Inoffizieller Glossar für Begriffe rund um Softwarearchitektur im Foundation Level

2022.1-DE, 20230321

Inhaltsverzeichnis

Einführung

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

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

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

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

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

Dieses Buch ist in Arbeit und unfertig.

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

Persönliche Kommentare

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

Kommentar (Markus Harrer)

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

Begriffe können referenziert werden

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

Das Schema der URL ist sehr einfach:

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

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

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

License

Danksagung

Begriffe

A

Abhängigkeit

Siehe Kopplung.

Abhängigkeits-Umkehr-Prinzip / Dependency Inversion Principle

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

Abhängigkeitsinjektion / Dependency Injection (DI)

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*.

Abstraktion (Ergebnis)

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

Abstraktion (Vorgehen)

Entfernens von Details, um die Aufmerksamkeit auf Aspekte von größerer Bedeutung zu lenken. Ähnlich wie der Prozess der Verallgemeinerung.

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.

Adapter

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

zusammenzuarbeiten.

Aggregat

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

Aggregation

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

Angemessenheit

Eignung für einen bestimmten Zweck.

arc42

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

Architektur

Siehe Softwarearchitektur.

Architektur-Qualitätsanforderung

Siehe Architekturziel.

Architekturbegründung

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

Architekturbeschreibung

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

Architekturbewertung

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

Siehe Beurteilung

Anmerkung (Gernot Starke)

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

Architekturblickwinkel

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

Architekturentscheidung

Entscheidung mit nachhaltiger oder wesentlicher Auswirkung auf die Architektur.

Beispiel: Entscheidung über Datenbanktechnologie oder technische Grundlagen der Benutzeroberfläche.

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

Architekturmodell

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

Architekturmuster

Ein Architekturmuster beschreibt eines grundlegendes strukturelles Organisationsschema für Softwaresysteme. Es liefert eine Reihe von vordefinierten Teilsystemen, spezifiziert ihre Verantwortlichkeiten und enthält Richtlinien für die Organisation der Beziehungen zwischen ihnen ([buschmann1996], Seite 12). Vergleichbar mit Architekturstil.

Beispiele:

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

Architektursicht

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

- Kontextabgrenzung
- Bausteinsicht
- Laufzeitsicht
- Verteilungssicht

[bass] 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).

Architekturstil

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

Architekturtaktik

TODO

Architekturziel

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

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

Artefakt

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

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.

B

Baustein

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

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

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

Bausteinsicht

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

Begründung

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

Belang

Belange in Bezug auf eine Architektur sind Anforderungen, Ziele, Einschränkungen, Absichten oder Bestrebungen eines Stakeholders für diese Architektur. ([rozanski-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.

Beurteilung

Siehe auch Bewertung.

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

Beziehung

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

Blackbox

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

Bottom-up-Ansatz

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

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

C

C4 Model

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

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

D

Dienst

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

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

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

(basierend auf Wikipedia)

Dokument

Ein (üblicherweise schriftliches) Artefakt zur Informationsvermittlung.

Dokumentation

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

Dokumentationserstellung

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

Domain-Driven Design (DDD)

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

Siehe auch:

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

Domänenmodell

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

E

Eingebettete Systeme

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

Einschränkung

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

Gestützt auf eine Definition von Scott Ambler

Entwurfsbegründung

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

Entwurfsmuster

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

Unserer Ansicht nach sollte jeder ernsthafte Softwareentwickler zumindest einige Muster aus dem bahnbrechenden Buch Gang-of-Four von Erich Gamma ([ref-gamma-1994]) und seinen drei Verbündeten kennen.

Entwurfsprinzip

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

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

Gleichzeitig sollten folgende *gute Eigenschaften* angestrebt werden:

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

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

F

Fachlicher Kontext

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

Siehe Kontextabgrenzung.

