ein Architekturstil, ähnlich wie Microservices. Auf scs-architecture.org[81] 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, so dass das vollständige System eine Zusammenarbeit vieler kleinerer Softwaresysteme ist. Dies verhindert das Problem großer Monolithen, die stetig wachsen und irgendwann nicht mehr wartbar sind.“

[80] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010
[81] http://scs-architecture.org/

Sensitivitätspunkt

Link

Link oder Referenz

(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.

“SoC besagt, dass …”

Separation of concern (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.

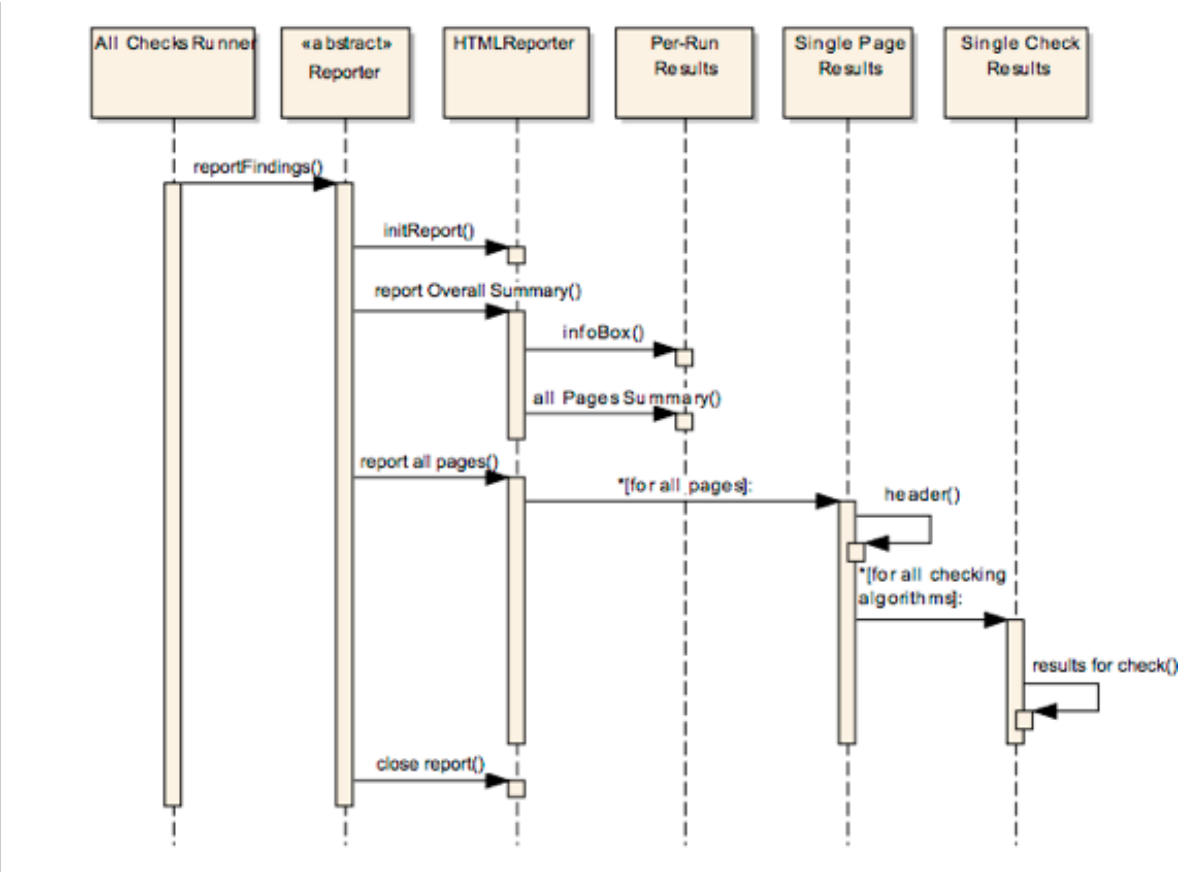
Eine weitere Definition lautet: Aufteilung eines Systems in Elemente, die sich möglichst wenig überschneiden.

~~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.



Beispiel eines Sequenzdiagramms

Dienst

zu erledigen

Service (DDD)

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

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[82].)

Single-Responsibility-Prinzip (SRP)

Was ist das?

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.

[82] https://en.wikipedia.org/wiki/Singleton_pattern

Softwarearchitektur:

Es gibt mehrere (!) gültige und plausible Definitionen des Begriffs *Softwarearchitektur*. Die IEEE 1471[83] Norm schlägt folgende Definition vor:

> Softwarearchitektur: Die grundlegende Organisation eines Systems, die in seinen Komponenten verkörpert ist, ihre Beziehungen untereinander und zur Umgebung sowie die Prinzipien, die seinen Entwurf und seine Entwicklung leiten.

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

> Architektur: (System) grundlegende Konzepte oder Eigenschaften eines Systems in seiner Umgebung, die in seinen Elementen, Beziehungen und den Prinzipien seines Entwurfs und seiner Weiterentwicklung verkörpert sind.

Die Schlüsselbegriffe dieser Definition bedürfen einer Erläuterung:

doppelter Doppelpunkt

- Komponenten: Teilsysteme, Module, Klassen, Funktionen oder allgemeiner gesagt *Bausteine*: Strukturelemente von Software: Komponenten werden üblicherweise in einer Programmiersprache implementiert, können aber auch andere Artefakte n, 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 Nachbar und von diesen~~ übertragen.

