9 Nichtfunktionale Anforderungen

Ziel jeder Softwareentwicklung ist es, die Anforderungen des Auftraggebers bzw. des Marktes an das Softwareprodukt zu erfüllen. Es werden funktionale und nichtfunktionale Anforderungen unterschieden.

Funktional

Funktionale Anforderungen (*functional requirements*, kurz FRs) spezifizieren, welche Funktionalität oder welches Verhalten das Softwareprodukt unter festgelegten Bedingungen besitzen bzw. erfüllen soll.

Nichtfunktional

Nichtfunktionale Anforderungen (*nonfunctional requirements*, kurz NFRs), auch Technische Anforderungen genannt, beschreiben Aspekte, die typischerweise mehrere oder alle funktionalen Anforderungen betreffen bzw. überschneiden *(cross-cut)*. Sie haben in der Regel einen Einfluss auf die gesamte Softwarearchitektur. Außerdem beeinflussen sich nichtfunktionale Anforderungen gegenseitig.

Beispiel

Sicherheit *(security)* steht häufig im Konflikt mit Benutzbarkeit, Speichereffizienz ist meist gegenläufig zur Laufzeiteffizienz.

Für manche nichtfunktionale Anforderungen kann auch eine Dienstgüte – *Quality of Service* (QoS) genannt – festgelegt werden, z. B. bezogen auf Leistung und Effizienz.

Nichtfunktionale Anforderungen können sich auf verschiedene Bausteinarten beziehen.

Beispiel

Die Effizienz des Softwaresystems ergibt sich aus dem komplexen Zusammenspiel verschiedener Komponenten. Die Zuverlässigkeit und teilweise auch die Änderbarkeit werden durch die einzelnen Komponenten bestimmt.

Vereinfacht ausgedrückt, legen funktionale Anforderungen fest, *was* das Softwareprodukt tun soll, während die nichtfunktionalen Anforderungen spezifizieren, *wie* es arbeiten soll.

Abgrenzung

Die Abgrenzung zwischen funktionalen und nichtfunktionalen Anforderungen ist oft nicht einfach. Beispielsweise können Zeitanforderungen als Verhalten des Softwareproduktes betrachtet oder auch als nichtfunktionale Anforderung aufgefasst werden.

Beispiel

Oft hängt die Unterscheidung davon ab, wie die Anforderung formuliert ist [Glin07, S. 23]:

H. Balzert, *Lehrbuch der Softwaretechnik*, DOI 10.1007/978-3-8274-2246-0_9,
© Spektrum Akademischer Verlag Heidelberg 2011

»Das System soll einen unautorisierten Zugriff auf die Kundendaten verhindern.« Diese Formulierung spricht für eine nichtfunktionale Anforderung.
»Die Datenbank soll den Zugriff auf die Kundendaten nur den Benutzern gestatten, die sich durch den Benutzernamen und das Passwort autorisiert haben.« Diese Formulierung spricht für eine funktionale Anforderung.

In [Glin07] wird daher vorgeschlagen, *nicht* zwischen funktionalen und nichtfunktionalen Anforderungen zu unterscheiden, sondern nach folgenden vier Aspekten zu spezifizieren:

- Beschreibungsart: Operational, qualitativ, quantitativ, deklarativ
- Art der Anforderung: Funktionen, Daten, Leistung, spezifische Qualität, Randbedingung
- Erfüllungsgrad: »Harte« oder »weiche« Anforderungen
- Rolle der Anforderung: präskriptiv, normativ, *assumptive*

ISO/IEC 9126

In dem häufig zitierten internationalen Standard ISO/IEC 9126 werden folgende Qualitätscharakteristika mit ihren Teilmerkmalen spezifiziert (ohne Rangfolge):

- **Funktionalität** *(functionality)*
  - Angemessenheit *(suitability)*
  - Genauigkeit *(accuracy)*
  - Interoperabilität *(interoperability)*
  - Sicherheit *(security)*
  - Konformität der Funktionalität *(functionality compliance)*
- **Zuverlässigkeit** *(reliability)*
  - Reife *(maturity)*
  - Fehlertoleranz *(fault tolerance)*
  - Wiederherstellbarkeit *(recoverability)*
  - Konformität der Zuverlässigkeit *(reliability compliance)*
- **Benutzbarkeit, Gebrauchstauglichkeit** *(usability)*
  - Verständlichkeit *(understandability)*
  - Erlernbarkeit *(learnability)*
  - Bedienbarkeit *(operability)*
  - Attraktivität *(attractiveness)*
  - Konformität der Benutzbarkeit *(usability compliance)*
- **Effizienz** *(efficiency)*
  - Zeitverhalten *(time behaviour)*
  - Verbrauchsverhalten *(resource utilisation)*
  - Konformität der Effizienz *(efficiency compliance)*
- **Wartbarkeit** *(maintainability)*
  - Analysierbarkeit *(analyzability)*
  - Änderbarkeit *(changeability)*
  - Stabilität *(stability)*
  - Testbarkeit *(testability)*
  - Konformität der Wartbarkeit *(maintainability compliance)*

- **Portabilität** *(portability)*
  - Anpassbarkeit *(adaptability)*
  - Installierbarkeit *(installability)*
  - Koexistenz *(co-existence)*
  - Austauschbarkeit *(replaceability)*
  - Konformität der Portabilität *(portability compliance)*

NFRs

Obwohl seit über 30 Jahren nichtfunktionale Anforderungen postuliert werden, gibt es bis heute keinen Konsens über die wichtigsten nichtfunktionalen Anforderungen, ihre Definition und ihre Untergliederung. In [MZN10] wurden 182 Quellen der letzten 30 Jahre zu nichtfunktionalen Anforderungen ausgewertet. Es wurden 252 nichtfunktionale Anforderungen identifiziert, davon 114 mit einem Bezug zu Qualität. Die am meisten genannten nichtfunktionalen Anforderungen sind (Top 5):

Top 5

1. **Leistung** *(performance)* (89 %)
2. **Zuverlässigkeit** *(reliability)* (68 %)
3. **Benutzbarkeit, Gebrauchstauglichkeit** *(usability)* (62 %)
4. **Sicherheit** *(security)* (60 %)
5. **Wartbarkeit** *(maintainability)* (55 %)

Auch die Mehrbenutzerfähigkeit oder die Verteilbarkeit einer Anwendung auf mehrere, miteinander vernetzte Computersysteme könnte man als nichtfunktionale Anforderungen ansehen. Da diese Anforderungen in der Literatur aber *nicht* unter nichtfunktionalen Anforderungen aufgeführt werden, werden sie hier unter Einflussfaktoren beschrieben (siehe »Einflussfaktoren auf die Architektur«, S. 135).

NFRs & Systemtypen

In [MZN10, S. 315] werden die fünf Systemtypen Echtzeitsysteme, sicherheitskritische Systeme, Websysteme, Informationssysteme und Prozesssteuerungssysteme mit ihren relevanten nichtfunktionalen Anforderungen identifiziert (Abb. 9.0-1). In allen Systemtypen werden Leistung, Sicherheit und Benutzbarkeit als relevant angesehen.

Die Zuordnung von relevanten nichtfunktionalen Anforderungen zu Anwendungsdomänen zeigt die Tab. 9.0-1 [MZN10, S. 316].

Hinweis

Anhand der Abb. 9.0-1 und der Tab. 9.0-1 können Sie feststellen, welche nichtfunktionalen Anforderungen für Ihren Systemtyp und Ihre Anwendungsdomäne relevant sind.

Nichtfunktionale Anforderungen lassen sich nach verschiedenen Kriterien klassifizieren.

