The COSMIC Functional Size Measurement Method

Version 3.0.1

Measurement Manual

(The COSMIC Implementation Guide for ISO/IEC 19761: 2003)

Deutsche Fassung, August 2011

1. Einführung

1.1 Anwendungsaspekte der COSMIC-Methode

1.1.1 Anwendungsbereiche

Die COSMIC-Methode zur Messung des Funktionsumfangs wurde für die Bestimmung der Funktionalität von Software für folgende Anwendungsbereiche entwickelt:

- Business-Applikationssoftware, die typischerweise Geschäftsprozesse unterstützt, wie zum Beispiel die Banken, Versicherungen, Buchhaltung, Personal- und Finanzabteilungen, Vertrieb und Fertigung. Derartige Software wird im Allgemeinen als „datenorientiert“ bezeichnet, da hierbei die Verwaltung umfangreicher Datenbestände im industriellen Umfeld im Vordergrund steht.
- Echtzeitsysteme, deren Aufgabe es ist, Steuerungs- und Kontrollfunktionen zu Ereignissen in der realen Welt zu sichern. Beispiele dieser Software-Art sind Telefon- und Nachrichtenübertragungssysteme, eingebettete Systeme in Haushaltsgeräten, Fahrstühlen, Autos und Flugzeugen für die Prozesssteuerung und automatischer Datenbereitstellung sowie Software in den Betriebssystemen der Computer.
- Hybride Softwaresysteme der beiden oben genannten, wie beispielsweise Reservierungssysteme für Fluglinien oder Hotels.

1.1.2 Nicht betrachtete Anwendungsbereiche

Die COSMIC-Messmethode wurde nicht konzipiert für die Bestimmung der Funktionalität von mathematischen Anwendungssystemen, d. h. von Software, die komplexe mathematische Algorithmen oder spezielle, komplexe Regeln beinhalten, wie z. B. Expertensysteme, Simulationssoftware, Autonome Lernsysteme, Wettervorhersagesysteme usw., sowie kontinuierliche Prozesssteuerungsformen, beispielsweise in Audio- oder Videosystemen, wie zum Beispiel in Computer-Spielen, Musikinstrumenten usw.

Für Software mit einer derartigen Funktionalität ist es jedoch möglich, spezielle, geeignete Erweiterungen der COSMIC-Messmethode zu definieren. Das so genannte *Measurement Manual* beschreibt, in welchem Kontext derartige spezifische Erweiterungen (genannt *local extension)* benutzt werden können und zeigt Beispiele dafür. In diesem Fall der spezifischen Erweiterung sollten allerdings die Konventionen, die in diesem Manual beschrieben sind, eingehalten werden.

1.1.3 Einschränkungen bei der Bestimmung des Funktionsumfang

Im Bereich der softwaregestützten Anwendungssysteme berücksichtigt die COSMIC–Methode zur Funktionsumfangsbestimmung nicht alle möglichen Aspekte einer Funktionalität, die den Software-Umfang ausmacht. So werden beispielsweise weder der Einfluss der unterschiedlichen Komplexitätsaspekte noch die Anzahl der Datenattribute pro Datenbewegung als Software-Funktionsumfang in dieser Methode berücksichtigt (dazu mehr im Kapitel 3 zur Messplanung). Wie bereits im Abschnitt 1.1.2 beschrieben, können im Bedarfsfall derartige Aspekte als spezifische Erweiterungen die COSMIC-Messmethode ergänzen.

1.1.4 Einschränkungen bei der Messung sehr kleiner Softwaresysteme

Alle Messmethoden zum Funktionsumfang (kurz FSM) basieren auf Annahmen zu einem vereinfachten Model der Software-Funktionalität, welches im Allgemeinen auf den „Durchschnitt“ bzw. den „typischen“ Anwendungsbereich zugeschnitten ist. Daher sind insbesondere für die Messung, den Vergleich oder die Anwendung (z. B. als Schätzung) sehr kleiner Software-Systeme oder sehr kleiner Veränderungen an Software-Systemen, die also eher „untypisch“ sind, besondere Vorgehensweisen anzuwenden. „Sehr klein“ bedeutet im Falle der COSMIC–Methode „sehr wenig Datenbewegungen“.

1.2 Funktionale (Nutzer-) Anforderungen

Die COSMIC-Messmethode impliziert eine Menge von Modellen, Prinzipien, Regeln und Prozessen zu den Funktionalen Anforderungen (*Functional User Requirements (FUR)*) zu einer gegebenen Software. Das Ergebnis ist dabei ein numerischer Wert (als *'value of a quantity'* von der ISO definiert), die den funktionalen Umfang bzw. die Größe der Software bezogen auf die COSMIC-Methode darstellt.

Der durch die COSMIC-Methode gemessene funktionale Umfang ist unabhängig von irgendwelchen Implementationsentscheidungen der spezifischen Operationalität der gemessenen Software. „Funktionalität“ bedeutet hierbei die „durch die Software für den Nutzer bereitzustellenden Informationen“.

Genauer gesagt beschreibt eine FUR-Angabe „was“ die Software funktional für den Nutzer, der Sender und Empfänger von Daten ist, leisten muss. Eine FUR-Angabe schließt technische oder qualitative Anforderungen aus, die beschreiben, „wie“ die Software zu arbeiten hat (Eine explizite Definition für FUR ist im Abschnitt 2.2 angegeben). Ausschließlich die FUR-Angaben werden für die Bestimmung des Funktionsumfangs zugrunde gelegt.

1.2.1 Die praktische Bestimmung der funktionalen Anforderungen aus den Software-Artefakten[1]

In der Praxis der Software-Entwicklung findet man selten Dokumente oder Artefakte, in denen die FUR klar beschrieben sind und sich deutlich von anderen Anforderungsarten unterscheiden bzw. geeignet sind, direkt und ohne weitere Hintergrundinformationen gemessen werden zu können. Das bedeutet, dass der Software-Messanwender (später einfach Messperson genannt) zunächst die FUR-Angaben extrahieren und auf das COSMIC-Software-Modell zuschneiden muss.

Abbildung 1.1 zeigt die Möglichkeit der FUR-Angabenbestimmung aus den Software Engineering Artefakten, bevor die Software selbst überhaupt existiert. Das bedeutet, das der Funktionsumfang vor der Implementation der Software in ein konkretes Computer-System bestimmt werden kann.

[1] Derartige Artefakte können Pflichtenhefte, Aufgabenstellungen, Skizzen oder bereits Software-Modelle darstellen.

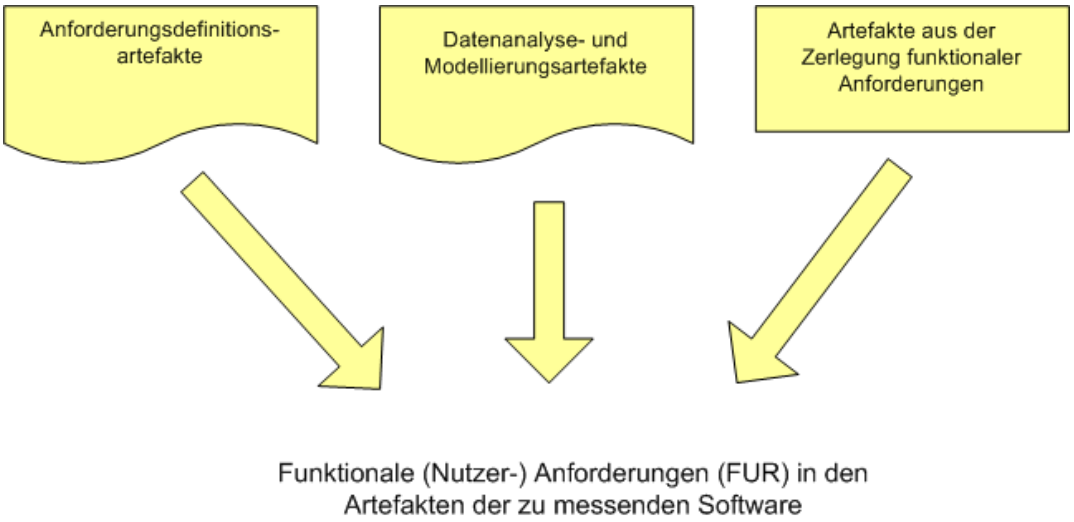

Abb. 1.1: COSMIC-Modell der Bestimmung noch nicht implementierter funktionaler Anforderungen

Es ist wichtig darauf hinzuweisen, dass die genannten Artefake bereits vor der Software erstellt werden, d. h. während der Auflistung bzw. Analyse der Anforderungen, wobei auch unterschiedliche Granularitätsstufen, abhängig vom Detaillierungsgrad der erfassten Anforderungen entstehen (siehe auch Abschnitt 2.4).