“Nachbarsysteme”

- 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*.

Link

Das *Software Engineering Institute* führt eine Sammlung weiterer Definitionen[84] Kategorie:

ISO-IEC-IEEE-42010

Einheit?

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.

Einheit?

(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.

“ISO-25000-Qualitätsmodellen”

[83] https://en.wikipedia.org/wiki/IEEE_1471
[84] http://www.sei.cmu.edu/architecture/start/glossary/classicdefs.cfm

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

Links für die einzelnen Begriffe

SOLID (Single-Responsibility, Open-Closed, Liskovsche Substitution, Interface-Segregation und Dependency-Inversion) ist ein (von Robert C. Martin[85]) 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 Martin-SOLID.

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~~.

ausschreiben

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.

ausschreiben

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).

Kategorie: Entwurfsprinzip

ausschreiben

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.

“sollte stabiler sein”?

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.

[85] http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod

Ursprüngliche Definitionen des SDP (wie Martin-2003) beinhalten eine Kennzahl *I* der Instabilität[86]. 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).

Kategorie: Entwurfsprinzip

ausschreiben

ausschreiben

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).

Kategorie: ISO-IEC-IEEE-42010

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.

visualisiert

Strukturelement

~~Siehe~~ *Baustein* oder *Komponente*

Link

Link

Symmetrische Kryptographie

Symmetrische Kryptographie basiert auf einem identischen Schlüssel für die Verschlüsselung und Entschlüsselung von Daten. Sender und Empfänger vereinbaren einen Schlüssel für die Kommunikation. Siehe Schneier, Symmetric Algorithms, Seite 17.

Kategorie: Sicherheit

[86] https://en.wikipedia.org/wiki/Software_package_metrics

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].
- Softwareintensive 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.

Kategorie: ISO-IEC-IEEE-42010

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).

Kategorie: ISO-IEC-IEEE-42010

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[87]

Qualitätsmerkmal Testbarkeit

Maß an Effektivität und Effizienz mit dem 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. Vgl. Website von ISO 25010[88]. Kategorie: Qualität, 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. Vgl. Website von ISO 25010[89].

Kategorie: Qualität, ISO 25010

[87] http://arc42.de
[88] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010
[89] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010

TLS

Transport-**L**ayer-**S**ecurity (Transportschichtsicherheit) bezeichnet eine Reihe von Protokollen zum kryptographischen 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“[90].

Kategorie: Sicherheit

TOGAF

The Open Group Architecture Framework. Konzeptioneller Rahmen für die Planung und Wartung von Unternehmens-IT-Architekturen.

Werkzeug- und Materialansatz

zu erledigen

Top-Down

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

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)

Wem seine?

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

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.

[90] https://tools.ietf.org/html/rfc7568

Schulungsleiter

Ein Trainer 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 Schulungsleitern organisieren und durchführen. Nur akkreditierte Schulungsanbieter können Akkreditierungen von Schulungsleitern beantragen.

Schulungslevel

Das iSAQB® CPSA Schulungsprogramm ist (derzeit) in zwei Schulungslevel gegliedert: *Foundation Level* und *Advanced Level*. Die Schulungslevel sollten aufeinander aufbauendes Wissen enthalten. Die genauen Beziehungen untereinander und die Inhalte dieser Schulungslevel sind in den jeweiligen Lehrplänen festgelegt.

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.

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.

Unified Modeling Language / Vereinheitlichte Modellierungssprache (UML)

"Grafische"

Graphische 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.

Qualitätsmerkmal Benutzerfreundlichkeit

Einheit?

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. Vgl. ~~Website von~~ ISO 25010[91]. Kategorie: Qualität, ISO 25010

[91] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010

Einheit?

Qualitätsmerkmal Schutz vor Fehlbedienung

Maß, in dem ein System Benutzer davor schützt, Fehler zu machen. Teilmerkmal von: Benutzerfreundlichkeit. Vgl. Website von ISO 25010[92]. Kategorie: Qualität, ISO 25010

Einheit?

Qualitätsmerkmal Ästhetik der Benutzeroberfläche

Maß, in dem eine Benutzeroberfläche dem Benutzer eine angenehme und zufriedenstellende Interaktion ermöglicht. Teilmerkmal von: Benutzerfreundlichkeit. Vgl. Website von ISO 25010[93]. Kategorie: Qualität, ISO 25010

Nutzungsbeziehung

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

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.

Sicht

“dürfen nicht verändert werden”

Siehe: Architektursicht.

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 t aus [http://c2.com/cgi/wiki?IterativeDevelopment])

Link

Gegensatz zur *iterativen Entwicklung*

richtiger Link

geschlechtergerechte Sprache

Netz des Vertrauens / Web of Trust

ausschreiben

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.

ausschreiben

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

[92] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010
[93] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010

Whitebox

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

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

Was ist “wegabstrahieren”?

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

Dem Satz fehlt ein Subjekt

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.

Übersetzungen

Hier finden Sie Übersetzungen von englischen Begriffen ins Deutsche (s. unten) und deutschen Begriffen ins Englische (nächster Abschnitt).

Einige dieser Begriffe beruhen auf den rechtlichen und organisatorischen Grundlagen des iSAQB-Vereins (und haben daher nichts mit Softwarearchitektur zu tun).