Filter

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

Fitnessfunktion

"Eine architektonische Fitnessfunktion bietet eine objektive Integritätsbewertung einiger architektonischer Merkmale." ([ref-ford-2017]).

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

G

Geschäftsarchitektur

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

H

Heuristik

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

Hybrider Architekturstil

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

I

Inkrementelle Entwicklung

Siehe iterative und inkrementelle Entwicklung.

Integrität

Verschiedene Bedeutungen:

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

Daten- oder Verhaltensintegrität:

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

Siehe auch Konsistenz.

ISO 25010

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

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

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

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

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

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

Iterative Entwicklung

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

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

Gegensätzlich zur Wasserfall-Entwicklung.

Iterative und inkrementelle Entwicklung

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

K

Kapselung

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

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

Kapselung ist ein Mechanismus zum Verbergen von Informationen.

Kohäsion

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

Komplexität

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

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

(Übersetztes englisches Zitat von Mark Needham)

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

Komponente

Siehe Baustein. Strukturelement einer Architektur.

Konsistenz

Ein konsistentes System enthält keine Widersprüche.

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

Kontext (eines Systems)

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

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

Unterscheide zwischen *geschäftlichem* und *technischem* Kontext:

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

Kontextabgrenzung

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

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

Siehe Kontext.

Kontextgrenze

Kontextgrenze ist ein Prinzip des Strategieentwurfs von Domain-Driven Design. "Eine Kontextgrenze definiert ausdrücklich den Kontext, in dem ein Domänenmodell für ein Softwaresystem gilt. Idealerweise wäre ein einziges, einheitliches Modell für alle Systeme in derselben Domäne am besten. Dies ist zwar ein ehrenwertes Ziel, aber in Wirklichkeit ist es

normalerweise in mehrere Modelle zerstückelt. Es ist sinnvoll, dies so hinzunehmen und damit zu arbeiten." (Übersetztes englisches Zitat aus Wikipedia)

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

Konzept

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

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

Konzepte bilden die Basis für konzeptionelle Integrität.

Konzeptionelle Integrität

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

Kopplung

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

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

L

Latenz

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

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

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

Laufzeitsicht

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

Liskovsches Substitutionsprinzip

Bezieht sich auf die objektorientierte Programmierung: Wenn Vererbung genutzt wird, dann richtig: Instanzen von abgeleiteten Typen (Unterklassen) müssen vollständig an die Stelle ihrer Basistypen treten können. Wenn der Code Basisklassen verwendet, können diese Referenzen durch jede beliebige Instanz einer abgeleiteten Klasse ersetzt werden, ohne dass dies die Funktionalität des Codes beeinträchtigt.

M

Microservices

Architekturstil, der die Unterteilung von großen Systemen in kleine Einheiten vorschlägt, um damit Softwaresysteme u.a. schneller auszuliefern und kostengünstiger zu skalieren.

Model-View-Controller

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

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

Modellgetriebene Softwareentwicklung / Model-driven software development (MDSD)

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

Modellierungswerkzeug

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

Modul

(Siehe auch Modulare Programmierung)

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

Muster

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

Siehe Architekturmuster oder Entwurfsmuster.

N

Netz des Vertrauens (EN: Web of Trust)

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

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

Nichtfunktionale Anforderung

Anforderungen, die *die Lösung einschränken*. Nichtfunktionale Anforderungen werden auch als

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

Notation

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

Nutzungsbeziehung

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

O

Open-Close-Prinzip (OCP)

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

P

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

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.

Port

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

Q

Qualitative Bewertung

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

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

Siehe auch Beurteilung.

Qualität

Siehe Softwarequalität und Qualitätsmerkmale.

Qualitätsanforderung

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

Qualitätsbaum

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

- Standards zu Produktqualität, wie ISO 25010, enthalten Vorschläge von *allgemeinen*

Qualitätsbäumen.

- Die Qualität eines spezifischen Systems kann mit einem *spezifischen* Qualitätsbaum beschrieben werden (siehe nachfolgendes Beispiel).