In [Glin07, S. 25] erfolgt eine Klassifizierung in Leistungs-Anforderungen, spezifische Qualitäts-Anforderungen und Randbedingungen *(constraints)*:

Klassifizierung 1

- Leistungs-Anforderungen: Zeit- und Speicherbegrenzungen wie Zeit, Geschwindigkeit, Volumen, Durchsatz

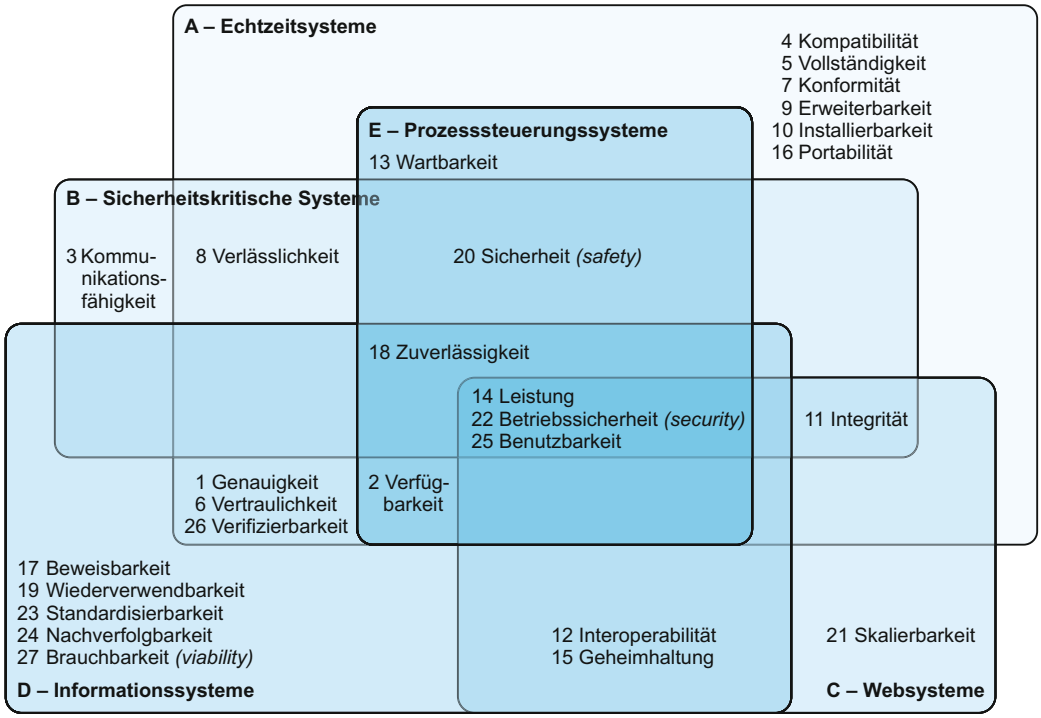

Abb. 9.0-1: Zuordnung von nichtfunktionalen Anforderungen zu Systemtypen.

- Qualitäts-Anforderungen: »-keiten« wie Zuverlässigkeit, Benutzbarkeit, Sicherheit, Wartbarkeit, Verfügbarkeit, Portabilität
- Randbedingungen: Physik, Recht, Kultur, Umgebung, Entwurfs- und Implementierungsschnittstellen usw.

In [SoAr] wird folgende Klassifizierung vorgenommen:

Klassifizierung 2

- Laufzeit-Anforderungen: Dazu zählen alle Anforderungen, die zur Laufzeit gemessen werden können wie Funktionalität, Leistungen, Sicherheit, Benutzbarkeit, Verfügbarkeit, Interoperabilität.
- Nicht-Laufzeit-Anforderungen: Dazu zählen alle Anforderungen, die nicht zur Laufzeit gemessen werden können wie Modifizierbarkeit, Portabilität, Wiederverwendbarkeit, Integrierbarkeit, Testbarkeit.
- Architektur-Anforderungen: Anforderungen, die spezifisch für die Architektur selbst sind wie konzeptionelle Integrität, Korrektheit.
- Domänenspezifische Anforderungen: Anforderungen für spezifische Domänen wie Sensitivität, Kalibrierbarkeit.

In [MaBr01] werden zwei Klassen unterschieden:

Klassifizierung 3

- Laufzeit-Anforderungen: Benutzbarkeit, Leistung, Sicherheit, Konfigurierbarkeit, Korrektheit, Zuverlässigkeit, Verfügbarkeit, Skalierbarkeit.

Anwendungsdomäne	relevante NFRs
Banken und Finanzen	Genauigkeit, Geheimhaltung, Leistung, Sicherheit, Benutzbarkeit
Ausbildung	Interoperabilität, Leistung, Zuverlässigkeit, Skalierbarkeit, Sicherheit, Benutzbarkeit
Energieressourcen	Verfügbarkeit, Leistung, Zuverlässigkeit, Betriebssicherheit, Benutzbarkeit
Regierung und Militär	Genauigkeit, Vertraulichkeit, Leistung, Geheimhaltung, Beweisbarkeit, Wiederverwendbarkeit, Sicherheit, Standardisierbarkeit, Benutzbarkeit, Verifizierbarkeit, Brauchbarkeit
Versicherungen	Genauigkeit, Vertraulichkeit, Integrität, Interoperabilität, Sicherheit, Benutzbarkeit
Medizin, Gesundheitswesen	Kommunikationsfähigkeit, Vertraulichkeit, Integrität, Leistung, Geheimhaltung, Zuverlässigkeit, Sicherheit, Betriebssicherheit, Nachverfolgbarkeit, Benutzbarkeit
Telekommunikation	Kompatibilität, Konformität, Verlässlichkeit, Installierbarkeit, Wartbarkeit, Leistung, Portabilität, Zuverlässigkeit, Benutzbarkeit
Verkehrswesen	Genauigkeit, Verfügbarkeit, Kompatibilität, Vollständigkeit, Vertraulichkeit, Verlässlichkeit, Integrität, Leistung, Sicherheit, Betriebssicherheit, Verifizierbarkeit

Tab. 9.0-1: NFRs & Anwendungsdomänen.

- Entwicklungszeit-Anforderungen: Weiterentwickelbarkeit, Wiederverwendbarkeit, Modifizierbarkeit, Lokalisierbarkeit, Zusammensetzbarkeit.

Laufzeit-Anforderungen sind wertvoll für den Benutzer und unterstützen die kurzfristige Wettbewerbsfähigkeit. Entwicklungszeit-Anforderungen erhöhen in der Regel den Wert des Softwareprodukts und sorgen für die langfristige Wettbewerbsfähigkeit.

Klassifizierung 4

Eine weitere Unterscheidungsmöglichkeit ist eine Klassifizierung in quantitative und qualitative nichtfunktionale Anforderungen. Zu den quantitativen, d. h. messbaren Anforderungen gehören z. B. Laufzeit-Anforderungen. Zu den qualitativen Anforderungen gehören z. B. die Benutzbarkeit und die Wartbarkeit, die nicht oder nur sehr aufwändig zu messen sind (siehe hierzu auch [Glin08]).

Klassifizierung 5

Bezogen auf den Softwarelebenszyklus lassen sich nichtfunktionale Anforderungen nach Ansicht des Autors auch wie folgt klassifizieren:

- Relevant während der Entwicklungszeit:
- Funktionalität
- Benutzbarkeit
- Sicherheit
- Relevant während der Installation:
- Funktionalität – Interoperabilität
- Portabilität – Anpassbarkeit
- Portabilität – Instalierbarkeit
- Portabilität-Koexistenz

- ■ Relevant während des Betriebs:
- □ Leistung/Effizienz
- □ Zuverlässigkeit
- □ Benutzbarkeit
- □ Sicherheit
- □ Wartbarkeit
- □ Weiterentwickelbarkeit

Relevant während der Entwicklungszeit bedeutet, dass ein System nicht abgenommen wird, wenn die angegebenen Anforderungen nicht erfüllt sind. Beispielsweise kann geprüft werden, ob die geforderte Funktionalität, Benutzbarkeit und Sicherheit vorhanden sind.

Relevant während der Installation bedeutet, dass die nichtfunktionalen Anforderungen während der Installation sichtbar werden.

Relevant während des Betriebs bedeutet, dass diese nichtfunktionalen Anforderungen bei der Abnahme des Systems in der Regel *nicht* getestet werden können, sondern dass der Grad ihrer Erfüllung oft erst während des Betriebs überprüft werden kann.

trade-offs

Nichtfunktionale Anforderungen können sich gegenseitig verstärken, es kann aber auch zu Zielkonflikten *(trade-offs)* kommen. Zielkonflikte können aber auch zwischen funktionalen und nichtfunktionalen Anforderungen auftreten. Die Tab. 9.0-2 zeigt, bezogen auf die wichtigsten Anforderungen, die gegenseitigen Abhängigkeiten (siehe auch [EgGr04], [Wieg03]).