Die folgenden Übersetzungen werden in einer einfachen JSON-Eingabedatei gepflegt[94], die sich im Open-Source-Github-Repository[95] findet.

Englisch-Deutsch-Übersetzungen

> Wichtiger Hinweis: Diese Übersetzungstabelle erhebt keinen Anspruch auf Vollständigkeit, verschiedene englische Begriffe werden nicht übersetzt, sondern vorzugsweise in der Originalsprache verwendet (z.B. viele Entwurfsmuster-Bezeichnungen).

Die folgenden Tabellen wurden mit Groovy und Gradle automatisch aus JSON erzeugt[96].

Englisch	Deutsch
Adaption	Anpassung
Adequacy	Angemessenheit
Approach	Ansatz
Appropriateness	Angemessenheit
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
Availability	Verfügbarkeit
Building block	Baustein

[94] Die Dokumentation auf https://github.com/isaqb-org/glossary enthält alle Informationen, die zur Erzeugung der Übersetzungstabellen erforderlich sind. Derzeit werden nur englisch und deutsch unterstützt. Die Übersetzungstabellen werden im JSON-Format geführt, Verbesserungsvorschläge sind sehr willkommen!

[95] https://github.com/isaqb-org/glossary

[96] 133 englische Begriffe, erzeugt am 04. Oktober 2016

Englisch	Deutsch
Building block view	Bausteinsicht
Business	Fachlichkeit, Domäne
Business architecture	fachliche Architektur, Geschäftsarchitektur
Cabinet (as methaphor for template)	Schrank (als Metapher für Template)
Cash audit	Rechnungsprüfung
Cash auditor	Rechnungsprüfer
Certification authority	Zertifizierungsstelle
Certification body	Zertifizierungsstelle
Chairman	Vorsitzender
Channel	Kanal
Cohesion	Kohäsion, innerer Zusammenhalt
Commensurability	Angemessenheit, Messbarkeit, Vergleichbarkeit
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
Crosscutting	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-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

Englisch	Deutsch
Fees rules and regulations	Gebührenordnung
General meeting	Mitgliederversammlung
Influencing Factor	Einflussfaktor
Information hiding principle	Geheimnisprinzip
Integrity	Integrität
Interdependency (between design decisions)	Abhängigkeit (zwischen Entwurfsentscheidungen)
Interface	Schnittstelle
Interface description	Schnittstellenbeschreibung, Schnittstellendokumentation
Learning goal	Lernziel
License fee	Lizenzgebühr
License holder	Lizenznehmer
Licensee	Lizenznehmer
Licensing agreement	Lizenzvertrag, Lizenzvereinbarung
Licensor	Lizenzgeber
Local court	Amtsgericht
Means for describing	Beschreibungsmittel
Means for documenting	Beschreibungsmittel
Measurability	Messbarkeit
Members' meeting	Mitgliederversammlung
Modeling Tool	Modellierungswerkzeug
Module	Komponente, Modul, Baustein
Node	Knoten
Non-exclusive license	Einfache Lizenz
Non-profit	Gemeinnützig
Notification	Benachrichtigung
Objective	Ziel
Operational processes	Betriebsprozesse (von Software)
Pattern	Muster
Pattern language	Mustersprache, Musterfamilie
Perspective	Perspektive
Principle	Prinzip, Konzept
Quality attribute	Qualitätsmerkmal, Qualitätseigenschaft
Quality characteristic	Qualitätsmerkmal, Qualitätseigenschaft
Quality feature	Qualitätsmerkmal, Qualitätseigenschaft
Rationale	Begründung, Erklärung
Real-time system	Echtzeitsystem
Registered trademark	Marke (gesetzlich geschützt)
Relationship	Beziehung
Relationship (kind of)	Beziehungsart
Repository	Ablage
Requirement	Anforderung
Resolution	Beschluss
Responsibility	Verantwortlichkeit
Rights of use	Nutzungsrecht
Runtime	Laufzeit
Runtime view	Laufzeitsicht
Security Goals	Schutzziele, Sachziele
Skill	Fähigkeit, Fertigkeit

Englisch	**Deutsch**
Specification (of software architecture)	Beschreibung (von Softwarearchitektur)
sponsoring (board) member	materiell förderndes Mitglied
statutory	satzungsgemäß
Structure	Struktur
Task	Aufgabe
Team regulations	Arbeitsgruppenordnung
Term	Begriff
Thriftyness	Sparsamkeit, Wirtschaftlichkeit
Tools	Arbeitsmittel, Werkzeug
Tools-and-material-approach	Werkzeug-Material-Ansatz
Tradeoff	Kompromiss, Abwägung
Training provider	Schulungsanbieter
Treasurer	Schatzmeister
Uses relationship	Benutzt-Beziehung, Nutzungsbeziehung
View	Sicht
Workflow management	Ablaufsteuerung
Working environment	Arbeitsumgebung
Working group	Arbeitsgruppe
Working group head	Arbeitsgruppenleiter

Deutsch-Englisch-Übersetzungen

In diesem Abschnitt sammeln wir die iSAQB-Übersetzung der Begriffe aus dem Deutschen ins Englische.

> Wichtiger Hinweis: Diese Übersetzungstabelle erhebt keinen Anspruch auf Vollständigkeit, verschiedene englische Begriffe werden nicht übersetzt, sondern vorzugsweise in der Originalsprache verwendet (z.B. viele Entwurfsmuster-Bezeichnungen).