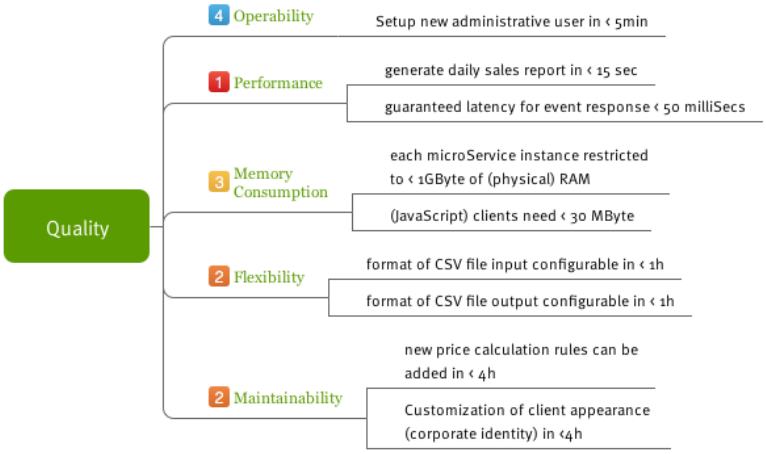

Figure 1. Beispiel eines Qualitätsbaums

Qualitätseigenschaft

Synonym: Qualitätsmerkmal.

Qualitätsmerkmal

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

In der ISO-25010 Norm sind folgende Qualitätsmerkmale definiert:

- Funktionale Eignung
  - Funktionale Vollständigkeit,
  - Funktionale Korrektheit
  - Funktionale Angemessenheit
- Leistungseffizienz
  - Zeitverhalten
  - Ressourcenverbrauch
  - Kapazität
- Kompatibilität
  - Koexistenz
  - Interoperabilität
- Benutzerfreundlichkeit
  - Erkennbarkeit der Brauchbarkeit
  - Erlernbarkeit

    - Bedienbarkeit
    - Schutz vor Fehlbedienung
    - Ästhetik der Benutzeroberfläche
    - Zugänglichkeit
- Zuverlässigkeit
    - Verfügbarkeit
    - Fehlertoleranz
    - Wiederherstellbarkeit
- Sicherheit
    - Vertraulichkeit
    - Integrität
    - Nichtabstreitbarkeit
    - Verantwortlichkeit
    - Authentifizierbarkeit
- Wartbarkeit
    - Modularität
    - Wiederverwendbarkeit
    - Analysierbarkeit
    - Modifizierbarkeit
    - Testbarkeit
- Portierbarkeit
    - Adaptierbarkeit
    - Installierbarkeit
    - Austauschbarkeit

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

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

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

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

Qualitätsmodell

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

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

Kommentar (Gernot Starke)

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

Quantitative Bewertung

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

Querschnittsbelang

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

Siehe auch Konzept.

Querschnittskonzept

Siehe Konzept

Synonym: Prinzip, Regel.

R

Redesign

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

Refactoring

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

Nicht mit Re**design** zu verwechseln.

Risiko

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

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

Round-Trip-Engineering

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

Anmerkung (Gernot Starke)

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

Anmerkung (Matthias Bohlen)

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

S

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

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

Weitere Quellen: siehe [martin-solid].

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.

Schnittstelle

Mehrere Bedeutungen, je nach Kontext:

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

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

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

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

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

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

Schnittstellenaufteilungsprinzip (Interface Secration Principle, ISP)

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

Sensitivitätspunkt

(In der qualitativen Bewertung, wie ATAM): Element des Architektursystems, das mehrere Qualitätsmerkmale beeinflusst. Wenn beispielsweise eine Komponente *sowohl* für die

Laufzeitleistung *als auch* die Robustheit verantwortlich ist, ist diese Komponente ein Sensitivitätspunkt.

