Kapitel 2 Motivation

albion.eu www.tectrain.ch www.accso.de

< Kapitel 1 Grundlegendes zum Modul FLEX

Kapitel 3 Modularisierung >

Kapitel 2 Motivation

FLEX Lehrplan

2 Motivation

Dauer: 120 Min Übungszeit: Keine

2.1 Begriffe und Konzepte

Verfügbarkeit, Zuverlässigkeit, Time-to-Market, Flexibilität, Vorhersagbarkeit, Reproduzierbarkeit, Homogenisierung der Stages, Internet /Web-Scale, verteilte Systeme, Parallelisierbarkeit der Feature-Entwicklung, Evolution der Architektur (Build for Replacement), Heterogenität, Automatisierbarkeit.

2.2 Lernziele

2.2.1 Was sollen die Teilnehmer können?

- Architekturen können auf unterschiedliche Qualitätsziele hin optimiert werden. In diesem Modul lernen die Teilnehmer, wie sie flexible Architekturen erstellen, die schnelles Deployment und damit schnelles Feedback aus der Anwendung des Systems erlauben.
- Sie haben die Treiber für die Architektur-Typen verstanden, die in diesem Lehrplanmodul vermittelt werden, welche Konsequenzen die Treiber für die Architekturen haben und wie die Wechselwirkung der Architekturen mit Organisation, Prozessen und Technologien ist.
- Sie haben die Tradeoffs der vorgestellten Architektur-Typen (mindestens Microservices, Self Contained Systems und Deployment Monolithen) verstanden und können diese sowohl vermitteln als auch im Rahmen konkreter Projekte/Systementwicklungen anwenden, um angemessene Architekturentscheidungen zu treffen.

2.2.2 Was sollen die Teilnehmer verstehen?

- Auf die Fähigkeit, neue Features schnell in Produktion bringen zu können, hat die Architektur entscheidenden Einfluss.
- Abhängigkeiten zwischen Komponenten, die von unterschiedlichen Entwicklungsteams verantwortet werden, beeinflussen die Dauer, bis Software in Produktion gebracht werden kann, weil sie die Kommunikationsaufwände erhöhen und sich Verzögerungen fortpflanzen.
- Die Automatisierung von Aktivitäten (wie z. B. Test- und Deployment-Prozesse) erhöht die Reproduzierbarkeit, Vorhersagbarkeit und Ergebnisqualität dieser Prozesse. Das kann zu einer Verbesserung des gesamten Entwicklungsprozesses führen.
- Die Vereinheitlichung der verschiedenen Umgebungen (z. B. Entwicklung, Test, QA, Produktion) reduziert das Risiko von spät entdeckten und (in anderen Umgebungen) nicht reproduzierbaren Fehlern aufgrund unterschiedlicher Konfigurationen.
- Die Vereinheitlichung und Automatisierung sind wesentliche Aspekte von Continuous Delivery.
- Continuous Integration ist eine Voraussetzung für Continuous Delivery.
- Eine geeignete Architektur ist die Voraussetzung für eine Parallelisierbarkeit der Entwicklung sowie die unabhängige Inbetriebnahme von eigenständigen Bausteinen. Das können, müssen aber nicht „Services“ sein.

- Einige Änderungsszenarien lassen sich leichter in monolithischen Architekturen umsetzen. Andere Änderungsszenarien lassen sich leichter in verteilten Service-Architekturen umsetzen. Beide Ansätze können kombiniert werden.
- Es gibt unterschiedliche Arten der Isolation mit unterschiedlichen Vorteilen. Beispielsweise kann der Ausfall auf eine Komponente begrenzt werden oder Änderungen können auf eine Komponente begrenzt werden.
- Bestimmte Arten der Isolation sind zwischen Prozessen mit Remote-Kommunikation deutlich einfacher umzusetzen.
- Remote-Kommunikation hat aber Nachteile – z. B. viele neue Fehlerquellen.