Deutsch	Englisch
Abhängigkeit	Coupling, Dependency
Abhängigkeit (zwischen Entwurfsentscheidungen)	Interdependency (between design decisions)
Ablage	Repository
Ablaufsteuerung	Workflow management
Abwägung	Tradeoff
Amtsgericht	Local court
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
Architekturziel	Architectural objective, Architecture objective
Artefakt	Artifact
Aspekt	Aspect
Aufgabe	Task
Baustein	Building block, Component, Module
Bausteinsicht	Building block view
Begriff	Term
Begründung	Rationale
Begutachtung	Assessment
Belang	Aspect, Concern
Benachrichtigung	Notification
Benutzt-Beziehung	Uses relationship
Beschluss	Resolution
Beschreibung (von Softwarearchitektur)	Specification (of software architecture)
Beschreibungsmittel	Means for describing, Means for documenting

Deutsch	Englisch
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
Erklärung	Rationale
Fachdomäne	Domain
fachliche Architektur	Business architecture, Domain-related architecture
Fachlicher Bereich	Domain
Fachlichkeit	Business
Fertigkeit	Skill
Fähigkeit	Skill
Gebührenordnung	Fees rules and regulations
Geheimnisprinzip	Information hiding principle
Gemeinnützig	Non-profit
Geschäftsarchitektur	Business architecture
Geschäftsbereich	Domain
innerer Zusammenhalt	Cohesion
Integrität	Integrity
Kanal	Channel
Kapselung	Encapsulation
Knoten	Node
Kohäsion	Cohesion
Komponente	Component, Module
Kompromiss	Tradeoff
Kontextabgrenzung	Context view
Konzept	Principle
Kopplung	Coupling
Laufzeit	Runtime
Laufzeitsicht	Runtime view
Lehrplan	Curriculum
Lernziel	Learning goal
Lizenzgeber	Licensor
Lizenzgebühr	License fee

Deutsch	Englisch
Lizenznehmer	Licensee, License holder
Lizenzvereinbarung	Licensing agreement
Lizenzvertrag	Licensing agreement
Mal-/Zeichenprogramm	Drawing Tool
Marke (gesetzlich geschützt)	Registered trademark
materiell förderndes Mitglied	sponsoring (board) member
Messbarkeit	Commensurability, Measurability
Mitgliederversammlung	General meeting, Members' meeting
Modellierungswerkzeug	Modeling Tool
Modul	Module
Muster	Pattern
Musterfamilie	Pattern language
Mustersprache	Pattern language
Nutzungsbeziehung	Uses relationship
Nutzungsrecht	Rights of use
Perspektive	Perspective
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	Crosscutting
Randbedingung	Constraint
Rechnungsprüfer	Cash auditor
Rechnungsprüfung	Cash audit
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
Schutzziele	Security Goals
Schätzung	Estimation
Sicht	Architectural view, View
Sparsamkeit	Economicalness, Thriftyness
Stellvertretender Vorsitzender	Deputy chairman
Struktur	Structure
Unternehmens-IT-Architektur	Enterprise IT architecture
Untersuchung	Assessment
Verantwortlichkeit	Responsibility

Deutsch	Englisch
Verein	Association
Verfügbarkeit	Availability
Vergleichbarkeit	Commensurability
Verteilung	Deployment
Verteilungsartefakt	Deployment unit
Verteilungssicht	Deployment view
Vertraulichkeit	Confidentiality
Vorsitzender	Chairman
Vorstand	Executive board
Werkzeug	Tools
Werkzeug-Material-Ansatz	Tools-and-material-approach
Wirtschaftlichkeit	Economicalness, Thriftyness
Zerlegung	Decomposition
Zertifizierungsstelle	Certification authority, Certification body
Ziel	Objective

Kategorien

Wir verwenden Kategorien zur besseren Strukturierung der Begriffe im Glossar. Jeder Begriff kann keiner oder mehreren der folgenden Kategorien angehören:

Architekturmuster
: Bezeichnung eines Architekturmusters oder -stils aus z.B. [Buschmann+96], [Fowler2003], [Hohpe+2003], [Quian+2010] oder anderen Grundlagenwerken.

Kommunikation
: Wird zur Kommunikation von Informationen zu beliebigen Aspekten der Softwarearchitektur verwendet oder benötigt.

DDD
: Schlüsselwort aus dem CPSA-Advanced-Lehrplan „Domain Driven Design“

Entwurfsmuster
: Bezeichnung eines Entwurfsmusters aus z.B. [Gamma+95] oder anderen Grundlagenwerken.

Entwurfsprinzip
: Bezeichnung eines grundlegenden Entwurfsprinzips.

Foundation
: Begriffe, die im CPSA-Foundation-Lehrplan vorausgesetzt werden oder enthalten sind.

Grundlegend
: Grundlegender Begriff

Improve
: Schlüsselwort aus dem CPSA-Advanced-Lehrplan „IMPROVE“.

iSAQB
: Offizielle iSAQB-Begriffe; viele von ihnen werden für Vertrags-, Vereins- oder andere Organisationszwecke verwendet.

Kennzahl
: Definiertes Maß, inwieweit ein Softwaresystem (oder der zugehörige Prozess) eine bestimmte Eigenschaft besitzt. Beispiele: Größe (z.B. Codezeilen, zyklomatische Komplexität, Kopplung, mittlere Betriebsdauer zwischen Ausfällen)

Sicherheit
: Maßnahmen zur Erreichung grundlegender Prinzipien, wie Vertraulichkeit, Integrität, Verfügbarkeit und Nichtabstreitbarkeit für Daten in einem IT-System.