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

Separation of Concerns (SoC)

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

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

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

Ähnlich wie das Single-Responsibility-Prinzip.

Sequenzdiagramm

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

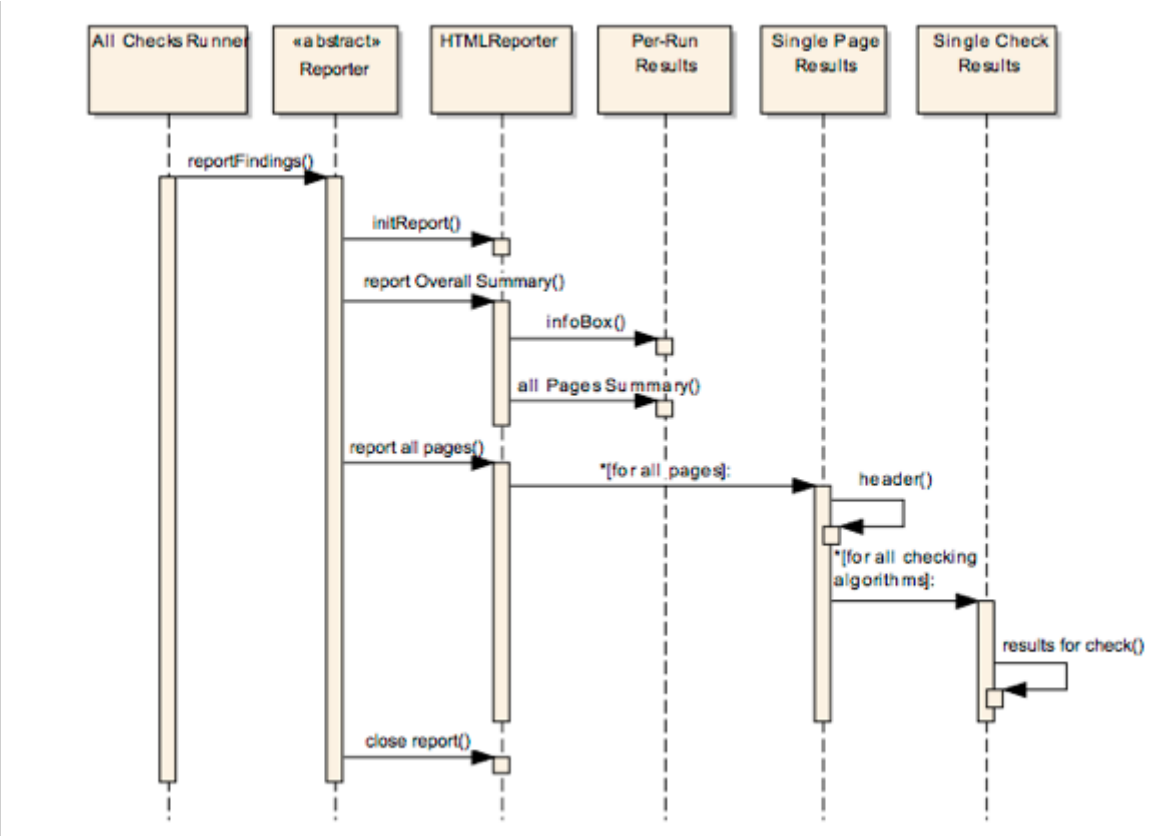

Figure 2. Beispiel eines Sequenzdiagramms

Sicht

Siehe Architektursicht.

Single-Responsibility-Prinzip (SRP)

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

Kohäsion wird manchmal als gleichbedeutend mit SRP angesehen.

Softwarearchitektur

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

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

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

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

Die Schlüsselbegriffe dieser Definition bedürfen einer Erläuterung:

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

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

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

Entwicklungsprojekten oder -organisationen).

Softwarequalität

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

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

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).