Man beachte, dass funktionale Anforderungen erstellt werden, bevor sie der jeweiligen Hard- oder Software zugeordnet werden (können). Die COSMIC-Methode wurde bisher hauptsächlich für Software-Systeme angewandt, bei denen die funktionalen Anforderungen bereits der gemessenen Software zugeordnet war. Jedoch kann die COSMIC-Methode auch auf die FUR-Angaben angewandt werden, bevor die Software zugeordnet wurde, wodurch die Zuordnungsentscheidungen irrelevant werden. So kann beispielsweise der Funktionsumfang eines Handheld-Rechners mit der COSMIC-Methode einfach bestimmt werden, ohne irgendwelche Kenntnisse bezüglich der Hard- oder Software. Allerdings sind für die Anwendung der COSMIC–Methode unter Einbeziehung spezieller Hardware-Merkmale noch mehr praktische Erfahrungen notwendig, bevor diese Methode für diesen Anwendungsfall als hinreichend valide gilt.

1.2.2 Die Bestimmung der funktionalen Anforderungen aus bereits installierter Software

In speziellen Fällen kann es möglich sein, dass zu der zu messenden Software nur wenige Architektur- und Design-Dokumente existieren, so dass die FUR nicht explizit dokumentiert sind (beispielsweise für Altsoftware-Systeme). In diesem Fällen ist es nur möglich, die FUR von den bereits installierten bzw. implementierten Artefakten auf dem Computer zu bestimmen (siehe hierzu auch die Abbildung 1.2).

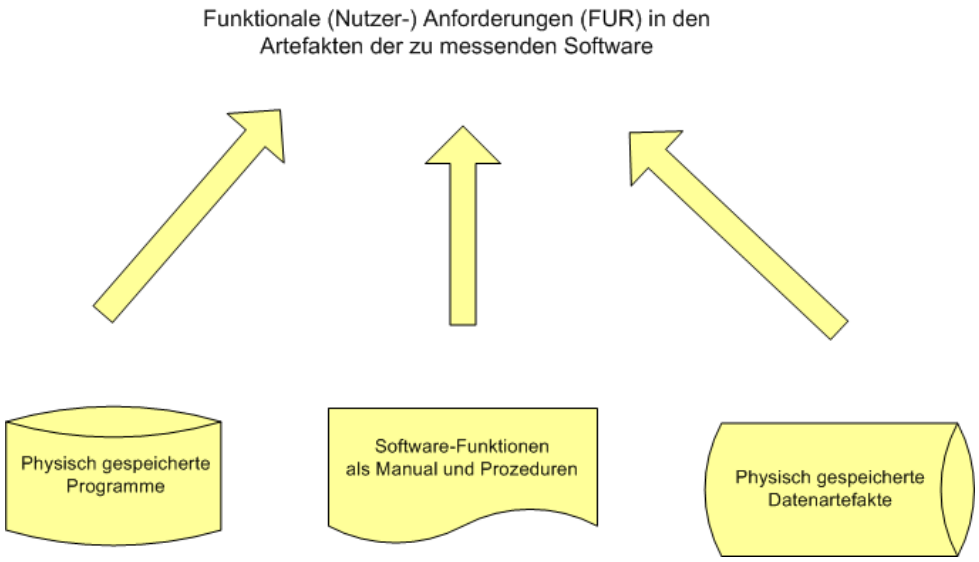

Abb. 1.2: COSMIC-Modell der Bestimmung bereits implementierter funktionaler Anforderungen

1.2.3 Die Bestimmung der funktionalen Anforderungen aus Software-Artefakten

Das vorliegende Messhandbuch beschreibt und definiert dafür die Konzepte des COSMIC-Software-Modells, d. h. die Ziele der Anforderungsbestimmung bzw. -ableitung. Diese Konzepte sind in einer Menge von Prinzipien in zwei COSMIC-Software-Modellen eingebettet: das *Software-Kontextmodell* und das *Generische Software-Modell*.

Diese beiden Modelle sind im Kapitel „Methodenübersicht“ beschrieben. Das Grundverständnis und die grundlegenden Anforderungen für den Leser sind dort beschrieben. Auch die wesentlichen Komponenten dieser Modelle sind im Kapitel 2 und 3 im Detail dargestellt.

1.3 Das COSMIC-Software-Kontextmodell

Die zu messende Software muss bei der Anwendung der COSMIC-Methode bezüglich des Messobjektes (measurement scope) sehr genau definiert sein, wobei diese Darstellung den Kontext zu anderer involvierter Software und/oder Hardware berücksichtigen muss. Das Software-Kontextmodell beschreibt also die Prinzipien und Konzepte für diese Definition.

Die wesentlichen Begriffe sind in den (folgenden) Abschnitten 1.3 und 1.4 fett gedruckt und haben eine exponierte Bedeutung im Rahmen der COSMIC-Methode. Die formalen Definitionen sind noch einmal im Glossary in einer Dokumentationsübersicht beschrieben (siehe Anhang). Diese und die anderen Begriffe werden in den folgenden Kapiteln 2, 3 und 4 noch detaillierter erläutert.

Tab. 1.1: Die allgemeinen Prinzipien des Kontextmodells

Prinzipien des COSMIC-Software-Kontextmodells
a) Software ist stets mit Hardware verbunden.
b) Software ist typischerweise in **Schichten** (layers) strukturiert.
c) Eine Schicht kann aus ein oder mehreren verbundenen Software-Komponenten (**peer** pieces of software) bzw. die Software kann selbst aus separaten Teilkomponenten bestehen.
d) Eine zu messende Software-Einheit (peer component) soll stets durch ihre Messobjektcharakterisierung (measurement **scope**) definiert werden, welche sich vollständig in einem einfachen Layer befinden.
e) Die Messobjektcharakterisierung für die zu messende Software sollte sich auf den **Zweck der Messung** beziehen.
f) Die Nutzer- bzw. **Anwendungsfunktion** der Software aus den zu messenden funktionalen Anforderungen sollten als Sendung und/oder Empfang von Daten identifiziert werden.
g) Software interagiert mit einem Nutzer in Form von **Datenbewegungen** (data movements) im Rahmen von **Abgrenzungen** (boundaries) und kann Daten zu und von einer **persistenten Speicherung** innerhalb dieser Abgrenzung bewegen.
h) Die FUR der Software kann in verschiedenen **Stufen der Granularität** ausgedrückt werden.
i) Die Stufe bzw. das Niveau dieser Granularität, in der die Messung durchgeführt wird, sind die **funktionalen Prozesse**.
j) Wenn es nicht möglich ist, auf dem jeweiligen Granularitätsniveau zu messen, sollten die zu messenden FUR näherungsweise entsprechend bestimmt werden.

Das Konzept des Software-Kontextmodells wird noch einmal im Kapitel 2 bei der Beschreibung der Messstrategie ausführlicher beschrieben.

1.4 Das Generische Software-Modell

Nachdem die FUR der auf der Grundlage des Software-Kontextmodells zu messenden Software herausgearbeitet wurden, besteht nun die Aufgabe, die einzelnen Komponenten der zu messenden Funktionalität durch Anwendung des *Generischen Software-Modells* zu bestimmen.