↓Anforderung / Effekt→	Funktionalität +	Leistung / Effizienz +	Zuverlässigkeit +	Benutzbarkeit +	Sicherheit +	Wartbarkeit +
Funktionalität	**o**	–	–	**+**–	–	–
Leistung / Effizienz	–	**o**	–	**+**–	–	–
Zuverlässigkeit	–	–	**o**	**+**	**+**	**+**
Benutzbarkeit	**+**–	**+**–	**+**	**o**	–	**o**
Sicherheit	–	–	**+**	–	**o**	**+**
Wartbarkeit	–	–	**+**	**o**	**o**	**o**

Tab. 9.0-2: Gegenseitige Abhängigkeiten.

Die Tab. 9.0-2 ist wie folgt zu lesen: In den Zeilen stehen die Anforderungs-Attribute. Die Spalten repräsentieren die potenziellen Effekte auf die anderen Anforderungs-Attribute.

Fügt eine Anforderung eine zusätzliche Funktionalität hinzu, dann zeigt die erste Zeile, dass dies einen negativen Einfluss auf die Effizienz (–), Zuverlässigkeit (–), die Sicherheit (–) und die Wartbarkeit (–) hat. Ein positiver Aspekt ergibt sich für die Benutzbarkeit (+).

Ein umgekehrter Effekt entsteht, wenn eine Anforderung entfällt oder der Grad einer Anforderung reduziert wird. Wird beispielsweise die Sicherheit reduziert (Zeile 5), dann erhöht sich die Leistung/Effizienz.

Verfeinerung

Oft ist es erforderlich, Anforderungs-Attribute weiter zu untergliedern. Beispielsweise kann das Anforderungs-Attribut Leistung/Effizienz unterteilt werden in Laufzeit-Effizienz und Speicher-Effizienz.

Problem

Die Tab. 9.0-2 gibt Anhaltspunkte für mögliche positive oder negative gegenteilige Effekte. Die Abhängigkeiten müssen mit den konkreten Anforderungen überprüft werden. Es kann durchaus sein, dass zunächst offensichtliche Konflikte bei näherem Betrachten nicht bestehen, da sich die Anforderungen auf verschiedene Subsysteme beziehen.

Zunächst wird die »Wartbarkeit« näher betrachtet, da sie die Softwarearchitektur stark beeinflusst:

■ »Wartbarkeit«, S. 116

Für langlebige Softwaresysteme wird die »Weiterentwickelbarkeit« *(evolvability)* immer wichtiger. Sie wird daher nicht untergeordnet unter der Wartbarkeit behandelt, sondern separat:

■ »Weiterentwickelbarkeit«, S. 119

Geforderte Sicherheitsaspekte haben ebenfalls einen wesentlichen Einfluss auf die Softwarearchitektur:

■ »Betriebssicherheit und Funktionssicherheit«, S. 121

Die Sicherstellung der Zuverlässigkeit eines Softwaresystems erfordert besondere Maßnahmen:

■ »Zuverlässigkeit«, S. 124

Da die nichtfunktionalen Anforderungen »Leistung« und »Effizienz« oft synonym oder gegenseitig untergeordnet klassifiziert werden, werden sie im Folgenden gemeinsam behandelt:

■ »Leistung und Effizienz«, S. 128

Die Gebrauchstauglichkeit und Benutzbarkeit kann durch verschiedene architektonische Maßnahmen gefördert werden:

■ »Benutzbarkeit«, S. 130

Für die Installierbarkeit und die Weiterentwickelbarkeit eines Softwareprodukts spielt die Portabilität eine wichtige Rolle. Sie wird daher als eigenständige, nichtfunktionale Anforderung behandelt:

■ »Portabilität«, S. 132

Literatur

[MZN10], [Glin08], [Glin07], [SoAr], [MaBr01], [EgGr04], [Wieg03]

9.1 Wartbarkeit

Software ist auch nach der Inbetriebnahme ständigen Veränderungen unterworfen. Diese Veränderungen können sich durch die Umwelt des Softwaresystems ergeben, aber auch durch neue oder veränderte Anforderungen. Ziel beim Entwurf der Architektur muss es daher sein, Veränderungen auch in der Betriebsphase vorzunehmen, sodass die Einsatzbereitschaft des Softwaresystems gewährleistet bleibt. Änderungen sollen mit geringem Aufwand und in kurzer Zeit durchgeführt werden können. Wartungsaktivitäten umfassen meist kleine Änderungen und Fehlerbehebungen, weil Zeit- und Kostenanforderungen eine große Rolle spielen. Ziel der Wartbarkeit ist die Lebenserhaltung des Systems, *nicht* die Weiterentwickelbarkeit (siehe »Weiterentwickelbarkeit«, S. 119). Ein System, das *nicht* weiterentwickelt werden kann, wird zu einem Altsystem – in der Softwaretechnik *Legacy*-System genannt.

Die nichtfunktionale Anforderung **Wartbarkeit** *(maintainability)* wird in der Norm ISO/IEC 9126–1 als Qualitätsmerkmal eingeordnet und wie folgt definiert:

Definition

> Fähigkeit des Softwareprodukts änderungsfähig zu sein. Änderungen können Korrekturen, Verbesserungen oder Anpassungen der Software an Änderungen der Umgebung, der Anforderungen und der funktionalen Spezifikationen einschließen.

Wartbarkeit wird untergliedert in folgende Teilmerkmale:

- **Analysierbarkeit** *(analyzability)*
  Fähigkeit des Softwareprodukts, Mängel oder Ursachen von Versagen zu diagnostizieren oder änderungsbedürftige Teile zu identifizieren.
- **Änderbarkeit** *(changeability)*
  Fähigkeit des Softwareprodukts, die Implementierung einer spezifizierten Änderung zu ermöglichen.
- **Stabilität** *(stability)*
  Fähigkeit des Softwareprodukts, unerwartete Wirkungen von Änderungen der Software zu vermeiden.
- **Testbarkeit** *(testability)*
  Fähigkeit des Softwareprodukts, die modifizierte Software zu validieren.

Zu dem Teilmerkmal Analysierbarkeit gehört auch die Verständlichkeit und die Rückverfolgbarkeit *(traceability)*.

In [MZN10, S. 315] wird Wartbarkeit wie folgt definiert:

Anforderungen, die die Fähigkeit eines Softwareproduktes beschreiben, modifiziert zu werden. Modifizieren beinhaltet die Korrektur von Defekten oder die Vornahme von Verbesserungen oder Änderungen der Software.

Definition

Zu den Teilmerkmalen nach ISO/IEC 9126–1 kommen noch folgende Teilmerkmale hinzu:

- **Verständlichkeit** *(understandability)*
- **Modifizierbarkeit** *(modifiability)*

Die Wartbarkeit ist entscheidend in der Betriebsphase eines Systems.

Problem

Während der Betriebsphase durchgeführte Änderungen führen in der Regel zu einer Verschlechterung der Struktur – auch als Architekturerosion *(architectural decay, architectural drift)* bezeichnet. Dadurch werden weitere Änderungen erschwert oder verhindert. Durch *Reengineering* kann versucht werden, die Wartbarkeit zu verbessern.

↓Anforderung / Effekt→	Funktionalität +	Leistung / Effizienz +	Zuverlässigkeit +	Benutzbarkeit +	Sicherheit +	Wartbarkeit +
Wartbarkeit	–	–	**+**	**o**	**o**	**o**

Tab. 9.1-1: Gegenseitige Abhängigkeiten.

Wie die Tab. 9.1-1 zeigt, führt eine zunehmende Funktionalität und eine Steigerung der Leistung/Effizienz jeweils zu einer schlechteren Wartbarkeit. Wird die Zuverlässigkeit verbessert, dann verbessert sich auch die Wartbarkeit. Der Zusammenhang zwischen der Benutzbarkeit und der Wartbarkeit ist nicht eindeutig. Führt eine verbesserte Benutzbarkeit zu einem umfangreicheren Code, dann kann dies die Wartbarkeit verschlechtern. Reduziert sich dagegen durch eine verbesserte Benutzbarkeit der Quellcode, dann kann dies die Wartbarkeit verbessern. Analog gilt dies für den Gesichtspunkt der Sicherheit.

Die Wartbarkeit steht außerdem im Zusammenhang mit der nichtfunktionalen Anforderung **Weiterentwickelbarkeit** (siehe »Weiterentwickelbarkeit«, S. 119). Während die Wartbarkeit sich auf eine überschaubare Lebenszeit eines Softwaresystems bezieht, berücksichtigt die Weiterentwickelbarkeit auch sehr lange Lebenszeiten eines Softwaresystems.