Struktur

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

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

Strukturelement

Siehe Baustein oder Komponente

System

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

- Systeme gemäß Beschreibung in [ISO/IEC 15288]: Systeme, die vom Menschen geschaffen wurden und mit einem oder mehreren der folgenden Aspekte konfiguriert werden können: Hardware, Software, Daten, Menschen, Prozesse (z.B. Prozesse zur Bereitstellung eines Dienstes für Benutzer), Verfahren (z.B. Bedieneranweisungen), Anlagen, Material und natürlich vorkommende Entitäten.
- Softwareprodukte und Dienste gemäß Beschreibung in [ISO/IEC 12207].

- Software-intensive Systeme gemäß Beschreibung in [IEEE Std 1471:2000]: jegliche Systeme, in denen Software wesentliche Einflüsse zum Entwurf, zur Entwicklung, Verbreitung und Weiterentwicklung des Systems als Ganzes beisteuert, um individuelle Anwendungen, Systeme im herkömmlichen Sinne, Teilsysteme, Systemverbünde, Produktlinien, Produktfamilien, ganze Unternehmen und sonstige Interessensvereinigungen zu umspannen.

Szenario

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

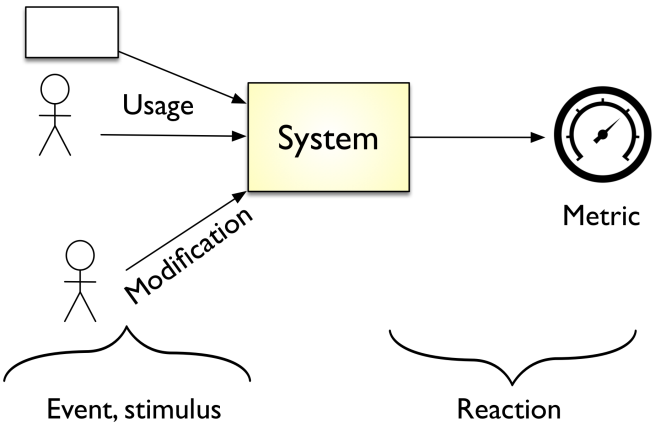

Figure 3. Allgemeine Form eines (Qualitäts-)Szenarios

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

Üblicherweise werden Qualitätsszenarien in drei Kategorien unterteilt

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

T

Technischer Kontext

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

Siehe Kontextabgrenzung.

Template (zur Dokumentation)

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

Ein sehr bekanntes Template ist arc42

Top-Down

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

U

Unified Modeling Language (UML)

Unified Modeling Language / Vereinheitlichte Modellierungssprache (UML)

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

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

Unternehmens-IT-Architektur

Synonym: Unternehmensarchitektur.

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

V

Verbergen von Informationen

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

Implementierung.

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

Verfolgbarkeit

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

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

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

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)

W

Wasserfall-Entwicklung

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

Siehe auch iterativen Entwicklung

Whitebox

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

Siehe auch Blackbox.

Workflow-Management-System (WFMS)

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

Wrapper

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

Anmerkung (Gernot Starke)

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

Z

Zeitliche Kopplung

Es gibt unterschiedliche Interpretationen aus verschiedenen Quellen. Zeitliche Kopplung

- bedeutet, dass Prozesse, die miteinander kommunizieren, beide aktiv sein müssen. Siehe [tanenbaum-2016].
- wenn du oft verschiedene Komponenten gleichzeitig eincheckst (*modifizierst*). Siehe [tornhill-2015].
- wenn es eine implizite Beziehung zwischen zwei oder mehr Mitgliedern einer Klasse gibt, die es erforderlich macht, dass die Clients das eine Mitglied vor dem anderen aufrufen. Mark Seemann, siehe Design Smell Temporal Coupling
- bedeutet, dass ein System auf die Antwort eines anderen Systems warten muss, bevor es mit der Bearbeitung fortfahren kann. Siehe Rest Antipattern

Zerlegung

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