Dieses Generische Software-Modell basiert darauf, dass im Folgenden behandelte allgemeinen Prinzipien für die zu messenden Software gelten. (Wie auch im Glossary angegeben, helfen funktionale Umfangsmessmethoden, die Funktionstypen allerdings nicht ihr Auftreten zu bestimmen. In der nachfolgenden Tabellen werden diese Typen definiert,

weil es für die COSMIC-Methode wichtig ist, die oben genannte Unterscheidung treffen zu müssen.)

Tab. 1.2: Die allgemeinen Prinzipien des Generischen Software-Modells

Prinzipien des Generischen COSMIC-Software-Modells
a) Software empfängt Eingabedaten des Nutzers und produziert **Ausgabedaten** und/oder andere Ausgaben bzw. Ausgabesignale für den Nutzer.
b) Funktionale Systemanforderungen der zu messenden Software kann in eindeutige Funktionsprozesse abgebildet werden.
c) Jeder funktionale Prozess besteht aus **Teilprozessen**.
d) Ein Teilprozess kann sowohl eine **Datenbewegung** (data movement) oder **Datenmanipulation** (data manipulation) sein.
e) Jeder funktionale Prozess ist mit einem **Eingang** (Entry) zur Datenbewegung aus einer Nutzerfunktion verbunden, der dem Prozess signalisiert, das in der Nutzerfunktion ein **Ereignis** stattfindet.
f) Eine Datenbewegung „bewegt“ eine einfache **Datengruppe**.
g) Eine Datengruppe besteht aus einer eindeutigen Menge von **Datenattributen**, die ein einfaches, zu betrachtendes Objekt beschreibt.
h) Es gibt vier Typen von Datenbewegungen bzw. Datenbehandlungen. Der **Eingang** (Entry) bewegt Daten von der Anwendung bzw. dem Nutzer in die Software. Der **Ausgang** (Exit) bewegt wiederum eine Datengruppe aus der Software zum Nutzer. Ein **Ausschreiben** (Write) bewegt eine Datengruppe aus der Software in eine persistente Speicherungsform. Ein **Einlesen** (Read) bewegt eine Datengruppe aus der persistenten Speicherung in die betrachtete Software.
i) Ein funktionaler Prozess sollte stets aus mindestens einem Eingang bzw. einem Einschreiben und einem dazugehörigen Ausgang bzw. Ausschreiben bestehen. Das bedeutet, dass ein funktionaler Prozess aus **mindestens zwei Datenbewegungen** besteht.
j) Als eine Näherung für den Messprozess sind Datenmanipulationen für die Teilprozesse nicht separat zu messen; die **Funktionalität einer Datenmanipulation** sollte **über** damit **zusammenhängende Datenbewegungen bestimmt** werden.

Die detaillierten Konzepte eines Generischen Software-Modells werden im Kapitel 3 näher beschrieben.

Nachdem die FUR nun auf der Grundlage des Generischen Software-Modells zugeschnitten wurden, kann die unmittelbare Messung auf der Grundlage der COSMIC-Messphasen (siehe auch Kapitel 4) erfolgen.

Die Ergebnisse wiederum sollten nach den Konventionen der COSMIC-Messberichte (siehe Kapitel 5) dargestellt sein.

1.5 Der COSMIC-Messprozess

Die allgemeinen COSMIC-Messprozess besteht aus drei Phasen:

- die ***Messstrategie***, die das Software-Kontextmodell auf die zu messende Software anwendet,
- die ***Zuordnungsphase*** (Mapping Phase) in welcher das Generische Software-Modell für die zu messende Software angewandt wird,
- die (eigentliche) ***Messphase***, die schließlich die aktuellen Umfangsmessungen enthält.

Das Ergebnis der Anwendung des Messprozesses auf die betrachtete Software ist ein Funktionsumfangsmaß für die funktionalen Anforderungen (FUR) ausgedrückt in *COSMIC Function Points* bzw. **CFP**.

Die Beziehungen dieser drei Phasen der COSMIC-Methode untereinander ist in der folgenden Abbildung dargestellt.

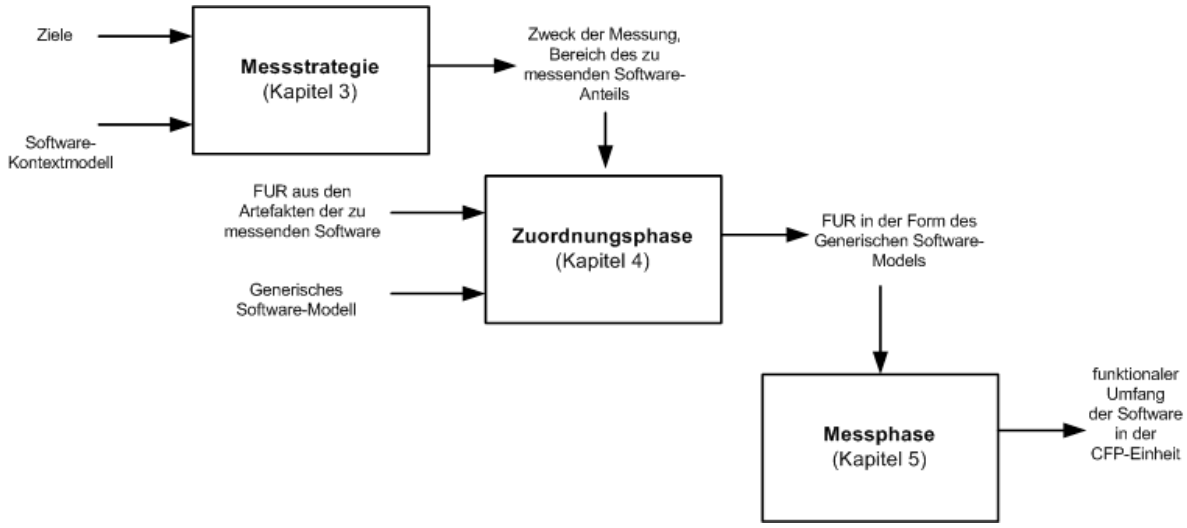

Abb. 1.3: Allgemeine Struktur der COSMIC-Methode

2. Die COSMIC-Messstrategiephase

Dieses Kapitel beschäftigt sich mit den vier Schlüsselaspekten zur funktionellen Größenbestimmung, die vor der eigentlichen Messung berücksichtigt werden müssen, und zwar der Zweck und der Scope der Messung, die Identifizierung der Nutzerfunktion und das Granularitätsniveau der Messung. Die Betrachtung dieser Aspekte führt zu der Frage

Welcher Umfang sollte gemessen werden?

oder im Falle einer realisierten Messung

Wie sind die Messergebnisse zu interpretieren?

Es ist sehr wichtig, die Datenaufbereitung für diese Messstrategiephasen unmittelbar zum Zeitpunkt der jeweiligen Messungen durchzuführen. Fehlerhafte Definitionen und Aufbereitungen führten zu Messungen, die nicht zuverlässig interpretiert und verglichen werden können bzw. und auch keinen korrekten Input für Aufwandschätzprojekte darstellen. Die hier behandelten vier Abschnitte zu den Messstrategiephasen geben formale Definitionen, Regeln und Beispiele für jeden der Schlüsselparameter, die das Softwaremesspersonal im Verlauf der Herleitung einer Messstrategie benötigt (siehe Abb. 2.1), an. In jedem Abschnitt wird werden die Hintergründe für die Bedeutung der jeweiligen Parameter durch die Verwendung geeigneter Analogien, die deren Rolle auch in anderen Messprozessen zeigen, und folglich auch für das Gebiet der funktionalen Größenmessung zu betrachten sind.