ISO-IEC-IEEE-42010
: ISO/IEC/IEEE 42010:2011 System- und Software-Engineering – Architekturbeschreibung. Hinweis: Eine neue Version der Norm sollte bis Ende 2016 veröffentlicht werden.

Literaturverzeichnis

Anderson-2008

Ross Anderson, Security Engineering - A Guide to Building Dependable Distributed Systems, 2. Auflage 2008, John Wiley & Sons.

Eines der ausführlichsten Werke zu Informationssicherheit.

Bachmann+2000

Bachmann, Felix/Bass, Len/Carriere, Jeromy/Clements, Paul/Garlan, David/Ivers, James/Nord, Robert/Little, Reed. *Software Architecture Documentation in Practice*, Special Report CMU/SEI- 2000-SR-004, 2000.

Bass+2012

Bass, L/Clements, P/Kazman, R.: *Software Architecture in Practice*

3. Auflage, Addison-Wesley, 2012. Auch wenn aufgrund des Titels anders zu vermuten, handelt es sich um ein grundlegendes (und zuweilen abstraktes) Buch. Die Autoren haben einen starken Hintergrund in ultragroßen (häufig militärischen) Systemen – daher stehen ihre Ratschläge gegebenenfalls manchmal im Widerspruch zu kleinen oder schlanken Projekten.

Brown-2015

Brown, Simon: *Software Architecture For Developers*,

https://leanpub.com/software-architecture-for-developers[97] Leanpub Publishing. Äußerst praktisch und pragmatisch.

Buschmann+1996

Auch POSA-1 genannt.

Buschmann, Frank/Meunier, Regine/Rohnert, Hans/Sommerlad, Peter: *A System of Patterns: Pattern-Oriented Software Architecture 1*, 1. Auflage, 1996, John Wiley & Sons.

Höchstwahrscheinlich das berühmteste und bahnbrechendste Werk zu Architekturmustern.

Buschmann+2007

Auch POSA-4 genannt.

Buschmann, Frank/Henney, Kevlin/Schmidt, Douglas C.: *Pattern-Oriented Software Architecture: A Pattern Language for Distributed Computing*, Band 4, 2007, John Wiley & Sons.

Beschreibt eine Mustersprache für verteiltes Rechnen und stellt dem Leser die besten Praktiken und Hauptaspekte beim Bau von verteilten Softwaresystemen vor. Das Buch verbindet Hunderte von eigenständigen Mustern, Mustersammlungen und Mustersprachen aus dem Literaturbestand der POSA-Reihe.

[97] https://leanpub.com/software-architecture-for-developers

Buschmann+2007b

Auch POSA-5 genannt.

Buschmann, Frank/Henney, Kevlin/Schmidt, Douglas C.: *Pattern-Oriented Software Architecture: On Patterns and Pattern Languages*, Band 5, 2007, John Wiley & Sons.

Eine Metaerklärung, behandelt die Frage, was eine Mustersprache ist und vergleicht verschiedene Musterparadigmen.

Clements+2002

Clements, Paul/Kazman, Rick/Klein, Mark: *Evaluating Software Architectures: Methods and Case Studies*, Addison-Wesley, 2001.

Clements+2010

Clements, Paul/Bachmann, Felix/Bass, Len/Garlan, David/Ivers, James/Little, Reed/Merson, Paulo/Nord, Robert. *Documenting Software Architectures: Views and Beyond*, 2. Auflage, Addison Wesley, 2010

Evans-2004

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

Fowler-2003

Fowler, Martin: *Patterns of Enterprise Application Architecture,* Addison-Wesley, 2003.

Hervorragende Unterstützung beim Bau von Informationssystemen.

Gamma+1994

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

Ein Klassiker zum Thema Entwurfsmuster.

GoF (Gang of Four)

Siehe Gamma+1994

Gorton-2011

Gorton, I. *Essential Software Architecture*, 2. Auflage, Springer, 2011

Hargis+2004

Hargis, Gretchen/Carey, Michelle/Hernandez, Ann: *Developing Quality Technical Information: A Handbook for Writers and Editors,* IBM Press, 2. Auflage, Prentice Hall, 2004.

Wenn Sie viele Dokumentationen erstellen müssen, sollten Sie sich dieses Buch ansehen.

Hofmeister+2000

Hofmeister, Christine/Nord, Robert/Soni, Dilip: *Applied Software Architecture,* 1. Auflage, Addison-Wesley, 1999

Hohpe+2003

Hohpe, G/Woolf, B: *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*, Addison Wesley, 2003.

Ein sehr wichtiges und zeitloses Werk für alle, die integrierte Systeme erstellen.

Was ist das?

Kelly+2009

Steven Kelly und Risto Pohjonen: *Worst Practices for Domain-Specific Modeling* IEEE Software, Band 26, Nr. 4 Juli/August 2009, S. 22-30.

Die Autoren erläutern verschiedene *schlechte Praktiken* der Domänenmodellierung und geben Tipps, wie und warum sie vermieden werden können/sollen.

keine Italics

Kruchten-1995

Kruchten, Philippe. *The 4+1 View Model of Architecture*, IEEE Software, Band 12 (6), S. 45-50, 1995.