2.2.3 Was sollen die Teilnehmer kennen?

- Gesetz von Conway
- Partitionierbarkeit als Qualitätsmerkmal
- Durchlaufzeiten durch die IT-Wertschöpfungskette als Wettbewerbsfaktor
- Aufbau einer Continuous-Delivery-Pipeline
- Die verschiedenen Test-Phasen in einer Continuous-Delivery-Pipeline

2.3 Referenzen

- Jez Humble, David Farley: Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, Addison-Wesley, 2010, ISBN 978-0-32160-191-9
- Eberhard Wolff: Continuous Delivery: Continuous Delivery: Der pragmatische Einstieg, dpunkt, 2014, ISBN 978-3-86490-208-6
- Jez Humble, Barry O'Reilly, Joanne Molesky: Lean Enterprise: Adopting Continuous Delivery, DevOps, and Lean Startup at Scale, O' Reilly 2014, ISBN 978-1-44936-842-5

Inhalte

(A) Architekturarbeit: Design für Anforderungen

1. Architekturarbeit mit der Architekturbrezel

Info
Stefan Toth: "Vorgehensmuster für Softwarearchitektur. Kombinierbare Praktiken in Zeiten von Agile und Lean". https://www.hanser-elibrary.com/doi/book/10.3139/9783446460096

In einem iterativen und fortlaufenden Prozess werden Anforderungen an die Software gesammelt, priorisiert und in eine Architektur umgesetzt (einschließlich Feedback und Reflexion).

Input und Feedback kommen von verschiedenen Stakeholdern (Entwicklungsteam, Benutzer, Business, Betrieb, Sicherheitsteam, Management etc.).

Der Gesamtprozess lässt sich am besten mit der "Architekturbrezel" darstellen, siehe nachstehendes Bild. Dabei ist insbesondere Wert auf einen effizienten *und* effektiven Gesamtprozess zu legen, um die Durchlaufzeiten durch die IT-Wertschöpfungskette gezielt zu optimieren.

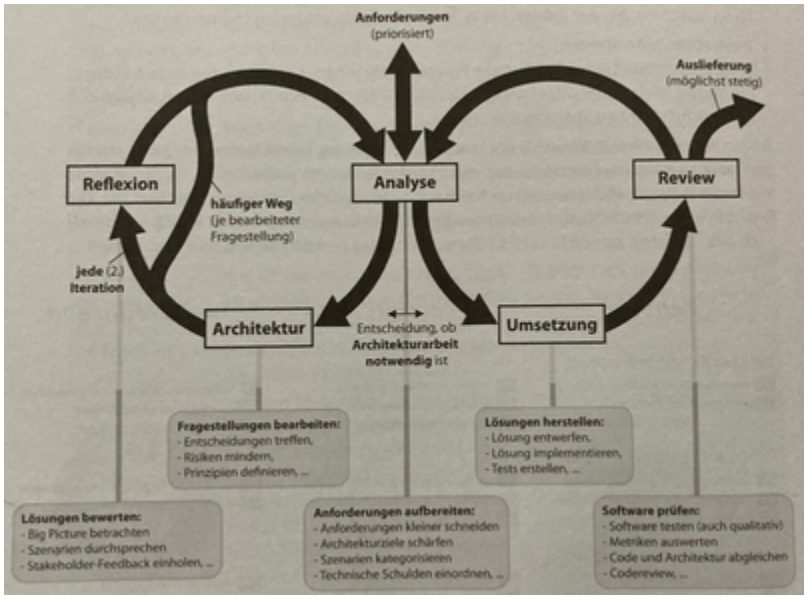

Abbildung: "Architekturbrezel"
(Quelle: Stefan Toth, "Vorgehensmuster für Softwarearchitektur. Kombinierbare Praktiken in Zeiten von Agile und Lean")

2. Die Architektur muss hinsichtlich Qualitätsattributen optimiert werden - abhängig von den Anforderungen

Anforderungen sind typischerweise Qualitätsattribute - oder lassen sich auf diese zurückführen, wie weiter unten (und im Rest der FLEX-Schulung) ausführlich beschrieben wird.