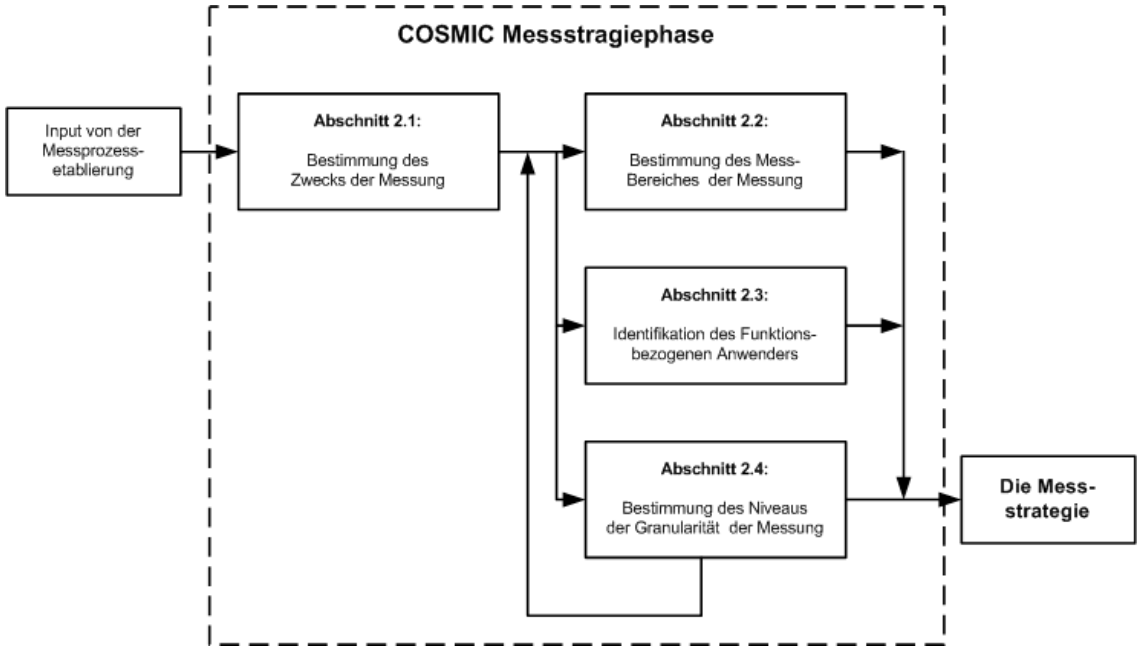

Abb. 2.1: Der Prozess der Messstrategiebestimmung

2.1 Die Definition des Zwecks der Messung

Tab. 2.1: Definition des Zwecks der Messung

Definition des Zwecks der Messung
Der Zweck der Messung sind Angaben, die definieren warum eine Messung notwendig ist und wofür die Messergebnisse benötigt werden.

2.1.1 Der Zweck der Messung und der jeweils zu messende Umfang

Es gibt viele Gründe für die Umfangsmessung, so wie es viele Gründe beispielsweise für die Messung des Umrisses eines Hauses gibt. Wenn es sich z. B. um die Kostenschätzung einer neuen Software-Entwicklung handelt, kann es notwendig sein, zuvor den Umfang der Software zu messen, so wie es beim Hausbau sinnvoll ist zuvor seine Umrisse zu kennen bzw. zu bestimmen. Im unterschiedlichen Kontext, d. h. wenn es notwendig ist, die aktuellen Kosten mit den geschätzten Kosten zu vergleichen, kann es sinnvoll sein, den in die Software-Erstellung eingehende Funktionsumfang eindeutig zu identifizieren, so wie es sinnvoll ist, beim Hausbau die aktuellen Maße mit den geplanten Umrissen zu vergleichen und damit die Bestätigung der korrekten Umsetzung anzuerkennen.

Der Grund für die Bedeutung einer Messung, auch zu sich wenig unterscheidenden Zeitpunkten, liegt darin, *was* gemessen wurde, unabhängig von den Einflüssen der Maßeinheit oder den Messprinzipien. Im Hausbeispiel ist die Messung der Hausumrisse vor dem Hausbau offensichtlich notwendig gemäß dem Bauplan. Die erforderlichen Dimensionen (Länge und Breite) wurden aus dem Bauplan extrahiert auf der Grundlage üblicher Skalierungsverfahren und allgemein anerkannter Konventionen.

In diesem Sinne basiert die funktionale Größenmessung der Software auf die funktionalen Nutzeranforderungen (FUR) aus einem „Bauplan“ bzw. aus den zuvor entwickelten Artefakten. Die FUR wurden aus diesen Artefakten unter Anwendung geeigneter Konventionen und der erforderlichen Dimensionen (Anzahl der Datenbewegungen) identifiziert und somit der Umfang berechnet.

Ein weiterer Sinn der Haus-Analogie zeigt, dass die Messung der Hausumrisse nach dem Hausbau eine ganz andere Art der Messung mit sich bringt. Jetzt sind die Dimensionen (Länge und Breite) vom Bau selbst unter Verwendung anderer Messwerkzeuge (z. B. einem Maßband) zu bestimmen. Jedoch unterscheidet sich die Messung physikalischer Objekte (vom Haus selbst gegenüber seiner Bauzeichnung) hinsichtlich der Dimensionen, der Maßeinheit (einschließlich spezieller Skalierungsvorschriften) und der Messprinzipien in gleicher Weise. Es liegt an dem Messpersonal, basierend auf den festgelegten Zweck, das zu messende Objekt als verbal, durch seinen Besitzer beschriebenes Haus, ob als Bauplan oder gebautes Haus, die geeignetsten Artefakte für die Messung auszuwählen. Natürlich können die gemessenen Umfänge dabei unterschiedlich sein. Dieselben Gründe gelten jedoch auch für die Software-Messung.

2.1.2 Beispiele für den Zweck der Messung

Im Folgenden sind Beispiele für typische Zweckangaben zur Messung aufgelistet:

- die Messung des Umfangs der sich ergebenden FUR als Eingabe für die Schätzung des Entwicklungsaufwands,
- die Messung des Umfangs der bestätigten Änderungen der FUR für das Management der Projekterweiterungen,
- die Umfangsmessung der FUR der entwickelten Software zur Bestimmung der Leistung des Entwicklers,
- die Messung des FUR-Umfangs der gesamten ausgelieferten Software im Verhältnis zur entwickelten Software für die Messung des Anteils funktionaler Wiederverwendung,
- die Umfangsmessung der FUR für bereits existierende Software als Input zur Messung der Performanz der Verantwortlichkeit des Management und der Anwendungs-unterstützung für diese Software,
- die Messung des Umfangs von Änderungen zu einem existierenden Software-System als Umfangsmessung für die Arbeitsergebnisse eines Wartungsteams,
- die Umfangsmessung der Funktionalität existierender Software für die Unterstützungsmöglichkeit künftiger Anwender.

Diese Zweckangaben helfen dem Messpersonal in folgender Weise:

- bei der Bestimmung des Messbereichs und damit der zu messenden Artefakte,
- die Bestimmung der eigentlichen Nutzer der Systemfunktionalität (wie im Abschnitt 2.3 noch gezeigt wird, ändert sich der Funktionsumfang in Abhängigkeit davon, wer bzw. was als Nutzer bzw. Anwender der Funktionalität definiert wurde),
- die Bestimmung des Messzeitpunktes im Rahmen des Software-Lebenszyklus,
- die Festlegung der erforderlichen Messgenauigkeit und somit der jeweils angewandten COSMIC-Messmethode bzw. der anzuwendenden, zweckspezifischen (locally-derived) COSMIC-Methode (zum Beispiel, wenn die Messung zu einem frühen Zeitpunkt m Lebenszyklus erfolgen soll, bevor die FUR klar beschrieben sind). Hierbei geht es vor allem um die Granularität bei der FUR-Messung.