Was ist zu tun?

Die Wartbarkeit kann durch die Einhaltung von folgenden Regeln verbessert werden:

Regeln

- Identifizieren Sie die Aspekte in Ihrem Entwurf, die sich ändern können, und trennen Sie diese Aspekte von denen, die gleich bleiben.

- Programmieren Sie gegen Abstraktionen und *nicht* gegen konkrete Klassen. Dadurch wird die Abhängigkeit von konkreten Klassen reduziert. Man spricht vom Prinzip der Abhängigkeitsumkehr *(dependency inversion principle).*
- Programmieren Sie gegen Schnittstellen *(interfaces)*. Durch den Polymorphismus funktioniert der Code auch mit jeder Klasse, die die entsprechende Schnittstelle implementiert (siehe »Schnittstellen, Fabriken und Komposition«, S. 503).

Prinzipien

Die Einhaltung folgender Prinzipien (siehe »Architekturprinzipien«, S. 29) unterstützen die Wartbarkeit:

- **Prinzip der Strukturierung**: Dadurch bessere Verständlichkeit und Analysierbarkeit.
- **Prinzip der Bindung und Kopplung**: Dadurch Begrenzung der Auswirkungen von Änderungen, d. h. bessere Stabilität.
- **Prinzip der Modularisierung**: Dadurch wird eine weitgehende Kontextunabhängigkeit von der Umgebung erreicht. Die Testbarkeit wird verbessert, da Module für sich testbar sind.
- **Geheimnisprinzip**: Änderungen an den Datenstrukturen wirken sich *nicht* auf die Zugriffsmethoden aus. Dadurch wird eine lokale Änderbarkeit ermöglicht.
- **Prinzip der Lokalität**: Da sich alle benötigten Informationen lokal an einer Stelle befinden, wird die Verständlichkeit und Analysierbarkeit wesentlich verbessert.
- **Prinzip der Verbalisierung**: Dadurch bessere Verständlichkeit und Analysierbarkeit.
- **Architekturprinzip: Konzeptionelle Integrität**: Dadurch schnellere Einarbeitung (siehe »Architekturprinzip: Konzeptionelle Integrität«, S. 30).
- **Architekturprinzip: Trennung von Zuständigkeiten**: Dadurch schnellere Einarbeitung und leichtere Analysierbarkeit (siehe »Architekturprinzip: Trennung von Zuständigkeiten«, S. 31).
- **Architekturprinzip: Sichtbarkeit**: Dadurch bessere Verständlichkeit und Analysierbarkeit (siehe »Architekturprinzip: Sichtbarkeit«, S. 34).

Architektur- & Entwurfs-Muster

Folgende Architektur- und Entwurfs-Muster unterstützen z. B. die Wartbarkeit:

- »Das Schichten-Muster *(layers pattern)*«, S. 46: Zur Entkopplung der Subsysteme.
- »Das Beobachter-Muster *(observer pattern)*«, S. 54: Zur Entkoppelung des Subsystems Benutzungsoberfläche von dem Subsystem Applikation.
- »Das MVC-Muster *(model view controller pattern)*«, S. 62: Zur Entkopplung der Subsysteme.
- »Das Proxy-Muster *(proxy pattern)*«, S. 83: Zum transparenten Zugriff auf entfernte Subsysteme (Remote-Proxy).

9.2 Weiterentwickelbarkeit

Während man früher dachte, Softwaresysteme leben im Einsatz 10 bis 20 Jahre, muss man heute feststellen, dass viele eingesetzte Softwaresysteme wesentlich länger in Betrieb sind. Wenn man heute davon ausgeht, dass neu entwickelte Systeme auch in Zukunft wesentlich länger eingesetzt werden, dann muss bereits bei der Softwarearchitektur dafür gesorgt werden, dass diese Systeme auch langfristig nutzbar sind. Das bedeutet, dass strukturelle Änderungen und die Erhaltung der Architekturintegrität berücksichtigt werden müssen. Folgender Begriff hat sich für diese Anforderungen gebildet:

Definition

Die nichtfunktionale Anforderung **Weiterentwickelbarkeit** *(evolvability)* – besser **Nachhaltigkeit** genannt – bezeichnet die Fähigkeit eines Softwareprodukts sich langfristig an geänderte Anforderungen und Techniken, die einen Einfluss auf die Architektur und/oder die funktionalen Erweiterungen haben, anzupassen, ohne die architektonische Integrität zu verletzen.

Die Weiterentwickelbarkeit lässt sich in folgende Merkmale untergliedern [RiBo09, S. 341]:

- Analysierbarkeit
- Änderbarkeit
- Testbarkeit
- Portabilität
- Erweiterbarkeit
- Integrität der Architektur

Weiterentwickelbarkeit bedeutet also, die Wartbarkeit langfristig zu erhalten. Nur so kann vermieden werden, dass ein Softwaresystem zu einem Altsystem *(legacy system)* wird – oft architektonischer Verfall genannt.

Weiterentwickelbarkeit erlaubt zusätzlich zur Wartung auch Modifikationen struktureller Art. Neue Techniken und Plattformen sowie neue nichtfunktionale Anforderungen, wie die Steigerung von Skalierbarkeit und die Einführung von Mehrsprachigkeit, werden ermöglicht.

↓Anforderung / Effekt→	Funktionalität +	Leistung / Effizienz +	Zuverlässigkeit +	Benutzbarkeit +	Sicherheit +	Wartbarkeit +
Weiterentwickelbarkeit	–	–	**o**	**o**	**o**	**o**

Tab. 9.2-1: Gegenseitige Abhängigkeiten.

Die Weiterentwickelbarkeit steht im Zusammenhang mit der Wartbarkeit (siehe »Wartbarkeit«, S. 116). Eine Erhöhung der Weiterentwickelbarkeit verbessert auch immer die Wartbarkeit, was umgekehrt aber *nicht* gilt.

Zusätzliche Funktionalität und eine Erhöhung der Leistung/Effizienz wirken sich negativ auf die Weiterentwickelbarkeit aus (Tab. 9.2-1). Eine verbesserte Zuverlässigkeit, Benutzbarkeit und Sicherheit kann sich positiv oder negativ auf die Weiterentwickelbarkeit auswirken.

Frage Überlegen Sie sich Beispiele, die den Unterschied zwischen Weiterentwickelbarkeit und Wartbarkeit zeigen.

Antwort Wird in einem Softwaresystem eine Plugin-Schnittstelle von der Architektur her vorgesehen, dann ist der Austausch von Komponenten im Betrieb statt durch Weiterentwicklung durch Änderungen möglich.

Was ist zu tun?

Die Weiterentwickelbarkeit kann durch die Einhaltung von folgenden Regeln verbessert werden:

- Eine gute Weiterentwickelbarkeit kann u. a. durch Komponentenmodelle erreicht werden.
- Wichtig sind ebenfalls die Rückverfolgbarkeit *(tracebility)* von Entwurfsentscheidungen sowie die explizite Darstellung von Abhängigkeiten.

Prinzipien Alle Prinzipien, die helfen, Komplexität zu beherrschen, unterstützen die Weiterentwickelbarkeit (siehe »Architekturprinzipien«, S. 29):

- **Prinzip der Strukturierung**: Dadurch bessere Verständlichkeit und Analysierbarkeit.
- **Prinzip der Hierarchisierung**: Dadurch bessere Verständlichkeit und Analysierbarkeit.
- **Prinzip der Bindung und Kopplung**: Dadurch Begrenzung der Auswirkungen von Änderungen, d. h. bessere Stabilität.
- **Prinzip der Modularisierung**: Dadurch wird eine weitgehende Kontextunabhängigkeit von der Umgebung erreicht. Die Testbarkeit wird verbessert, da Module für sich testbar sind.
- **Geheimnisprinzip**: Änderungen an den Datenstrukturen wirken sich *nicht* auf die Zugriffsmethoden aus. Dadurch wird eine lokale Änderbarkeit ermöglicht.
- **Prinzip der Lokalität**: Da sich alle benötigten Informationen lokal an einer Stelle befinden, wird die Verständlichkeit und Analysierbarkeit wesentlich verbessert.
- **Prinzip der Verbalisierung**: Dadurch bessere Verständlichkeit und Analysierbarkeit.