Solche Anforderungen müssen spezifiziert werden - je präziser, desto besser. Je messbarer, desto besser. Es ist eine der - wenn nicht die - wichtigste(n) Aufgabe(n) eines Architekten, relevante Anforderungen und Informationen zu sammeln, zu konsolidieren, zu verstehen und diese im architektonischen Entwurf umzusetzen.

Manchmal sind Anforderungen nicht einfach zu messen oder zu vage wie z.B. Verfügbarkeit, Zuverlässigkeit, Time-to-Market, Flexibilität, Vorhersagbarkeit, Reproduzierbarkeit, Homogenisierung der Stages, Internet/Web-Scale, verteilte Systeme, Parallelisierbarkeit der Feature-Entwicklung, Evolution der Architektur (Build for Replacement), Heterogenität, Automatisierbarkeit.

Was diese Begriffe genau bedeuten und wie sie in der Software und ihrer Architektur angegangen werden, ist aber entscheidend. Präzisierung über Qualitätsmerkmale ist nötig, siehe unten in Kapitel 2, Abschnitt (B).

3. Typische Tradeoffs

Architektonische Tradeoffs und Kompromisse beziehen sich auf die Entscheidungen, die während des Entwurfs und der Entwicklung einer Softwarearchitektur getroffen werden, um konkurrierende Ziele und Beschränkungen auszugleichen. Bei diesen Kompromissen geht es darum, zwischen verschiedenen Designoptionen zu wählen, die unterschiedliche Vor- und Nachteile haben. Einige häufige architektonische Kompromisse sind:

- **Flexibilität vs. Einfachheit**: Ein sehr flexibles System kann komplexer und schwieriger zu verstehen und zu warten sein als ein "einfaches" System, das z.B. als Monolith ohne Architekturprinzipien und Trennung von Zuständigkeiten implementiert ist. Andererseits ist ein solches "einfaches" System möglicherweise nicht in der Lage, alle Anforderungen bzw. wechselnde Bedürfnisse ausreichend gut zu erfüllen.
- **Security vs. Benutzerfreundlichkeit**: Die Verbesserung der Sicherheit kann dazu führen, dass ein System weniger gut benutzbar ist, und umgekehrt. So kann beispielsweise das Hinzufügen zusätzlicher Authentifizierungs- oder Verschlüsselungsfunktionen ein System zwar sicherer, aber auch weniger benutzerfreundlich machen.
- **Kosten vs. Qualität**: Manchmal stehen Kosten und Qualität in einem umgekehrten Verhältnis zueinander. Die Implementierung hochwertiger Funktionen oder die Verwendung teurer Technologien kann die Kosten des Projekts steigern., umgekehrt aber notwendig sein, um die geforderte Qualität der Software zu erreichen oder sie zu erhöhen.
- **Anpassbar vs. von der Stange** ("Customizable vs off-the-shelf"): Die Verwendung von Off-the-Shelf-Komponenten kann Zeit und Geld sparen, aber die Flexibilität des Systems einschränken. Eine maßgeschneiderte Lösung ist zwar flexibler, kann aber auch langfristig teurer und zeitaufwändiger sein.

Diese Kompromisse sind oft ein Balanceakt zwischen den Zielen der Software, den Rahmenbedingungen und Beschränkungen von Produkt oder Projekt und den Bedürfnissen der Stakeholder.

4. Schlüsselprinzip 1: Aufteilung in kleinere Teile - Die Notwendigkeit von Modularisierung, Partitionierung und Entkopplung auf allen Ebenen

Die Zerlegung eines Problems in kleinere Teile, auch bekannt als Dekomposition (auch Partitionierung, Teile&Herrsche, Divide&Conquer), ist ein wesentlicher Aspekt von Softwareentwicklung und -architektur, da sie dazu beitragen kann, die Komplexität zu bewältigen und den Gesamtentwurf des Systems zu verbessern. Durch die Aufteilung eines großen Problems in kleinere, besser handhabbare Teile können die Entwickler das Problem besser verstehen und lösen und ein besser wartbares und flexibleres System schaffen.