Lilienthal-2019

Lilienthal, Carola: *Langlebige Software-Architekturen: Technische Schulden analysieren, begren- zen und abbauen* 3. Auflage, dpunkt.verlag, 2019

Martin-SOLID

Martin, Robert C: S.O.L.I.D.

S.O.L.I.D ist ein Akronym für die ersten fünf Prinzipien des objektorientierten Entwurfs (OOD) von Robert C. Martin. Einige der ursprünglichen Werke wurden an unterschiedliche Orte verschoben - siehe:

- Wikipedia zu SOLID[98]
- Eine schöne Einführung[99] von Samuel Oloruntoba[100]

[98] https://en.wikipedia.org/wiki/SOLID_(object-oriented_design)
[99] https://scotch.io/bar-talk/s-o-l-i-d-the-first-five-principles-of-object-oriented-design
[100] https://scotch.io/@kayandrae

Martin-2003

Martin, Robert C: *Agile Software Development: Principles, Patterns and Practices*, Prentice Hall, 2002

McGraw-2006

Garry McGraw, “Software Security - Building Security In”, Addison-Wesley 2006

Deckt den gesamten Prozess des Softwareentwurfs aus Sicherheitsperspektive mit Risikomanagement, Code-Reviews, Risikoanalyse, Penetrationstests, Sicherheitstests und Missbrauchsfall-Entwicklung ab.

Parnas-1972

Parnas, David: *On the criteria to be used in decomposing systems into modules”*, Communications of the ACM, Band 15, Ausgabe 12, Dez. 1972

Einer der einflussreichsten Artikel aller Zeiten zum Thema Softwareentwicklung, der erstmals Kapselung und Modularität vorstellt.

POSA-1

Pattern-Oriented Software Architecture, Band 1. Siehe Buschmann+1996.

POSA-2

Pattern-Oriented Software Architecture, Band 2. Siehe Schmidt+2000.

POSA-4

Pattern-Oriented Software Architecture, Band 4. Siehe Buschmann+2007.

POSA-5

Pattern-Oriented Software Architecture, Band 5. Siehe Buschmann+2007b.

Quian+2010

Qian, K/Fu, X/Tao, L/Xu, C/Herrera, J: *Software Architecture and Design Illuminated,* 1. Auflagen, Jones und Bartlett, 2010.

Gut strukturierte und lesbare Zusammenstellung von Architekturstilen und -mustern.

Rozanski+2011

Rozanski, Nick/Woods, Eoin: *Software Systems Architecture - Working with Stakeholders Using Viewpoints and Perspectives*, 2. Auflage, Addison Wesley 2011.

Stellt eine Reihe von Architekturblickwinkeln und -perspektiven vor.

RMIAS-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

Konferenzpapier von Yulia Cherdantseva und Jeremy Hilton zu Beschreibung des RMIAS.

Schmidt+2000

Auch POSA-2 genannt.

Schmidt, Douglas C/Stal, Michael/Rohnert, Hans/Buschmann, Frank. Pattern-Oriented Software Architecture, Band 2: *Patterns for Concurrent and Networked Objects,* Wiley & Sons, 2000

Schneier-1996

Bruce Schneier, Applied Cryptography, 2. Auflage 1996, John Wiley & Sons. Umfassender Überblick über die moderne Kryptographie.

Shaw+1996

Shaw, Mary/Garlan, David: *Software Architecture: Perspectives on an Emerging Discipline*, Prentice Hall, 1996

Starke-2018

Starke, Gernot: *Effective Software Architectures: iSAQB CPSA-Foundation© Certification Study Guide* Leanpub, 2018. Online verfügbar https://leanpub.com/esa42[101].

Ein Lernleitfaden für die CPSA-Foundation-Prüfung.

[101] https://leanpub.com/esa42

Anhang A: Das iSAQB e.V.

Das International Software Architecture Qualification Board (iSAQB e.V., http://isaqb.org) ist eine gemeinnützige Organisation mit Mitgliedern aus der Industrie, von Entwicklungs- und Beratungsunternehmen, aus Bildung, Wissenschaft und anderen Organisationen.

Es ist ein Verein (e.V.) nach deutschem Recht mit folgenden Zielen:

- Erstellung und Pflege von konsistenten **Lehrplänen** für Softwarearchitekten.
- **Festlegung von Zertifizierungsprüfungen** basierend auf den verschiedenen CPSA-Lehrplänen.
- **Gewährleistung einer guten Unterrichtsqualität** für Softwarearchitekten.
- Gewährleistung einer hohen Qualität seiner Softwarearchitektur-Zertifizierungen.

iSAQB legt Ausbildung- und Prüfungsordnungen fest, aber führt selbst keine Schulungen oder Prüfungen durch. iSAQB-Schulungen werden von (lizenzierten) Schulungs- und Prüfungsorganisationen durchgeführt.

iSAQB überwacht und prüft die Qualität dieser Schulungen und aller zugehörigen Prozesse (z.B. Zertifizierungsverfahren).

Anhang B: Über die Autoren

Gernot Starke

Dr. Gernot Starke (innoQ[102]-Fellow) ist Mitgründer und langjähriger Nutzer des (quelloffenen) arc42[103]-Dokumentationstemplate. Er ist seit mehr als 20 Jahren als Softwarearchitekt, Coach und Consultant tätig und stellt sich erfolgreich der Herausforderung, effektive Softwarearchitekturen für Kunden aus verschiedenen Branchen zu entwickeln.