2.2 Definition des Messbereichs

Tab. 2.2: Definition des Messbereichs

Definition des Messbereichs
Der Messbereich (scope of measurement) ist die Menge der Funktionalen Nutzeranforderungen (FUR), die in einem speziellen Prozess der funktionalen Umfangsbestimmung einbezogen sind. Hinweis: Es sollte unterschieden werden zwischen dem „allumfassenden“ Messbereich bei dem die gesamte Software für den Messzweck einbezogen ist, und dem individuell zugeschnittenen Messbereich, bei dem der Umfang separat bzw. spezifisch gemessen werden sollte. In der COSMIC-Messanleitung wird der Begriff „Messbereich“ genau in diesem spezifischen Sinne behandelt und erläutert.

Die Funktionalen Nutzeranforderungen (FUR) sind gemäß der ISO wie folg definiert:

Tab. 2.3: Definition der funktionalen Nutzeranforderungen

Definition der Funktionalen Nutzeranforderungen (FUR)
Funktionale Nutzeranforderungen (FUR) sind eine Teilmenge der Nutzeranforderungen. Anforderungen beschreiben, was die Software zu leisten hat im Sinne von Aufgaben und Diensten. Hinweis: Die Funktionalen Nutzeranforderungen beinhalten Folgendes (sind aber nicht darauf begrenzt): • den Datentransfer (zum Beispiel als Nutzerdateninput oder als Steuersignalgebung), • die Datentransformation (beispielsweise als berechnete Bankzinsen oder ermittelte Durchschnittstemperatur), • die Datenspeicherung (zum Beispiel als gespeicherte Nutzerdaten oder Datensatzhaltung von Zeitbezogenen Temperaturangaben), • die Datensuche (z. B. als Auflistung der aktuellen Angestellten oder als Rückgabe der letzten Flugzeugposition). Beispiele für Nutzeranforderungen, die nicht Funktionale Anforderungen darstellen, sind • qualitative Anforderungen (wie beispielsweise Benutzbarkeit, Zuverlässigkeit, Effizienz und Portabilität), • organisatorischen Bedingungen (wie Operationslokalisierung, Zielhardware oder die Einhaltung spezieller Standards), • Umgebungsanforderungen (wie z B. Interoperabilität, Sicherheit, Privatsphäre und Datenschutz) • Implementationsbedingungen (wie Entwicklungssprache und Entwicklungsplanung).

Tab. 2.4: Regeln für den Messbereich

Regeln für den Messbereich
a) Der Messbereich einer Funktionalen Größenmessung (FSM) sollte unmittelbar durch den Zweck der Messung bestimmt werden. b) Der Messbereich einer einzelnen Messung sollte nicht über eine Schicht der zu messenden Software hinausgehen.

2.2.1 Die Abhängigkeit des Messbereichs vom Zweck der Messung

Es ist von besonderer Bedeutung, zunächst den Messbereich zu bestimmen, bevor man mit dem speziellen Messvorgang beginnt.

Auf das bereits oben genannte Hausbaubeispiel bezogen ist es für die Kostenabschätzung notwendig, die unterschiedlichen Teile des Hauses separat zu messen d. h. das Fundament, die Mauern und das Dach, da hierbei unterschiedliche Bautechniken zur Anwendung kommen. Das Gleiche gilt auch für Abschätzung der Kosten für die Software-Entwicklung.

Im Fall der Entwicklung eines Software-Systems sind die einzelnen Komponenten in unterschiedlichen Schichten der Systemarchitektur zu finden, so dass der Umfang in jeder dieser Schicht separat zu messen ist, was bedeutet, dass jede Komponente einen separaten Messbereich und einen speziellen Zweck der Umfangsmessung besitzt (siehe hierzu auch die Schichtenbetrachtung in Abschnitt 2.2.4).

In gleicher Weise gilt das bei der zu entwickelnden Software innerhalb einer Schicht aus Komponentenpaaren bei dem jedes eine unterschiedliche Technologie benötigt. Dabei ist es ebenso notwendig, den Messbereich für jede Komponente separat festzulegen, bevor die jeweilige Umfangsmessung erfolgt.

Beispiel 1: Wenn jede Komponente der Software eine andere Technik verwendet und die Messungen dienen der Schätzung des Entwicklungsaufwandes, dann sollte für jede Komponente ein getrennter Messbereich definiert werden, weil jeder Größenmessung eine unterschiedliche Entwicklungsproduktivität zugeordnet werden muss. (Weitere Informationen zu den zu betrachtenden Komponenten finden sich im Abschnitt 2.2.5.)

Der Zweck der Messung entscheidet also, welche Software-Anteile mit einbezogen bzw. auszuschließen sind im jeweiligen Messbereich.

Beispiel 2: Wenn es das Ziel ist, die funktionelle Größe der gesamten, durch ein bestimmtes Projekt-Team entwickelnden Software zu messen, wird es zunächst notwendig sein, die funktionalen Nutzeranforderungen aller vom Team kommenden unterschiedlichen Teile und Komponenten zu definieren. Dazu gehören beispielsweise die FUR eines Softwareteils, welches einmalig für die Datenkonvertierung verwendet wurde.

Beispiel 3: Wenn der Zweck ist, nur die Größe der neuen, unmittelbar verfügbaren Software zu messen, wäre dies kleiner als bei Beispiel 2 und die FUR der Konvertierungssoftware würde nicht in den Anwendungsbereich der Größenmessung einbezogen werden.

Insgesamt gilt, dass der Zweck der Messung stets nach der Regel (a) zu bestimmen ist (also welche Software einbezogen werden muss und welche auszuschließen ist) und der Messbereich gemäß Regel (b) von der einbezogenen Software festlegt, wie diese in weitere Komponenten zu unterteilen ist, die jeweils ihren eigenen Messbereich bilden und separat zu messen sind.

2.2.2 Beispiele für Messbereiche

Die folgenden Beispiele sind *generische Messbereichstypen*:

- Ein Firmenportfolio,
- Eine vertragsmäßige Einverständniserklärung zu Systemanforderungen,
- Ein Arbeitsergebnis eines Projekt-Teams (d. h. einschließlich der Parameterfestlegung für existierende Software, eingebetteter und wieder verwendeter Komponenten sowie speziell für das Projekt entwickelte Datenkonvertierungen und Testdaten-generierungen),
- Ein Entwicklungsergebnis eines Projekt-Teams (d. h. einschließlich der nur für dieses Projekt verwendeten Software sowie Unterstützungssoftware und Testkomponenten),
- Die gesamte Software in einer Schicht,
- Ein Software-Packet,
- Eine Applikation,
- Eine Haupt- (interaktions-) Komponente einer Applikation,
- Eine wieder verwendete Klasse,
- All die Änderungsanforderungen für ein neues Release einer bereits existierenden Software.

In der Praxis sollte ein Messbereich explizite Software charakterisieren und nicht generisch sein, wie im Fall des Arbeitsergebnisses des Projekt-Teams „A“, oder der Applikation „B“ oder dem Firmenportfolio „C“. Die Messbereichsfestlegung sollte unbedingt klären, was jeweils auszuschließen ist.

2.2.3 Niveaustufen der Dekomposition

Es sei bemerkt, dass die oben bestimmten unterschiedlichen Formen von generischen Messbereichen jeweils unterschiedliche Niveaustufen der Dekomposition, wie im Folgenden definiert, erfordern.

Tab. 2.5: Definition der Niveaustufen der Dekomposition

Definition der Niveaustufen der Dekomposition
Eine Niveaustufe (bzw. ein Level) eines Teilbereich der (ausgewählten) Software zeigt alle seine Komponenten, Teilkomponenten usw. Hinweis: Nicht zu verwechseln mit dem Niveau der Granularität.

Beispiel: Unterschiedliche Niveaus der Dekomposition, die verschiedenen generischen Messbereichstypen berücksichtigen, würde darin bestehen, dass ein „Applikationsportfolio“, welche aus mehreren Applikationen besteht, einerseits als Menge von Hauptkomponenten und andererseits als Menge wieder verwendeter Klassenobjekte betrachtet werden kann.