Die Aufteilung eines Systems in kleinere, unabhängige Komponenten oder Module ist entscheidend: Durch eine solche Partitionierung können Entwickler ein modulare(re)s Design erstellen, das einfach(er) zu verstehen, zu testen und zu warten ist. Die Partitionierung kann auch zur Verbesserung der Skalierbarkeit beitragen, da bestimmte Teile des Systems unabhängig voneinander skaliert werden können. Die Entkopplung ist die zentrale Voraussetzung für die Erfüllung dieser Anforderung.

Gründe, warum Dekomposition in der Softwareentwicklung und -architektur wichtig ist:

- **Modularität** und Modularisierung: Die Dekomposition ermöglicht es Entwicklern, modulare, einzelne Komponenten zu erstellen, die in verschiedenen Teilen des Systems wiederverwendet werden können. Dies erhöht die Flexibilität und Wartbarkeit des Systems und verkürzt die Gesamtentwicklungszeit.
- Leichter **verständlich:** Durch die Aufteilung eines großen Problems in kleinere Teile wird es für die Entwickler einfacher, das Problem und die Lösung zu verstehen. Jeder kleinere Teil kann unabhängig untersucht, entworfen und implementiert werden.
- Leichter zu **testen:** Wenn ein Problem in kleinere Komponenten aufgeteilt wird, ist es einfacher, jede Komponente einzeln zu testen, wodurch Fehler schneller erkannt und behoben werden können.
- **Skalierbarkeit:** Dekomposition kann dazu beitragen, ein System skalierbar zu machen, wenn so Entwickler neue Funktionen hinzufügen oder bestimmte Teile des Systems unabhängig voneinander skalieren können.
- Klare **Verantwortlichkeiten** und Scoping: Die Zerlegung eines Problems in kleinere Teile trägt dazu bei, die Verantwortung jeder Komponente klar zu definieren, was die Identifizierung und Behebung von Fehlern erleichtern und die Gesamtqualität des Systems verbessern kann.
- **Deployment** und Releases kleinerer, voneinander unabhängiger Einheiten sind möglich, der Release-Prozesses wird flexibler.

Dekomposition ist das Kernprinzip einer stabilen und handhabbaren Architektur, bei der die Funktionalität in kleine, unabhängige Teile (Komponenten, Dienste usw.) aufgeteilt wird, die unabhängig voneinander entwickelt, eingesetzt und skaliert werden können.

Es ist wichtig zu beachten, dass die Dekomposition immer einen Balanceakt darstellt: Eine zu starke Dekomposition führt i.d.R. zu erhöhter Komplexität und erhöhtem Koordinationsaufwand, während eine zu geringe Dekomposition dazu führt, dass das System schwer zu verstehen, zu testen und weiterzuentwickeln ist.

5. Schlüsselprinzip 2: Frühzeitige und reproduzierbare Testergebnisse - Der Bedarf für (Test-)Automatisierung und einfachen und immer-gleichen Setups

Frühzeitiges und reproduzierbares Testen ist bei der Software-Entwicklung und -Architektur hilfreich, da es wertvolle Informationen über die Funktionalität und Qualität des Systems liefert und dabei hilft, Fehler frühzeitig im Entwicklungsprozess zu erkennen und zu beheben.

- Durch **frühzeitiges Testen** erkennen und beheben Entwickler Fehler frühzeitig, bevor sie in späteren Phasen schwieriger und teurer zu beheben sind. Dies trägt dazu bei, die Gesamtqualität des Systems zu verbessern und verringert das Risiko, dass bei Änderungen am System neue Fehler entstehen.
- Die **Reproduzierbarkeit** ist beim Testen von Software essentiell, denn nur sie garantiert Entwicklern, Fehler und Testergebnisse nachzustellen und sicherzustellen, dass ein Fehler behoben ist und das System korrekt funktioniert. Außerdem können die Entwickler so das System unter verschiedenen Bedingungen testen und sicherstellen, dass es sich wie erwartet verhält. Insbesondere in einem verteilten System sind integrative Tests nur schwer und aufwändig reproduzierbar umzusetzen (siehe Kapitel 4, Abschnitt (I)).
- Die **Automatisierung** von Tests hilft enorm, um sicherzustellen, dass die Tests schnell und mit möglichst wenigen manuellen Eingriffen durchgeführt werden können. Die Automatisierung ermöglicht, Tests mehrfach, wiederholbar und in verschiedenen Umgebungen auszuführen, um insgesamt sicherzustellen, dass das System korrekt funktioniert. Außerdem können die Entwickler so Tests in verschiedenen Phasen des Entwicklungsprozesses durchführen, z. B. wenn neuer Code übergeben wird oder neue Funktionen hinzugefügt werden.