2008 hat Gernot das International Software Architecture Qualification Board (iSAQB e.V.[104]) mitgegründet und unterstützt es seitdem als aktives Mitglied.

Gernot hat mehrere (deutsche) Bücher zu Softwarearchitektur und verwandten Themen verfasst und dieses Glossar ins Leben gerufen.

Er lebt mit seiner Frau (*Cheffe Uli*) in Köln.

Ulrich Becker

Ulrich Becker arbeitet als Principal Consultant bei Method Park[105], seine Schwerpunkte sind Softwarearchitektur und Application Life Cycle Management.

Ulrich hat Informatik an der Universität Hamburg und der Universität Erlangen-Nürnberg studiert. Er hat 2003 an der Universität Erlangen-Nürnberg mit seiner Arbeit zur modellbasierten Verteilungskonfiguration promoviert. Anschließend wurde er Gruppenleiter der Gruppe Adaptive Systemsoftware am Fraunhofer IIS[106].

Seit 2005 ist Ulrich bei Method Park als Schulungsleiter, Consultant und Coach tätig und unterstützt dort seine Kunden bei der Verbesserung ihrer Entwicklungsprozesse und -methoden. Die meisten seiner Kunden sind aus der Automobilindustrie und anderen stark regulierten Branchen.

Ulrich ist Gründungsmitglied des iSAQB e.V.[107] und arbeitet dort in den Foundation-Level- und Advanced-Level-Arbeitsgruppen mit. Er lebt mit seiner Familie in Erlangen.

Matthias Bohlen

Matthias Bohlen, unabhängiger Experte[108] für effektive Produktentwicklung, begann seine Karriere als Softwareentwickler im Jahr 1980. Er schrieb Compiler für den MC68020-Prozessor von Motorola, der damals, als es noch keinen IBM PC gab, ziemlich revolutionär war. Und die Compiler verkauften sich hervorragend.

[102] http://innoq.com
[103] https://arc42.org
[104] https://isaqb.org
[105] http://www.methodpark.de
[106] http://www.iis.fraunhofer.de/
[107] http://isaqb.org
[108] http://mbohlen.de

Seitdem hat Matthias mit zahllosen Softwareteams gearbeitet und ihnen geholfen, funktionierende Software herauszubringen, ohne den Verstand zu verlieren. Und das tut er auch heute noch.

Matthias ist aktives Mitglied des International Software Architecture Qualification Board[109], schreibt einen Blog[110], ist im Bereich Schlank/Agil bekannt und trägt bei Konferenzen zu Softwareentwicklung vor.

Phillip Ghadir

Mitglied der Geschäftsleitung der innoQ Deutschland GmbH. Seit vielen Jahren berät Phillip Kunden aus verschiedenen Branchen zu Themen rund um Softwarearchitektur, -technik und -entwicklung. Er ist Mitgründer des iSAQB und hält regelmäßig Schulungen zu Softwarearchitektur ab.

Carola Lilienthal

Dr. Carola Lilienthal ist Softwarearchitektin und Mitgründerin der WPS Workplace-Solutions[111]. Sie arbeitet seit 20 Jahren als Entwicklerin, Projektmanagerin, Coach, Consultant und Architektin. Carola ist früh in Domain-Driven Design und agile Entwicklung eingestiegen und ist erfolgreich für zahlreiche Kunden aus unterschiedlichen Bereichen, hauptsächlich aus der Finanz-, Versicherungs- und Logistikbranche, tätig.

Seit 2003 analysiert sie Softwaresysteme in Java, C+, C#, PHP, ABAP und berät Entwicklungsteams, wie sie die Langlebigkeit ihres Codes verbessern können. Carola hält regelmäßig Vorträge bei Konferenzen und hat mehrere Artikel sowie ein Buch zu langlebiger Softwarearchitektur geschrieben.

Carola unterstützt seit 2008 das International Software Architecture Qualification Board (iSAQB e.V.[112]) als aktives Mitglied.

Mahbouba Gharbi

Mahbouba Gharbi ist CIO von iTech Progress, Buchautorin und Konferenzrednerin.

Seit einigen Jahren ist Mahbouba Vorstandsvorsitzende des iSAQB. Sie lebt mit ihrer Familie in Mannheim.

Mirko Hillert

Mirko ist Vorsitzender und CEO der iSAQB GmbH, dem kommerziellen Arm des iSAQB-Vereins. Er hat die meisten eher formalen Begriffe zu Akkreditierung u.Ä. beigesteuert.

[109] http://www.isaqb.org
[110] http://mbohlen.de
[111] https://wps.de
[112] http://isaqb.org

Simon Kölsch

Simon Kölsch arbeitet als Senior Consultant bei innoQ mit dem Schwerpunkt Web-Architektur und Security.

Begeistern kann er sich für Lösungen, die über den klassischen „Enterprise-Monolithen“ hinausgehen, inklusive der Architektur von verteilten Systemen sowie ihrer Infrastruktur, Logging und Monitoring.

Auf eine spezifische Technologie oder Programmiersprache ist er nicht festgelegt, hat aber einen starken JVM-Hintergrund.

Michael Mahlberg

Michael Mahlberg führt seine eigene Methodenberatung[113] in Deutschland und verbringt seine Zeit zumeist damit, Kunden bei ihrer Suche nach effektiveren Arbeitsweisen zu unterstützen, vor allem durch die Anwendung von schlanken und agilen Konzepten.