- **Architekturprinzip: Konzeptionelle Integrität**: Dadurch schnellere Einarbeitung (siehe »Architekturprinzip: Konzeptionelle Integrität«, S. 30).
- **Architekturprinzip: Trennung von Zuständigkeiten**: Dadurch schnellere Einarbeitung und leichtere Analysierbarkeit (siehe »Architekturprinzip: Trennung von Zuständigkeiten«, S. 31).
- **Architekturprinzip: Sichtbarkeit**: Dadurch bessere Verständlichkeit und Analysierbarkeit (siehe »Architekturprinzip: Sichtbarkeit«, S. 34).
- **Architekturprinzip: Selbstorganisation**: Dadurch geringere Komplexität und weniger zentrale Steuerung (siehe »Architekturprinzip: Selbstorganisation«, S. 34).

Architektur-& Entwurfs-Muster

Folgende Architektur- und Entwurfs-Muster unterstützen u. a. die Weiterentwickelbarkeit:

- »Fabrikmethoden-Muster *(factory method pattern)*«, S. 89: Ermöglicht eine schwache Kopplung zwischen Klassen. Erlaubt es mit geringen Änderungen am Anwendungsprogramm, eine Produktfamilie zu erweitern.
- »Das Strategie-Muster *(strategy pattern)*«, S. 96: Erlaubt den Austausch bzw. das Hinzufügen von Algorithmen.
- »Das Schichten-Muster *(layers pattern)*«, S. 46: Zur Entkopplung der Subsysteme.
- »Das Beobachter-Muster *(observer pattern)*«, S. 54: Zur Entkoppelung des Subsystems Benutzungsoberfläche von dem Subsystem Applikation.
- »Das MVC-Muster *(model view controller pattern)*«, S. 62: Zur Entkopplung der Subsysteme.

Literatur

[RiBo09], [BBR09]

9.3 Betriebssicherheit und Funktionssicherheit

Die nichtfunktionale Anforderung Sicherheit muss in die Betriebssicherheit und die Funktionssicherheit unterteilt werden.

Betriebssicherheit

Die **Betriebssicherheit** *(security)* eines Softwaresystems ist eine fundamentale Anforderung. In der Norm ISO/IEC 9126–1 ist die Sicherheit ein Teilmerkmal des Qualitätsmerkmals Funktionalität und ist wie folgt definiert:

Definition

> Fähigkeit des Softwareprodukts, Informationen und Daten so zu schützen, dass nicht autorisierte Personen oder Systeme sie nicht lesen oder verändern können, und autorisierten Personen oder Systemen der Zugriff nicht verweigert wird.

In [MZN10, S. 315] wird **Betriebssicherheit** wie folgt definiert:

Definition

> Anforderungen, die sich darauf beziehen, unautorisierten Zugriff zu Systemen, Programmen und Daten zu verhindern.

Betriebssicherheit wird in folgende Teilmerkmale untergliedert:

- Vertraulichkeit *(confidentiality)*
- Integrität *(integrity)*
- Verfügbarkeit *(availability)*
- Zugriffssteuerung *(access control)*
- Authentifizierung *(authentication)*

Die Betriebssicherheit ist entscheidend in der Betriebsphase eines Systems.

↓Anforderung / Effekt→	Funktionalität +	Leistung / Effizienz +	Zuverlässigkeit +	Benutzbarkeit +	Sicherheit +	Wartbarkeit +
(Betriebs-) Sicherheit	–	–	**+**	–	**o**	**+**

Tab. 9.3-1: Gegenseitige Abhängigkeiten.

Wie die Tab. 9.3-1 zeigt, führt eine zunehmende Funktionalität und eine Steigerung der Leistung/Effizienz jeweils zu einer reduzierten Sicherheit. Eine verbesserte Benutzbarkeit kann zu einer schlechteren Sicherheit führen, da sicherheitsrelevante Eingaben durch den Benutzer u. U. reduziert werden – um den Aufwand für den Benutzer zu reduzieren.

Eine erhöhte Zuverlässigkeit und eine verbesserte Wartbarkeit dürften sich positiv auf die Sicherheit auswirken.

Zusammenhänge

Die Realisierung der Anforderung **Mehrbenutzerfähigkeit** (siehe »Authentifizierung und Autorisierung«, S. 153) erfordert eine Benutzerverwaltung und eine Rechteverwaltung einschließlich Authentifizierung.

Was ist zu tun?

Regeln

Die Betriebssicherheit kann durch die Einhaltung von folgenden Regeln verbessert werden:

- Eine hohe Betriebssicherheit wird durch eine Authentisierung, Authentifizierung und Autorisierung erreicht (siehe »Authentifizierung und Autorisierung«, S. 153).

- Durch die Beachtung folgender, spezieller Entwurfsprinzipien kann die Betriebssicherheit verbessert werden (siehe »Authentifizierung und Autorisierung«, S. 153):
  - Prinzip der ökonomischen Sicherheitsmechanismen
  - Prinzip der sicheren Voreinstellungen
  - Prinzip der vollständigen Zugriffsüberprüfung
  - Prinzip des offenen Entwurfs
  - Prinzip der Aufteilung von Privilegien
  - Prinzip des kleinsten Privilegs
  - Prinzip der psychologischen Akzeptanz
  - Prinzip des kleinsten allgemeinen Mechanismus

Prinzipien

Die Einhaltung folgender Prinzipien (siehe »Architekturprinzipien«, S. 29) unterstützt die Sicherheit:

- **Prinzip der Bindung und Kopplung**: Dadurch Begrenzung der Auswirkungen von Sicherheitsmängeln.
- **Geheimnisprinzip**: Datenstrukturen können *nicht* direkt manipuliert werden.

Architektur-& Entwurfs-Muster

Folgende Architektur- und Entwurfs-Muster unterstützen u. a. die Sicherheit:

- »Das Schichten-Muster *(layers pattern)*«, S. 46: Einführung von Zugriffsschichten.
- »Das Proxy-Muster *(proxy pattern)*«, S. 83: Verwendung von Schutz-Proxy und Firewall-Proxy.
- »Das Fassaden-Muster *(facade pattern)*«, S. 69: Verwendung einer Fassade zur Zugangskontrolle.
- »Das Kommando-Muster *(command pattern)*«, S. 75: Zur Protokollierung ausgeführter Kommandos und zur Modellierung von Transaktionen.

Funktionssicherheit

Definition

Die **Funktionssicherheit** *(safety)* eines Systems ist ein Maß für die Fähigkeit einer Betrachtungseinheit, weder Menschen, Sachen noch die Umwelt zu gefährden [Biro97].

Während die Betriebssicherheit *(security)* den Schutz von (Software-)Systemen vor Einflüssen und insbesondere Bedrohungen aus der Umwelt fordert, verlangt die Funktionssicherheit den Schutz der Umwelt vor Systemen.

Der Unterschied liegt in der Wirkrichtung. Im Falle der Betriebssicherheit entsteht die Bedrohung außerhalb des betrachteten Systems und wirkt ggf. in das System hinein. Funktionssicherheit fragt nach Bedrohungen, die in einem betrachteten System entstehen und auf seine Umgebung wirken.

Analyse der Restrisiken

Aus diesem Grund erfordert die Beurteilung der Funktionssicherheit stets eine Analyse der gesamten Wirkkette, da die Kritikalität von Systemen, Systemteilen oder Funktionen nur anhand der potenziellen Schadenshöhe in der Umgebung eingeschätzt werden kann. Dies bedeutet im Normalfall, dass Software sowie elektronische und mechanische Komponenten analysiert werden müssen. Der Standard IEC 61508 10 (Funktionale Sicherheit sicherheitsbezogener elektrischer/ elektronischer/ programmierbar elektronischer Systeme) definiert Sicherheit im Sinne von *safety* als die Abwesenheit unakzeptabler Risiken. Diese Definition unterstreicht, dass sichere Systeme durchaus Restrisiken enthalten können. Sie dürfen eine bestimmte Akzeptanzschwelle nicht übersteigen. Daher ist im Rahmen einer Zertifizierung eines sicherheitskritischen Systems im Kontext eines Zulassungsverfahrens die quantitative Bestimmung der Restrisiken erforderlich. Anschließend muss entschieden werden, ob diese Risiken akzeptabel sind oder Modifikationen des Systems erforderlich sind. Man bezeichnet diesen Vorgang als Risikoakzeptanz.