- Einfache und gleiche (oder zumindest ähnliche) **Test-Setups** sind wichtig, um die Durchführung von Tests zu erleichtern und so sicherzustellen, dass die Tests in gleichen/ähnlichen Umgebungen durchgeführt werden wie die, in denen das System eingesetzt werden soll (Beispiel: Prä-Produktionsumgebung - hinsichtlich Verhalten, Sizing der Umgebung, Verhalten, Konfigurationen). Automatisierte, leicht reproduzierbare und ähnliche Test-Setups ermöglichen, Probleme frühzeitig und mit minimalem Aufwand zu erkennen und sicherzustellen, dass das System korrekt funktioniert und in der Produktion zuverlässig ist.

(B) ISO 25010: Definition der Qualitätseigenschaften

1. Warum Qualitätseigenschaften?

Der Entwurf einer Software-Architektur ist kein Selbstzweck. Sie muss stattdessen Qualitätseigenschaften (veralteter Begriff: nicht-funktionale Anforderungen) umsetzen. Es gibt verschiedene Quallitätsmodelle wie z.B. die ISO-Norm 25010, die solche Qualitätseigenschaften kategorisieren und herunterbrechen auf standardisierte Begriffe.

Damit wird ein Standard etabliert, mit dem Begriffe wie "Sicherheit", "Performance", "Wartbarkeit" etc. bestmöglich - wenn auch nicht ideal - in eine klar definierte und im besten Falle messbare Form gebracht werden.

Eine "flexible Architektur" muss diese Qualitätseigenschaften beachten und Trade-offs zwischen ihnen priorisiert umsetzen. Eine konkrete Messbarkeit kann z.B. über Qualitätsszenarien erfolgen.

2. Qualitätsmodell der ISO 25010

Info
https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

https://de.wikipedia.org/wiki/ISO/IEC_9126

ISO/IEC 25010 ist eine internationale Norm, die einen gemeinsamen Rahmen für die Beschreibung und Spezifikation der Qualitätseigenschaften (-merkmale, -attribute) von Softwaresystemen bietet.

Die ISO-Norm definiert somit eine Reihe von Qualitätseigenschaften und Untereigenschaften, die üblicherweise zur Bewertung der Qualität von Software verwendet werden, einschließlich Funktionalität, Leistung, Benutzerfreundlichkeit, Zuverlässigkeit und Sicherheit.

3. Überblick über Qualitätseigenschaften, aus der ISO-Norm (Englisch)

SOFTWARE PRODUCT QUALITY								
FUNCTIONAL SUITABILITY	PERFORMANCE EFFICIENCY	COMPATIBILITY	INTERACTION CAPABILITY	RELIABILITY	SECURITY	MAINTAINABILITY	FLEXIBILITY	SAFETY
FUNCTIONAL COMPLETENESS FUNCTIONAL CORRECTNESS FUNCTIONAL APPROPRIATENESS iso25000.com	TIME BEHAVIOUR RESOURCE UTILIZATION CAPACITY	CO-EXISTENCE INTEROPERABILITY	APPROPRIATENESS RECOGNIZABILITY LEARNABILITY OPERABILITY USER ERROR PROTECTION USER ENGAGEMENT INCLUSIVITY USER ASSISTANCE SELF-DESCRIPTIVENESS	FAULTLESSNESS AVAILABILITY FAULT TOLERANCE RECOVERABILITY	CONFIDENTIALITY INTEGRITY NON-REPUDIATION ACCOUNTABILITY AUTHENTICITY RESISTANCE	MODULARITY REUSABILITY ANALYSABILITY MODIFIABILITY TESTABILITY	ADAPTABILITY SCALABILITY INSTALLABILITY REPLACEABILITY	OPERATIONAL CONSTRAINT RISK IDENTIFICATION FAIL SAFE HAZARD WARNING SAFE INTEGRATION