Die Bestimmung des Messbereichs kann dabei mehr bedeuten als nur die einfache Entscheidung, welche Funktionalität in die Messung einbezogen werden sollte. Diese

Bestimmung kann auch die Betrachtung der jeweiligen Niveaustufe der Dekomposition der Software, in welcher die Messung vorgenommen wird, erfordern. Das ist eine wichtige Entscheidung für die COSMIC-Messstrategie, abhängig vom Zweck der Messung, da die unterschiedlichen Niveaustufen nicht einfach verglichen werden können. Das liegt vor allem daran, dass der Umfang eines betrachteten Anteils von Software sich nicht einfach aus der Summe der Größe seiner Teilkomponenten ergibt (siehe auch Abschnitt 4.3.1).

2.2.4 Schichten

Da für den Messbereich des zu messenden Software-Anteils stets auf eine einfache Software-Schicht bzw. Software-Layer zu begrenzen ist, erfordert dieser Prozess der Messbereichsbestimmung zuerst vom Messpersonal zu entscheiden, welches die jeweiligen Schichten einer betrachteten Software-Architektur eigentlich darstellen. In diesem Abschnitt wird daher der Begriff der Schicht zunächst definiert und im Zusammenhang mit interagierenden Software-Komponenten diskutiert, wie es im Rahmen der COSMIC-Methode erforderlich ist.

Die Gründe, warum diese Definitionen erforderlich sind, sind folgende:

- Das Messpersonal kann bisher in klassischen Software-Umgebungen tätig gewesen sein, die auch klassische, so genannte Spaghetti-Architekturen beinhalteten und die der aktuellen Systemarchitektur nicht mehr entsprechen. Das Messpersonal benötigt also in diesem Fall eine Anleitung zur Unterscheidung der Architekturschichten gemäß der COSMIC-Terminologie.
- Die Ausdrücke „Schicht“ bzw. „Schichtenarchitektur“ und „Interaktionskomponente“ werden nicht konsistent in der Software-Industrie verwendet. Wenn das Messpersonal Software in einer Schichtenarchitektur messen soll, muss man prüfen, dass diese Schichten im Sinne der COSMIC-Methode definiert bzw. zu ihr kompatibel betrachtet werden. Dafür sollte das Messpersonal eine Äquivalenz zwischen den spezifischen Architekturobjekten und dem Paradigma der Schichtenarchitektur gemäß dem Schichtenkonzept in diesem Messhandbuch explizit sichern.

Schichten können aufgrund folgender Definitionen und Prinzipien eindeutig bestimmt werden:

Tab. 2.6: Definition einer Schicht

Definition der Schicht
Eine Schicht bzw. ein Layer ist ein Abschnitt einer funktionalen Bereichs einer Software-Architektur, die gemeinsam mit der Hardware ein ganzheitliches Computersystem bildet, wobei • die Schichten hierarchisch organisiert sind, • es stets nur eine Schicht innerhalb eines Hierarchieniveaus gibt, • es jeweils übergeordnete und untergeordnete Hierarchieabhängig-keiten zwischen den funktionalen, software-gestützten Diensten bei denen jeweils zwei Schichten in der Software-Architektur direkt Daten austauschen gibt, • die jeweilige Software in eine der beiden Architekturschichten nur einen Teil der Daten beim Datenaustausch interpretiert.

Die Schichtenbestimmung ist ein iterativer Prozess. Die jeweilige Schicht wird durch einen fortschreitenden Abbildungsprozess immer exakter bestimmt. Als Schicht identifiziert, muss diese den folgenden Prinzipien und Regeln entsprechen:

Tab. 2.7: Prinzipien für die Schichtenbildung

Prinzipien für die Schichtenbildung
a) Die Software einer Schicht tauscht Daten mit der Software einer anderen Schicht auf der Grundlage funktionaler Prozesse aus. b) Die Hierarchiebezogene Abhängigkeit zwischen Schichten kann darin bestehen, dass eine Schicht die funktionalen Dienste der Software einer anderen, in der Hierarchie darunter liegenden Schicht, benutzt. Bei einer derartigen Anwendungsbeziehung bezeichnen wir die verwendete Softwareschicht als „übergeordnet“ und die anwendende Softwareschicht als „untergeordnet“. Die Software in der übergeordneten Schicht verlässt sich auf den Dienst der untergeordneten Schicht für einen korrekten Ablauf, wobei der korrekte Ablauf wiederum sich auf die darunter liegenden Schichten bezieht usw. Umgekehrt kann allerdings die Software der untergeordneten Schicht mit den davon abhängigen Schichten arbeiten, ohne die Dienste der übergeordneten Schichten zu benötigen. c) Die Software einer Schicht muss nicht notwendigerweise alle Funktionalitäten der durch die untergeordnete Schicht angebotenen Dienste benutzen. d) Die Daten, die als Austausch zwischen der Software zweier Schichten definiert ist, kann unterschiedlich bezüglich der FUR dieser beiden Software-Komponenten interpretiert werden. Dies bedeutet, dass bei den beiden Software-Komponenten unterschiedliche Attribute und/oder Teilgruppierungen der ausgetauschten Daten betrachtet werden können. Jedoch müssen dabei auch ein oder mehrere gemeinsame Datenattribute bzw. Datengruppen definiert sein, die von der jeweiligen Schicht im Sinne der bestehenden Funktionalität empfangen bzw. gesendet werden können.

Tab. 2.8: Regeln zur Schichtenbildung

Regeln zur Schichtenbildung
a) Wenn Software eine Schichtenarchitektur in dem hier betrachteten Sinne aufweist, sollte diese Architektur auch zur Identifikation der Schichten für den Messzweck angewandt werden. b) In den Anwendungsbereichen der MIS oder Business-Applikationen wird die oberste Schicht, d. h. die Schicht, die selbst keine Unterschicht darstellt, als *Applikationsschicht* bezeichnet. (Applikations-) Software dieser Schicht basiert auf den Diensten der Software in allen anderen Schichten für eine korrekte Funktionalität. Im Falle von Echtzeit-Software bezeichnet man die oberste Schicht als *System*, beispielsweise als Prozesssteuerungssystem-Software oder als Flugsteuerungssystem-Software.

c) Im Zweifelsfall sollte man die Konzepte der Kopplung und Bindung zwischen verschiedenen Interaktionschichten anwenden (siehe Anhang D).

d) Man sollte nicht annehmen, dass es möglich sei, irgendeine Software ohne genauere Betrachtung des architekturellen oder strukturellen Entwurfs in Schichten gemäß dem COSMIC-Modell einteilen zu können.

Funktionale Dienst-Software-Packete, wie beispielsweise Datenbankmanagementsysteme, Betriebssysteme oder Gerätetreiber, sollten üblicherweise als in separaten Schichten lokalisiert betrachtet werden.

Einmal identifiziert kann eine Schicht als separate *Komponente* in eine Generische Software-Modellmatrix mit Kennzeichnung der Korrespondenzen eingeordnet werden (siehe Anhang A).

Beispiele derartiger physischer Strukturen typischer Software-Architekturschichten (wobei Schicht in dem hier betrachteten Sinne zu verstehen ist) unterstützen Business-Applikations-Software und verhelfen zu eingebetteten Echtzeit-Software wie sie in den Abbildungen 2.2 und 2.3 angegeben sind.