Unfall - verhütung & technische Sicherheit

Die Funktionssicherheit kann in die Sicherheit des ausfallfreien Systems (Unfallverhütung) und die sogenannte technische Sicherheit des ausfallbehafteten Systems untergliedert werden. Systeme müssen so konstruiert sein, dass bei perfekter Funktion aller Komponenten keine unangemessenen Gefährdungen von ihnen ausgehen. Darüber hinaus wird oft verlangt, dass Systeme auch in Gegenwart bestimmter Ausfälle sicher bleiben. Dies kann z. B. durch redundante Komponenten erreicht werden, die die Funktion ausgefallener Komponenten übernehmen.

9.4 Zuverlässigkeit

Softwaretechnik

In der Softwaretechnik sind folgende Definitionen des Begriffs Zuverlässigkeit gebräuchlich:

Die nichtfunktionale Anforderung **Zuverlässigkeit** *(reliability)* wird in der Norm ISO/IEC 9126–1 als Qualitätsmerkmal eingeordnet und wie folgt definiert:

Definition

> Fähigkeit des Softwareprodukts, ein spezifiziertes Leistungsniveau zu bewahren, wenn es unter festgelegten Bedingungen benutzt wird.

Zuverlässigkeit wird untergliedert in folgende Teilmerkmale:

- **Reife** *(maturity)*
  Fähigkeit des Softwareprodukts, trotz Fehlzuständen in der Software nicht zu versagen.

- **Fehlertoleranz** *(fault tolerance)*
  Fähigkeit des Softwareprodukts, ein spezifiziertes Leistungsniveau bei Software-Fehlern oder Nicht-Einhaltung ihrer spezifizierten Schnittstelle zu bewahren.
- **Wiederherstellbarkeit** *(recoverability)*
  Fähigkeit des Softwareprodukts, bei einem Versagen ein spezifiziertes Leistungsniveau wiederherzustellen und die direkt betroffenen Daten wiederzugewinnen.

In [MZN10, S. 315] wird **Zuverlässigkeit** wie folgt definiert:

Definition

> Anforderungen, die die Fähigkeit eines Softwareprodukts spezifizieren, ohne Fehler zu arbeiten und ein festgelegtes Leistungsniveau beizubehalten, wenn es unter spezifizierten normalen Einsatzbedingungen während einer gegebenen Zeitperiode betrieben wird.

Zuverlässigkeit wird in folgende Teilmerkmale untergliedert:

- Vollständigkeit *(completness)*
- Genauigkeit *(accurarcy)*
- Konsistenz *(consistency)*
- Verfügbarkeit *(availibility)* einschl. 24/7-Betriebsfähigkeit
- Integrität *(integrity)*
- Korrektheit *(correctness)*
- **Reife** *(maturity)* (identisch mit ISO/IEC 9126–1)
- **Fehlertoleranz** *(fault tolerance)* (identisch mit ISO/IEC 9126–1)
- **Wiederherstellbarkeit** *(recoverability)* (identisch mit ISO/IEC 9126–1)
- Einhaltung gesetzlicher, unternehmensinterner und vertraglicher Regelungen *(compliance)*

Eingebettete Systeme

Für Zuverlässigkeit bezogen auf eingebettete Systeme existieren mehrere genormte Definitionen und zahlreiche Definitionen in der Fachliteratur.

Die Norm DIN 40041 / DIN 40041 90 (Zuverlässigkeit; Begriffe) definiert **Zuverlässigkeit** als Teil der Qualität im Hinblick auf das Verhalten der Einheit während oder nach vorgegebenen Zeitspannen bei vorgegebenen Anwendungsbedingungen.

Die DIN EN ISO 9000 / DIN EN ISO 9000 05 (Qualitätsmanagementsysteme – Grundlagen und Begriffe) definiert **Zuverlässigkeit** als Sammelbegriff zur Beschreibung der Verfügbarkeit und ihrer Einflussfaktoren Funktionsfähigkeit, Instandhaltbarkeit und Instandhaltungsbereitschaft. Der entsprechende Begriff im englischen Sprachraum für Zuverlässigkeit in diesem umfassenden Sinne ist *Dependability*.

[Biro97] verwendet eine eingeschränktere Definition des Zuverlässigkeitsbegriffs. Diese Zuverlässigkeit im engeren Sinne wird mit dem Begriff *Reliability* übersetzt. Sie ist ein Maß für die Fähigkeit

einer Betrachtungseinheit, funktionstüchtig zu bleiben, ausgedrückt durch die Wahrscheinlichkeit, dass die geforderte Funktion unter vorgegebenen Arbeitsbedingungen während einer festgelegten Zeitdauer ausfallfrei ausgeführt wird. Diese Definition gestattet eine Beschreibung der Zuverlässigkeit mit Hilfe stochastischer Techniken. So kann der Erwartungswert für die Zeitspanne bis zum Ausfall eines Systems – die sogenannte *Mean Time to Failure* (MTTF) – angegeben und als Zuverlässigkeitsmaß verwendet werden. Bei reparierbaren Systemen verwendet man den Erwartungswert zwischen zwei aufeinander folgenden Ausfällen – die sogenannte *Mean Time between Failure* (MTBF).

Alternativ kann die Überlebenswahrscheinlichkeit R(t) verwendet werden. Sie gibt die Wahrscheinlichkeit an, ein System, das zum Zeitpunkt Null in Betrieb genommen wurde, am Zeitpunkt t noch intakt anzutreffen.

Definition

Zuverlässigkeit im engeren Sinne bezeichnet die Wahrscheinlichkeit des ausfallfreien Funktionierens einer betrachteten Einheit über eine bestimmte Zeitspanne bei einer bestimmten Betriebsweise.

Die Zuverlässigkeit ist im Allgemeinen nicht konstant, sondern wird bei Software im Zuge der mit Fehlerkorrekturen einhergehenden Zunahme der Stabilität größer sowie durch neue Fehler im Rahmen von Funktionalitätserweiterungen kleiner. Bei Hardware-Komponenten sind Einflüsse des Verschleißes zu beachten. Die beobachtete Zuverlässigkeit wird ferner durch die Art der Benutzung eines Systems bestimmt. Es ist oft möglich, Zuverlässigkeitsprognosen mit statistischen Verfahren zu erzeugen (siehe [Ligg09]).

Definition

Verfügbarkeit ist ein Maß für die Fähigkeit einer Betrachtungseinheit, zu einem gegebenen Zeitpunkt funktionstüchtig zu sein.

Sie wird ausgedrückt durch die Wahrscheinlichkeit, dass die Betrachtungseinheit zu einem gegebenen Zeitpunkt die geforderte Funktion unter vorgegebenen Arbeitsbedingungen ausführt. Als Maß für Verfügbarkeit kann der Quotient MTBF / (MTBF + MTTR) verwendet werden. Der Wert dieses Quotienten kann auf zwei Arten gesteigert werden. Man kann die MTBF vergrößern, also die Zuverlässigkeit erhöhen, oder die MTTR *(Mean Time to Repair)* verringern, also die Stillstandszeitintervalle nach Ausfällen verkürzen.

Während die Steigerung der MTBF auch auf die Zuverlässigkeit wirkt, beeinflusst die Verringerung der MTTR nur die Verfügbarkeit. Die Zuverlässigkeit ist entscheidend in der Betriebsphase eines Systems.

↓Anforderung / Effekt→	Funktionalität +	Leistung / Effizienz +	Zuverlässigkeit +	Benutzbarkeit +	Sicherheit +	Wartbarkeit +
Zuverlässigkeit	–	–	**o**	**+–**	**+**	**+**

Tab. 9.4-1: Gegenseitige Abhängigkeiten.

Wie die Tab. 9.4-1 zeigt, führt eine zunehmende Funktionalität und eine Steigerung der Leistung/Effizienz jeweils zu einer reduzierten Zuverlässigkeit. Werden die Sicherheit und die Wartbarkeit verbessert, dann kann dies einen positiven Einfluss auf die Zuverlässigkeit haben. Eine Verbesserung der Benutzbarkeit kann sich sowohl positiv als auch negativ auf die Zuverlässigkeit auswirken. Werden beispielsweise alle Benutzereingaben auf Korrektheit überprüft, dann erhöht dies die Zuverlässigkeit. Werden andererseits Bedienungsvarianten für den Benutzer erhöht, dann kann die Zuverlässigkeit dadurch sinken, dass nicht alle Varianten getestet werden können.