Abbildung: ISO25010 , 2023 - Norm für Qualitätseigenschaften, EN
(Quelle: https://iso25000.com/images/figures/iso_25010_en.png)

Quality Attribute, aus ISO25010 (EN), 2023	**Qualitätseigenschaft** **(DE, eigene Übersetzung)**	**ISO25010 Description (EN)**	**Typ** **Ausführungsqualität (zur Laufzeit beobachtbar), "execution quality"** **Evolutionsqualität (manifestiert in der Struktur), "evolution quality"**
Functional Suitability	**Funktionalität**	**This characteristic represents the degree to which a product or system provides functions that meet stated and implied needs when used under specified conditions.**	
Functional completeness	Funktionale Vollständigkeit	Degree to which the set of functions covers all the specified tasks and user objectives.	Ausführungsqualität
Functional correctness	Funktionale Korrektheit, Richtigkeit	Degree to which a product or system provides the correct results with the needed degree of precision.	Ausführungsqualität
Functional appropriateness	Funktionale Angemessenheit	Degree to which the functions facilitate the accomplishment of specified tasks and objectives.	Ausführungsqualität
Performance efficiency	**Performanz, Effizienz**	**This characteristic represents the performance relative to the amount of resources used under stated conditions.**	
Time behaviour	Zeitverhalten	Degree to which the response and processing times and throughput rates of a product or system, when performing its functions, meet requirements.	Ausführungsqualität
Resource utilization	Verbrauchsverhalten	Degree to which the amounts and types of resources used by a product or system, when performing its functions, meet requirements.	Ausführungsqualität
Capacity	Leistungsfähigkeit	Degree to which the maximum limits of a product or system parameter meet requirements.	Ausführungsqualität
Compatibility	**Kompatibilität**	**Degree to which a product, system or component can exchange information with other products, systems or components, and/or perform its required functions while sharing the same hardware or software environment.**	
Co-existence	Ko-Existenz	Degree to which a product can perform its required functions efficiently while sharing a common environment and resources with other products, without detrimental impact on any other product.	Ausführungsqualität
Interoperability	Interoperabilität	Degree to which two or more systems, products or components can exchange information and use the information that has been exchanged.	Ausführungsqualität
Interaction capability		**Degree to which a product or system can be interacted with by specified users to exchange information ia the user interface to complete specific tasks in a variety of contexts of use.**	
Appropriateness recognizability	Erkennbarkeit der Angemessenheit	Appropriateness recognizability - Degree to which users can recognize whether a product or system is appropriate for their needs.	Ausführungsqualität
Learnability	Erlernbarkeit	Learnability - Degree to which the functions of a product or system can be learnt to be used by specified users within a specified amount of time.	Ausführungsqualität
Operability	Bedienbarkeit	Operability - Degree to which a product or system has attributes that make it easy to operate and control.	Ausführungsqualität
User error protection	Schutz vor Fehlbedienung	User error protection. Degree to which a system prevents users against operation errors.	Ausführungsqualität
User engagement	Benutzerengagement	Degree to which a user interface presents functions and information in an inviting and motivating manner encouraging continued interaction.	Ausführungsqualität
Inclusivity	Inklusivität	Degree to which a product or system can be used by people of various backgrounds (such as people of various ages, abilities, cultures, ethnicities, languages, genders, economic situations, etc.).	Ausführungsqualität
User assistance	Benutzerunterstützung	Degree to which a product can be used by people with the widest range of characteristics and capabilities to achieve specified goals in a specified context of use.	Ausführungsqualität
Self-descriptiveness	Selbstbeschreibungsfähigkeit	Degree to wich a product presents appropriate information, where needed by the user, to make its capabilities and use immediately obvious to the user without excessive interactions with a product or other resources (such as user documentation, help desks or	Ausführungsqualität