Seit er 18 ist, führt er seine eigenen Unternehmen im Bereich Computer und Software und hat schnell erkannt, dass Softwarearchitektur und (Entwicklungs-) Prozesse in gewisser Weise zeitlose Aspekte des Handwerks sind.

Heute konzentriert sich seine Arbeit zum großen Teil auf Prozesse und Interaktionen – ein Bereich, in dem er sowohl beruflich als auch ehrenamtlich tätig ist (beispielsweise gehört er zu den Gründern und Betreibern der Limited WIP Society Cologne[114]).

Die Architekturarbeit von Michael befasst sich daher tendenziell eher mit den Auswirkungen und Implikationen von Architektur- und Prozessentscheidungen *aufeinander* und den zugehörigen Optimierungsstrategien.

Andreas Rausch

Prof. Dr. Andreas Rausch leitet den Lehrstuhl für Software Systems Engineering an der Technischen Universität Clausthal.

Er hat 2001 an der Technischen Universität München, am Lehrstuhl von Prof. Dr. Manfred Broy, promoviert. Seine Forschungsschwerpunkte im Bereich des Software Systems Engineering sind Softwarearchitekturen, modellbasierte Softwareentwicklung und Vorgehensmodelle. In diesen Gebieten hat er mehr als 70 internationale Publikationen veröffentlicht.

Roger Rhoades

Roger Rhoades ist Gründer von Albion[115], einem Schulungs- und Consultingunternehmen in Deutschland.

Roger verfügt über mehr als 25 Jahre Erfahrung im Bereich Unternehmens-, Geschäfts- und Softwarearchitektur sowie im Management von internationalen Teams und Projekten. Diese praktische Erfahrung fließt in seine Schulungen ein, um sicherzustellen, dass die Teilnehmer nicht nur den theoretischen Inhalt, sondern auch die Herausforderungen bei dessen Umsetzung in der realen Welt verstehen.

[113] http://consulting-guild.de
[114] http://lwscologne.de
[115] https://albionacademy.de

Seit 2012 trägt Roger regelmäßig auf internationalen Konferenzen (z.B. EAMKon, Lean42 EAM, IT Strategy and Governance) vor.

Roger ist seit 2014 aktives Mitglied des International Software Architecture Qualification Board (iSAQB e.V.[116]). Zusätzlich zum iSAQB-Glossar unterstützt er aktiv die Weiterentwicklung der Foundation- und Advanced-Lehrpläne, Prüfungsfragen und Fallstudien.

Sebastian Fichtner

Gründer von flowtoolz.com[117]. App-Entwickler und Consultant. Begann 1995 mit dem Coden. Begeistert sich seitdem für Architektur. Arbeitet im Bereich Original-Apps und Open Source sowie an Projekten für verschiedene Kunden. Liebt Apple-Plattformen und die Sprache Swift.

[116] http://isaqb.org
[117] https://www.flowtoolz.com

Anhang C: Unser Anliegen

ELECTRONIC FRONTIER FOUNDATION eff.org

Alle Einnahmen aus diesem Buch werden der EFF gespendet. Indem Sie für dieses Buch bezahlen, unterstützen Sie deren Anliegen:

„Die **Electronic Frontier Foundation** ist die führende gemeinnützige Organisation zur Verteidigung der Bürgerrechte in der digitalen Welt. Die 1990 gegründete EFF setzt sich durch strategische Klageführung, Politikanalyse, Graswurzelaktivismus und Technologieentwicklung für die Privatsphäre der Nutzer, freie Meinungsäußerung und Innovation ein. Wir engagieren uns dafür, dass unsere Rechte und Freiheiten mit zunehmender Nutzung von Technologie verstärkt und geschützt werden.

Schon als das Internet noch in den Kinderschuhen steckte, hat die EFF erkannt, dass der Schutz des Zugangs zu Entwicklungstechnologie für die Freiheit aller von entscheidender Bedeutung ist. In den darauffolgenden Jahren hat die EFF mit ihrer radikal unabhängigen Stimme den Weg für Open-Source-Software, Verschlüsselung, Sicherheitsforschung, Filesharing-Werkzeuge und eine Welt aufstrebender Technologien bereitet.

Heute nutzt die EFF die einzigartige Fachkompetenz führender Technologen, Aktivisten und Anwälte zur Verteidigung der Redefreiheit im Internet, zur Bekämpfung von illegaler Überwachung, zum Eintritt für Nutzer und Innovatoren und zur Unterstützung von freiheitsfördernden Technologien.

Gemeinsam haben wir ein den gesamten Globus umspannendes Netz aus besorgten Mitgliedern und Partnerorganisationen aufgebaut. Die EFF berät politische Entscheidungsträger und leistet mit umfassenden Analysen, didaktischen Materialien und Aktivisten-Workshops Aufklärungsarbeit gegenüber der Presse und der Öffentlichkeit. Die EFF ermächtigt durch ihr Action Center hunderttausende Menschen und ist zu einer führenden Stimme in Debatten zu Onlinerechten geworden.

Die EFF ist eine spendenfinanzierte gemeinnützige Organisation gemäß US 501(c)(3) und ist von Ihrer Unterstützung abhängig, um sich weiterhin für Nutzer einsetzen zu können.“

(Übersetztes englisches Zitat von eff.org/about[118])

[118] http://eff.org/about