Was ist zu tun?

Regeln

Die Zuverlässigkeit kann durch die Einhaltung von folgenden Regeln verbessert werden:

- Identifikation der Subsysteme, von deren Zuverlässigkeit die Funktionsweise des Systems maßgeblich abhängt. Festlegung zusätzlicher Qualitäts- und Testmaßnahmen zur Sicherstellung der geforderten Zuverlässigkeit.
- Alle Benutzereingaben und alle Informationen, die andere Systeme liefern, auf Korrektheit überprüfen (defensives Programmieren).
- Mögliche Fehlerquellen in der Programmierung durch Ausnahmeregeln »abfangen« (In Java: `try-catch-finally`).
- Bereitstellung von Programmen, um bei einem Systemabsturz die Wiederherstellung der Betriebsbereitschaft und der Daten zu ermöglichen (siehe »Transaktionen«, S. 177).
- Mehrere Operationen, die als Gesamtheit ausgeführt werden müssen, durch Transaktionen absichern (siehe »Transaktionen«, S. 177).

Prinzipien

Die Einhaltung folgender Prinzipien (siehe »Architekturprinzipien«, S. 29) unterstützt die Zuverlässigkeit:

- **Prinzip der Strukturierung** und **Prinzip der Modularisierung**: Dadurch Begrenzung auf einzelne Subsysteme bei Ausfall dieser Subsysteme.
- **Geheimnisprinzip**: Dadurch, dass andere Systeme die Interna eines Subsystems nicht kennen, wirken sich Ausfälle auf andere Systeme nur aus, wenn die Schnittstellen betroffen sind.
- **Prinzip der Bindung und Kopplung**: Dadurch Begrenzung der Auswirkungen von unzuverlässigen Subsystemen.

Architektur- & Entwurfs-Muster

Folgende Architektur- und Entwurfs-Muster unterstützen u. a. die Zuverlässigkeit:

- »Das Schichten-Muster *(layers pattern)*«, S. 46: Die Un-Zuverlässigkeit einer Subsystem-Schicht wirkt sich nur begrenzt auf andere Schichten aus.
- »Das Kommando-Muster *(command pattern)*«, S. 75: Zur Protokollierung ausgeführter Kommandos, zur Modellierung von Transaktionen und zur Wiederholung von fehlgeschlagenen Transaktionen (Wiederherstellbarkeit).
- »Das Fassaden-Muster *(facade pattern)*«, S. 69: Dadurch, dass andere Systeme die Interna eines Subsystems nicht kennen (bei Black-Box), wirken sich Ausfälle auf andere Systeme nur aus, wenn die Fassade betroffen ist.

9.5 Leistung und Effizienz

Die nichtfunktionale Anforderung **Effizienz** *(efficiency)* wird in der Norm ISO/IEC 9126–1 als Qualitätsmerkmal eingeordnet und wie folgt definiert:

Definition

> Fähigkeit des Softwareprodukts, ein angemessenes Leistungsniveau bezogen auf die eingesetzten Ressourcen unter festgelegten Bedingungen bereitzustellen.

Effizienz wird untergliedert in folgende Teilmerkmale:

- **Zeitverhalten** *(time behaviour)*
  Fähigkeit des Softwareprodukts, angemessene Antwort- und Verarbeitungszeiten sowie Durchsatz bei der Funktionsausführung unter festgelegten Bedingungen sicherzustellen.
- **Verbrauchsverhalten** *(resource utilisation)*
  Fähigkeit des Softwareprodukts, eine angemessene Anzahl und angemessene Typen von Ressourcen zu verwenden, wenn die Software ihre Funktionen unter festgelegten Bedingungen ausführt.

In [MZN10, S. 315] wird **Leistung** *(performance)* wie folgt definiert:

Definition

> Anforderungen, die die Fähigkeit eines Softwareprodukts spezifizieren, eine angemessene Leistung relativ zu den benötigten Ressourcen zu erbringen, wobei die volle Funktionalität unter festgelegten Bedingungen bereitgestellt wird.

Leistung wird in folgende Teilmerkmale untergliedert:

- Antwortzeit *(response time)*
- Speicherplatz *(space)*

- Kapazität *(capacity)*
- Wartezeit *(latency)*
- Durchsatz *(throughput)*
- Rechenzeit *(computation)*
- Ausführungsgeschwindigkeit *(execution speed)*
- Übergangsverzögerung *(transit delay)*
- Auslastung *(workload)*
- **Verbrauchsverhalten** *(resource utilisation)* (identisch mit ISO/IEC 9126–1)
- Speicher-Inanspruchnahme *(memory usage)*
- **Genauigkeit** *(accurarcy)* (in ISO/IEC 9126–1 unter Funktionalität subsumiert)
- Modi *(modes)*
- Verzögerung *(delay)*
- Fehlerraten *(miss rates)*
- Datenverlust *(data loss)*
- Nebenläufige Transaktionsverarbeitung *(concurrent transaction processing)*

Die Leistung/Effizienz ist entscheidend in der Betriebsphase eines Systems.

↓Anforderung / Effekt→	Funktionalität +	Leistung / Effizienz +	Zuverlässigkeit +	Benutzbarkeit +	Sicherheit +	Wartbarkeit +
Leistung / Effizienz	–	**o**	–	**+**–	–	–

Tab. 9.5-1: Gegenseitige Abhängigkeiten.

Wie die Tab. 9.5-1 zeigt, führen zusätzliche Funktionalität, verbesserte Zuverlässigkeit, erhöhte Sicherheit und bessere Wartbarkeit zu einem negativen Effekt auf die Leistung/Effizienz. Eine verbesserte Benutzbarkeit kann sich positiv oder negativ auf die Leistung/Effizienz auswirken.

Was ist zu tun?

Regeln

Die Leistung/Effizienz kann durch die Einhaltung von folgenden Regeln verbessert werden:

- Die einzusetzenden Algorithmen sind so auszuwählen, dass sie für das geforderte Effizienzkriterium optimal sind.
- Identifikation der Subsysteme, die entscheidend für die Leistungsfähigkeit des Systems sind. Gezielte Optimierung dieser Subsysteme entsprechend dem geforderten Effizienzkriterium.

Prinzipien

Die Einhaltung folgender Prinzipien (siehe »Architekturprinzipien«, S. 29) unterstützt die Leistung/Effizienz:

- **Prinzip der Bindung und Kopplung**: Der Informationsfluss findet lokal in Komponenten und nicht zwischen Komponenten statt. Dadurch wird die Effizienz verbessert.

- **Achitekturprinzip Ökonomie**: Dadurch wird sichergestellt, dass nur das realisiert wird, was unbedingt benötigt wird. Die Effizienz wird nicht durch unnötige Funktionalität oder Architekturkonzepte reduziert.
- **Achitekturprinzip Trennung von Zuständigkeiten**: Dadurch, dass Zuständigkeiten an jeweils einer Stelle erledigt werden, wird ein unnötiger Informationsaustausch mit anderen Subsystemen vermieden.
- **Architekturprinzip Selbstorganisation**: Die Selbstorganisation von Komponenten reduziert zentrale Steuerungsmechanismen und verbessert dadurch die Effizienz.

Architektur- & Entwurfs-Muster

Folgendes Entwurfs-Muster unterstützt u. a. die Leistung/Effizienz:

- »Das Proxy-Muster *(proxy pattern)*«, S. 83: Einsatz eines **virtuellen Proxy**, um »teure« Methoden nur aufzurufen, wenn sie auch benötigt werden. Einsatz eines **Cache-Proxy** für aufwändige Methoden, damit verschiedene Clients auf die Ergebnisse zugreifen können.

9.6 Benutzbarkeit

Die nichtfunktionale Anforderung **Benutzbarkeit** *(usability)* – oft auch **Gebrauchstauglichkeit** genannt – wird in der Norm ISO/IEC 9126–1 als Qualitätsmerkmal eingeordnet und wie folgt definiert:

Definition

> Fähigkeit des Softwareprodukts, vom Benutzer verstanden und benutzt zu werden sowie für den Benutzer erlernbar und »attraktiv« zu sein, wenn es unter den festgelegten Bedingungen benutzt wird.