		other users).	
Reliability	**Zuverlässigkeit**	**Degree to which a system, product or component performs specified functions under specified conditions for a specified period of time.**	
Faultnessless	Fehlerfreiheit	Degree to which a system, product or component meets needs for reliability under normal operation.	Ausführungsqualität
Availability	Verfügbarkeit	Degree to which a system, product or component is operational and accessible when required for use.	Ausführungsqualität
Fault tolerance	Fehlertoleranz	Degree to which a system, product or component operates as intended despite the presence of hardware or software faults.	Ausführungsqualität
Recoverability	Wiederherstellbarkeit	Degree to which, in the event of an interruption or a failure, a product or system can recover the data directly affected and re-establish the desired state of the system.	Ausführungsqualität
Security	**Informationssicherheit**	**Degree to which a product or system protects information and data so that persons or other products or systems have the degree of data access appropriate to their types and levels of authorization.**	
Confidentiality	Vertraulichkeit	Degree to which a product or system ensures that data are accessible only to those authorized to have access.	Ausführungsqualität
Integrity	Integrität	Degree to which a system, product or component prevents unauthorized access to, or modification of, computer programs or data.	Ausführungsqualität
Non-repudiation	Nichtabstreitbarkeit	Degree to which actions or events can be proven to have taken place so that the events or actions cannot be repudiated later.	Ausführungsqualität
Accountability	Verfolgbarkeit	Degree to which the actions of an entity can be traced uniquely to the entity.	Ausführungsqualität
Authenticity	Authentizität, Echtheit, Glaubwürdigkeit	Degree to which the identity of a subject or resource can be proved to be the one claimed.	Ausführungsqualität
Resistance	Widerstand	Degree to which the product or system sustains operations while under attack from a malicious actor.	Ausführungsqualität
Maintainability	**Änderbarkeit**	**This characteristic represents the degree of effectiveness and efficiency with which a product or system can be modified to improve it, correct it or adapt it to changes in environment, and in requirements.**	
Modularity	Modularität	Degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components.	Evolutionsqualität
Reusability	Wiederverwendbarkeit	Degree to which an asset can be used in more than one system, or in building other assets.	Evolutionsqualität
Analysability	Analysierbarkeit	Degree of effectiveness and efficiency with which it is possible to assess the impact on a product or system of an intended change to one or more of its parts, or to diagnose a product for deficiencies or causes of failures, or to identify parts to be modified.	Evolutionsqualität
Modifiability	Modifizierbarkeit	Degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading existing product quality.	Evolutionsqualität
Testability	Prüfbarkeit, Testbarkeit	Degree of effectiveness and efficiency with which test criteria can be established for a system, product or component and tests can be performed to determine whether those criteria have been met.	Ausführungsqualität / Evolutionsqualität
Flexibility	**Flexibilität**	**Degree of effectiveness and efficiency with which a system, product or component can be transferred from one hardware, software or other operational or usage environment to another.**	
Adaptability	Anpassbarkeit	Degree to which a product or system can effectively and efficiently be adapted for different or evolving hardware, software or other operational or usage environments.	Ausführungsqualität / Evolutionsqualität
Scalability	Skalierbarkeit	Degree to which a product can handle growing or shrinking workloads or to adapt its capacity to handle variability.	Ausführungsqualität
Installability	Installierbarkeit	Degree of effectiveness and efficiency with which a product or system can be successfully installed and/or uninstalled in a specified environment.	Ausführungsqualität
Replaceability	Austauschbarkeit	Degree to which a product can replace another specified software product for the same purpose in the same environment.	Ausführungsqualität
Safety	**Betriebssicherheit**	**This characteristic represents the degree to which a product under defined conditions to avoid a state in which human life, health, property, or the environment is endangered.**	
Operational Constraint	Betriebseinschränkung	Degree to which a product or system constrains its operation to within safe parameters or states when encountering operational hazard.	Ausführungsqualität