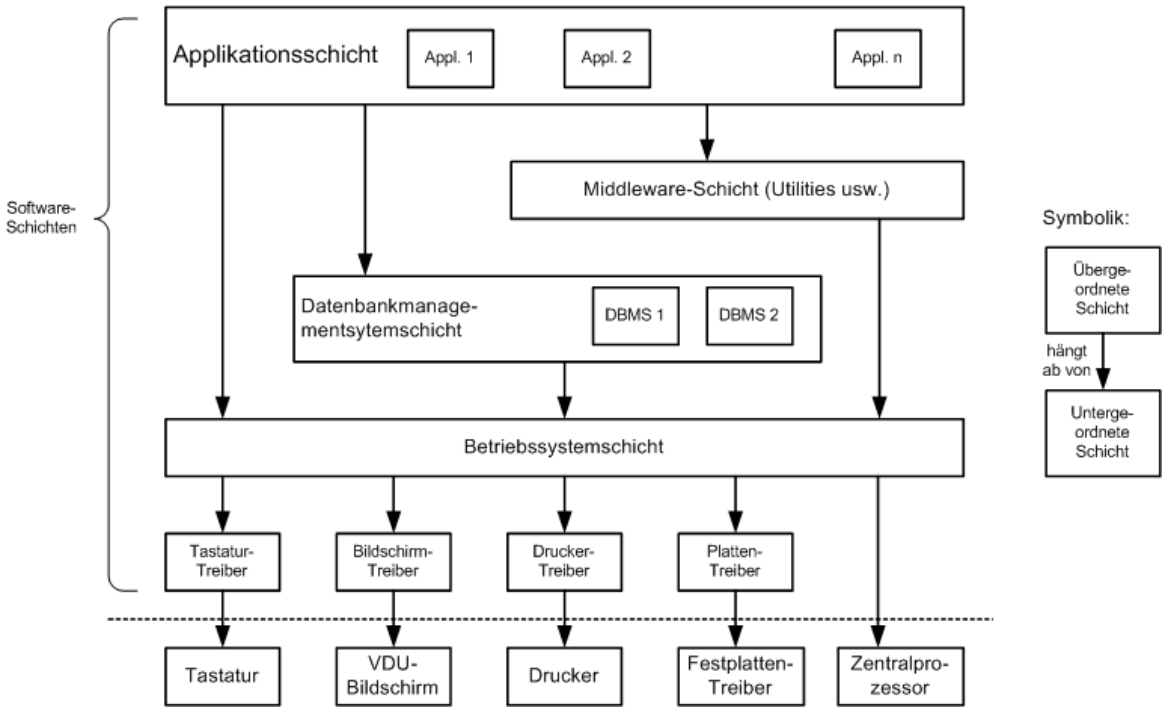

Abb. 2.2: Typische Software-Architekturschichten für Business-/MIS Applikationen

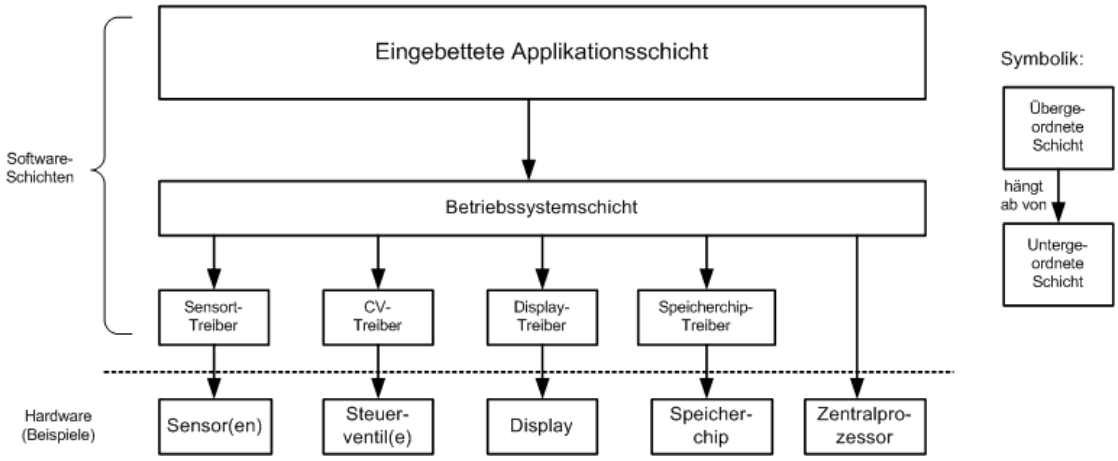

Abb. 2.3: Typische Software-Architekturschichten für eingebettete Echtzeitsysteme

2.2.5 Betrachtete Komponenten

Eine Betrachung bzw. der Foklus ist wie folgt definiert.

Definition der Betrachtetung selbst
Zwei Teile einer Software sind **Betrachtungen** (peers) derselben, wenn sie sich innerhalb einer Schicht befinden. Hinweis: Die beiden betrachteten Teile einer Software müssen nicht in einer Dekompositionsebene liegen.

Die betrachteten Komponenten können auf der Grundlage der folgenden Definitionen und Prinzipien identifiziert werden.

Definition der betrachteten Komponente
Eine betrachtete Komponente ist ein Bereich der funktionalen Einteilung der FUR eines Teils der Software innerhalb einer Schicht interagierender Mengen so dass dieser Bereich einen bestimmten Anteil der FUR dieses Softwareteils bildet. Hinweis: Der Ausdruck „betrachteter Teil einer Software“ wird benutzt, wenn irgendein unabhängiger Softwarebereich gemein ist.

Einmal identifiziert muss jeder Kandidat einer betrachteten Komponente folgende Prinzipien erfüllen:

Prinzipien der betrachteten Komponenten
a) In einer Menge betrachteter Komponenten eines Softwarebereichs in einer Schicht gibt es keine hierarchischen Abhängigkeiten zwischen diesen Komponenten als die durch die Schichten selbst bestimmt sind. b) Alle betrachteten Komponenten eines Softwarebereichs kooperieren miteinander in der Weise, dass der jeweilige Softwarebereich erfolgreich ausgeführt werden kann. c) Die FUR der betrachteten Komponenten einer Schicht, die gemeinsam Daten übertragen bzw. verteilen, definieren diese Daten identisch, d. h. sie erkennen dieselben Datenattribute und -strukturen dieser Daten, die sie weitergeben bzw. verteilen.

Einmal identifiziert kann auch jede betrachtete/gemessene Komponente als eine individuelle Komponente der Generischen Software-Modellmatrix (siehe Anhang A) mit einer referenzierenden Bezeichnung erfasst werden.

BEISPIEL: Wenn eine Business-Applikation in drei Hauptkomponenten entwickelt wurde und zwar als Nutzeroberflächenkomponente (user interface bzw. front-end), als Geschäftsregelkomponente (business rules) und als Datenservice-Komponente (data services). Dann sind dies die betrachteten Komponenten.

Hinweis: Zwei getrennte, aber zu betrachtende Teile der Software, die in der gleichen Schicht vorhanden sind und die Daten miteinander austauschen können durch eine Art alternativer Sequenz, die nach dem Prinzip (c) für zwei betrachtete Komponenten definiert sind, ersetzt werden.

Das folgende Diagramm zeigt eine Situation, die dem obigen Beispiel entspricht. Die gesamte Software befindet sich in ein und derselben Anwendungsschicht. Der Austausch zwischen den beiden betrachteten Komponenten der Applikation X und zwischen den beiden betrachteten Teilen der Software (Komponente 2 der Applikation X als auch der Application Y) kann über eine der beiden alternativen Sequenzen, die nach dem Prinzip (c) für zwei betrachtete Komponenten definiert sind, erfolgen.