Die Benutzbarkeit wird in folgende Teilmerkmale untergliedert:

- **Verständlichkeit** *(understandability)*
  Fähigkeit des Softwareprodukts, den Benutzer zu befähigen, zu prüfen, ob die Software angemessen ist und wie sie für bestimmte Aufgaben und Nutzungsbedingungen verwendet werden kann.
- **Erlernbarkeit** *(learnability)*
  Fähigkeit des Softwareprodukts, den Benutzer zu befähigen, die Anwendung zu erlernen.
- **Bedienbarkeit** *(operability)*
  Fähigkeit des Softwareprodukts, den Benutzer zu befähigen, die Anwendung zu bedienen und zu steuern.
- **Attraktivität** *(attractiveness)*
  Fähigkeit des Softwareprodukts für den Benutzer attraktiv zu sein, z. B. durch die Verwendung von Farbe oder die Art des grafischen Designs.

- **Konformität der Benutzbarkeit** *(usability compliance)*
  Fähigkeit des Softwareprodukts, Standards, Konventionen, Stilvorgaben *(style guides)* oder Vorschriften bezogen auf die Benutzbarkeit einzuhalten.

In [MZN10, S. 315] wird Benutzbarkeit wie folgt definiert:

Definition

> Anforderungen, die die Benutzer-Interaktionen mit dem System und den benötigten Aufwand, um das System zu erlernen, zu benutzen, die Eingaben in das System vorzubereiten und die Ausgabe des Systems zu interpretieren, festlegen.

Neben den identischen Teilmerkmalen der Norm ISO/IEC 9126-1 werden in [MZN10, S. 315] folgende weitere Teilmerkmale aufgeführt:

- Bedienungskomfort *(ease of use)*
- Ausführungsgeschwindigkeit *(execution speed)*
- Ergonomie *(human engineering)*
- Benutzungsfreundlichkeit *(user friendliness)*
- Einprägsamkeit *(memorability)*
- Effizienz *(efficiency)*
- Benutzerproduktivität *(user productivity)*
- Bedienungskomfort *(ease of use)*
- Ausführungsgeschwindigkeit *(execution speed)*
- Ergonomie *(human engineering)*
- Brauchbarkeit *(usefulness)*
- Erwartungskonformität *(likeability)*
- Benutzerreaktionszeit *(user reaction time)*

Die Benutzbarkeit ist entscheidend in der Entwicklungs- und der Betriebsphase eines Systems.

↓Anforderung / Effekt→	Funktionalität +	Leistung / Effizienz +	Zuverlässigkeit +	Benutzbarkeit +	Sicherheit +	Wartbarkeit +
Benutzbarkeit	**+–**	**+–**	**+**	**o**	–	**o**

Tab. 9.6-1: Gegenseitige Abhängigkeiten.

Wie die Tab. 9.6-1 zeigt, kann zusätzliche Funktionalität und verbesserte Leistung/Effizienz die Benutzbarkeit verbessern aber auch verschlechtern. Eine erhöhte Zuverlässigkeit führt in der Regel auch zu einer verbesserten Benutzbarkeit. Eine verstärkte Sicherheit kann sich auf die Bequemlichkeit der Benutzbarkeit negativ auswirken. Eine verbesserte Wartbarkeit dürfte sich neutral auf die Benutzbarkeit auswirken.

Was ist zu tun?

Regeln

Die Benutzbarkeit kann durch die Einhaltung von folgenden Regeln verbessert werden:

- Alle Aktivitäten, die die Benutzerinteraktion betreffen, werden in einem Subsystem angeordnet.
- Das Subsystem Benutzungsoberfläche greift nur auf das Subsystem Applikation zu, nicht aber auf das Subsystem Persistenz.

Prinzipien

Die Einhaltung folgender Prinzipien (siehe »Architekturprinzipien«, S. 29) unterstützt die Benutzbarkeit:

- **Prinzip der Verbalisierung**: Durch die domänengerechte Wahl der Begriffe auf der Benutzungsoberfläche werden das Verständnis, die Einarbeitung und die Bedienung für den Benutzer erleichtert.
- **Architekturprinzip**: **Ökonomie**: Es wird nur die Funktionalität zur Verfügung gestellt, die der Benutzer zur Erledigung seiner Aufgaben benötigt. Dadurch wird die Benutzbarkeit verbessert.
- **Architekturprinzip**: **Trennung von Zuständigkeiten**: Alle Aufgaben, die mit der Benutzerinteraktion zu tun haben, werden in einem Subsystemen gekapselt.

Architektur- & Entwurfs-Muster

Folgende Architektur- und Entwurfs-Muster unterstützen die Benutzbarkeit:

- »Das Schichten-Muster *(layers pattern)*«, S. 46: Ermöglicht es, die Benutzungsoberfläche in einer Schicht anzuordnen.
- »Das Kommando-Muster *(command pattern)*«, S. 75: Zur Realisierung von Undo- und Redo-Kommandos so wie die Ermöglichung einer Makro-Aufzeichnung.
- »Das MVC-Muster *(model view controller pattern)*«, S. 62: Unterstützt die leichte Erweiterung um zusätzliche Ansichten *(views)* und Eingaben über verschiedene Geräte *(controls).*

9.7 Portabilität

Die nichtfunktionale Anforderung **Portabilität** *(portability)* wird in der Norm ISO/IEC 9126–1 als Qualitätsmerkmal eingeordnet und wie folgt definiert:

Definition

> Fähigkeit des Softwareprodukts, von einer Umgebung in eine andere übertragen zu werden. Umgebung kann organisatorische Umgebung, Hardware- oder Software-Umgebung einschließen.

Portabilität wird in folgende Teilmerkmale untergliedert:

- **Anpassbarkeit** *(adaptability)*
  Fähigkeit des Softwareprodukts, die Software an verschiedene, festgelegte Umgebungen anzupassen, wobei nur Aktionen oder Mittel eingesetzt werden, die für diesen Zweck für die betrachtete Software vorgesehen sind.

- **Installierbarkeit** *(installability)*
  Fähigkeit des Softwareprodukts, in einer festgelegten Umgebung installiert zu werden.
- **Koexistenz** *(co-existence)*
  Fähigkeit des Softwareprodukts, mit anderen unabhängigen Softwareprodukten in einer gemeinsamen Umgebung gemeinsame Ressourcen zu teilen.
- **Austauschbarkeit** *(replaceability)*
  Fähigkeit des Softwareprodukts, diese Software anstelle einer spezifizierten anderen in der Umgebung jener Software für denselben Zweck zu verwenden.
- **Konformität der Portabilität** *(portability compliance)*
  Fähigkeit des Softwareprodukts, Standards oder Konventionen bezogen auf die Portabilität einzuhalten.

Nach [MZN10] gehört Portabilität nicht zu den am häufigsten genannten nichtfunktionalen Anforderungen. Sie wird jedoch bei Echtzeitsystemen als eine wichtige Anforderung aufgeführt.

Was ist zu tun?

Regel

Die Portabilität kann durch die Einhaltung von folgender Regel verbessert werden:

- Die von einer möglichen Portabilität betroffenen Teile sind zu identifizieren und in eigene Subsysteme zu kapseln.

Prinzipien

Die Einhaltung folgender Prinzipien (siehe »Architekturprinzipien«, S. 29) unterstützen die Portabilität:

- **Prinzip der Modularisierung**: Die zu portierenden Teile sind in Module gekapselt.
- **Architekturprinzip Trennung von Zuständigkeiten**: Die von der Portabilität betroffenen Teile sind in Subsystemen isoliert.
- **Architekturprinzip Sichtbarkeit**: Die zu portierenden Teile sind in der Architektur gut sichtbar.

Architektur- & Entwurfs-Muster

Folgende Architektur- und Entwurfs-Muster unterstützen die Portabilität:

- »Das Schichten-Muster *(layers pattern)*«, S. 46: Oft muss nur die unterste Schicht modifiziert werden, um ein System auf eine andere Hardware- oder Software-Umgebung zu übertragen. Soll die Benutzungsoberfläche für einen anderen Gerätetyp geändert werden, dann muss oft nur die oberste Schicht angepasst werden.
- »Das MVC-Muster *(model view controller pattern)*«, S. 62: Unterstützt die Portabilität auf andere Ein- und Ausgabegeräte.
- »Das Fassaden-Muster *(facade pattern)*«, S. 69: Ermöglicht die Anpassung an Schnittstellen umgebender Systeme.