Risk Identification	Risiko-Identifikation	Degree to which a product can identify a course of events or operations that can expose life, property or environment to unacceptable risk.	Ausführungsqualität
Fail Safe	Fehlertoleranz	Degree to which a product can automatically place itself in a safe operating mode, or to revert to a safe condition in the event of a failure.	Ausführungsqualität
Hazard Warning	Risiko- oder Gefahrenhinweis	Degree to which a product or system provides warnings of unacceptable risks to operations or internal controls so that they can react in sufficient time to sustain safe operations.	Ausführungsqualität
Safe Integration	Sicherheitsgerechte Integration	Degree to which a product can maintain safety during and after integration with one or more components.	Ausführungsqualität

4. ISO-25010 - Update von 2023

Gernot Starke: "Shortcomings of ISO 25010", https://www.innoq.com/en/articles/2023/02/iso-25010-shortcomings/

Die alte ISO-Norm von 2011 (hatte damals ISO 9126 ersetzt) hat einige Schwächen, die oft kritisiert werden ("zu sperrig", "da fehlen relevante Eigenschaften" wie Safety, "strikte Hierarchie ist fraglich", "praktischer Nutzen fraglich", "wie kann man das konstruktiv nutzen?", siehe Gernot Starkes Blogbeitrag).

Mit der neuen ISO-25010 von November 2023 sind einige Verbesserungen und Änderungen ggü. 2011 dazugekommen, darunter:

a) Neue Hauptkategorie "Safety" wurde aufgenommen:

- "Safety" meint Betriebssicherheit, d.h. den Schutz von Mensch und Umwelt vor physischem Schaden.
- "Security" betriebt die Informationssicherheit und damit in erster Linie den Schutz der Daten vor unberechtigtem Zugriff und Datenmissbrauch.

b) "Flexibility" (Flexibilität) hat "Portability" (Übertragbarkeit) ersetzt.

c) "Scalability" (Skalierbarkeit) wurde neu aufgenommen, bei "Flexibility".

5. Qualitätsszenarien

Info
Rick Kazman, Paul Clements, Len Bass Software Architecture in Practice, Third Edition

Qualitätsmodell (und Beispiel-Szenarien) von arc42: https://quality.arc42.org/

Software Architektur im Stream, Video zu "Qualitätsszenarien"

Mit **Qualitätsszenarien** lassen sich die (sonst abstrakten und schwammigen) Qualitätsanforderungen gut beschreiben und machen die Anforderungen verfolgbar und messbar. Ein Qualitätsszenario besteht aus

- Quelle des Stimulus, und Stimulus
- betroffenes Artefakt
- dessen Antwort, und Metrik für die Antwort
- Kontext / Umgebung

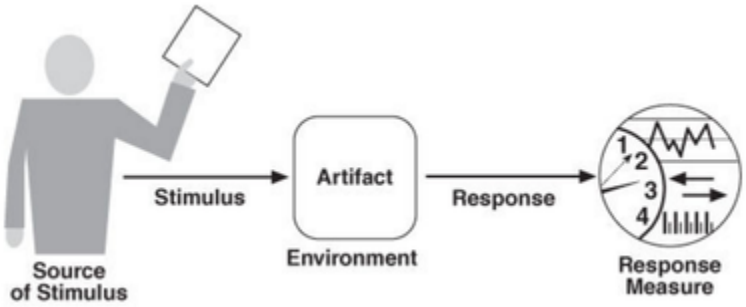

Abbildung: Qualitätsszenario
(Quelle: https://esb-dev.github.io/mat/saa-qua-bh.pdf , Folie 14)

Beispiel, verkürzt (von https://quality.arc42.org/)

- Stimulus: An authenticated user requests generation of the daily sales report in PDF format via the graphical user interface.
- Metrik: The system generates this report in less than 10 seconds.