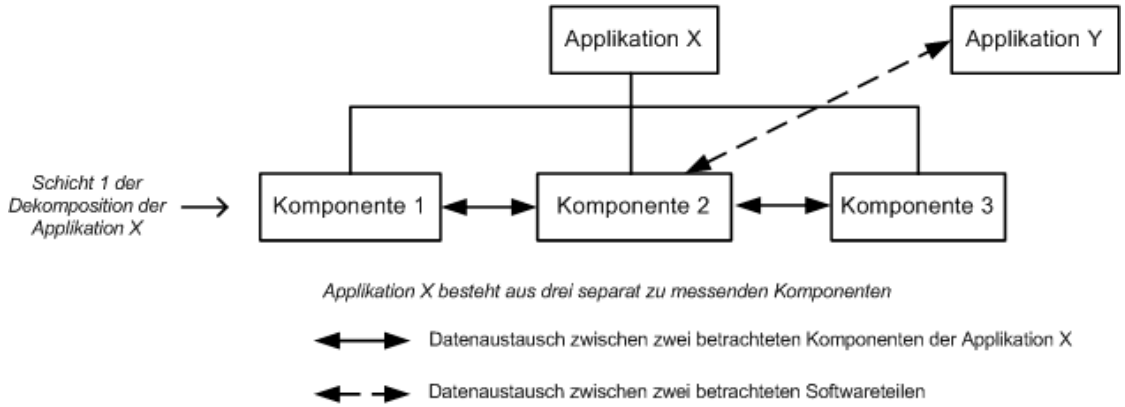

Abb. 2.4: Relationen zwischen den Konzepten der Betrachtung, einer Komponente und einer betrachteten Komponente

2.3 Bestimmung der Anwendungsfunktionen[2]

2.3.1 Variierender Funktionsumfang bezüglich der Anwendungsfunktion

Zu Beginn soll eine Analogie betrachtet werden: Die Messung der Grundfläche eines Büros kann unter vier verschiedenen Konventionen vorgenommen werden (Hinweis: der Messbereich (also das Büro) ist in allen vier Fällen derselbe.)

1. Der Gebäudebesitzer hat eine Gebühr für das Büro zu zahlen. Für den Gebäudebesitzer wird die Grundfläche in Gesamtquadratmeter aus externer Sicht bestimmt, die alle Flure und Wände bzw. Mauern mit einschließt.
2. Der Gebäudeheizungsmanager ist an den „Netto-Quadratmetern“ der internen Flächen interessiert einschließlich der öffentlichen und Fahrstuhlbereiche aber ausschließlich der Dicke der Wände.
3. Der beim Büromieter angestellte Reinigungsbeauftragte ist nur an den Quadratmetern interessiert, die die öffentlichen Bereiche ausschließen aber alle durch den Mieter benutzten Korridore beispielsweise einschließen.
4. Der Büroplanungsmanager wiederum ist an den Quadratmetern interessiert, die nur die Nutzungsfläche einschließt.

Diese Analogie zeigt, dass der Umfang eines Objektes in Abhängigkeit von den unterschiedlichen Interessen der Nutzer dieses Objektes, d. h. hinsichtlich der Anwendungsfunktion (functional user), variieren kann. Im Falle der Software können die Anwendungsfunktionen sehr unterschiedlich genutzt werden und daher zum Beispiel je nach Anwendungsart zu ganz unterschiedlichem Funktionsumfang führen.

2.3.2 Anwendungsfunktionen

Ein „Nutzer“ bzw. „Anwender“ ist definiert als „jemand oder etwas, dass mit der zu messenden Software interagiert“. Diese Definition ist allerdings für die Verwendung der COSMIC-Methode zu ungenau. Für die COSMIC-Methode ist die Auswahl des Nutzers bzw. der Nutzer durch die zu messenden funktionalen Anforderungen determiniert. Dieser Nutzer- bzw. Anwendungstyp ist daher wie folgt definiert.

Definition der Anwendungsfunktion
Ein Anwendungstyp ist ein Sender und/oder ein einbezogener Empfänger von Daten der Funktionalen Anwendungsanforderungen (FUR) eines speziellen Softwarebereichs (piece of software).

Für die Anwendung der COSMIC-Methode ist es wichtig, die Anwendungsfunktionen eines zu messenden Softwarebereichs von den anderen (nichtfunktionalen) Anforderungen zu unterscheiden bzw. zu trennen.

[2] Wir haben hier die passive Form „Anwendungsfunktion“ im Gegensatz zur aktiven Form des „Funktionalen Nutzers bzw. Anwenders“ in der englischen Vorlage verwendet, was allerdings die Gültigkeit der nachfolgenden Aussagen auch nicht verändert.

Beispiel 1: Die Anwendungsfunktion im Falle einer Business-Applikation ist normalerweise durch den Menschen bzw. Kunden bestimmt bzw. stellt im Falle anderer Applikationen deren Schnittstellen dar. Im Falle einer Echtzeitanwendung wird die Anwendungsfunktion normalerweise durch spezielle Hardware bzw. Geräte oder anderer Softwareschnittstellen bestimmt. Die funktionalen Nutzeranforderungen (FUR) derartiger Software sind normalerweise als Datensender und/oder Datenempfänger aus der bzw. zur Software zu verstehen.

Allerdings muss die Gesamtzahl der Nutzer bzw. Anwendungen, d. h. derjeniger, die mit der Software interagieren, auch das Betriebssystem enthalten. Diese Bedingung ist allerdings allen Applikationen gemeinsam, da sie normalerweise durch Compiler und Interpreter verarbeitet werden, was eigentlich als reale Anwendungsfunktion einer Applikation verborgen bleibt.

Beispiel 2: Betrachtet sei die Anwendungssoftware eines mobilen Telefons, welches bereits in der Einführung charakterisiert wurde. Auch wenn wir Betriebssystem-funktionen beim Mobilen Telefon ausschließen, könnte die Nutzung darin bestehen, dass a) ein Benutzer eine Taste drückt oder b) Geräte (z. B. der Bildschirm, die Tastatur usw.) und andere zu betrachtende Applikationen direkt mit dem Telefon interagieren könnten. Der menschliche Nutzer zum Beispiel möchte nur eine Teilmenge der unterschiedlichen Funktionalitäten sehen, welche für die Bedienung des Telefon (unbedingt) notwendig sind. Somit sehen zwei verschiedene Nutzer auch unterschiedliche Funktionalitäten. Der Umfang der FUR eines menschlichen Nutzers ist gewöhnlich kleiner als der Umfang der FUR, die für die Anwendung des Telefons insgesamt zu entwickeln sind.

Regeln zur Bestimmung der Anwendungsfunktion
a) Die Anwendungsfunktion eines zu messenden Softwarebereichs sollte durch den Zweck der Messung bestimmt werden. b) Wenn der Zweck der Messung eines Softwarebereichs auf den Aufwand für die Entwicklung oder Modifikation der Software ausgerichtet ist, sollten die Anwendungsfunktionen auch entsprechend jeweils die neue bzw. modifizierte Funktionalität darstellen.

Nach der Identifizierung der Anwendungsfunktionen ist unmittelbar die Abgrenzung (Boundary), die einfach zwischen dem zu messenden Softwarebereich und dem Benutzer liegt, zu bestimmen. Wir ignorieren dabei irgendwelche dazwischen liegende Hard- oder Software.

Definition der Abgrenzung
Eine Abgrenzung (Boundary) ist als eine konzeptuelle Schnittstelle zwischen der zu messenden Software und seinen Anwendungsfunktionen definiert. Hinweis: Die Abgrenzung eines Softwarebereichs ist die konzeptionelle Grenze dieses Bereichs und der Operationsumgebung wie sie sich extern aus der Sicht der Anwendungsfunktionen zeigt. Die Abgrenzung erlaubt der Softwaremess-person eindeutig zu unterscheiden, was sich innerhalb der zu messenden Software befindet und welches die Operationsumgebung darstellt.

Es sei darauf hingewiesen, dass der Begriff der Abgrenzung aus dem Standard IOC/IEC 14143/1:2007 entnommen wurde mit der Modifikation des „funktionalen“ hin zum „Nutzer“ bzw. zur Anwendungsfunktion. Um Mehrdeutigkeiten zu vermeiden, darf die Abgrenzung nicht mit einer (willkürlich) um die zu messende Software gezogenen Linie verwechselt werden. Die Abgrenzung (Boundary) ist auch nicht mit demMessbereich (Scope of a measurement) zu verwechseln.

Die folgenden Regeln können für die Bestimmung einer möglichen Abgrenzung sinnvoll sein.

Regeln für die Abgrenzung
a) Identifiziere die Anwendungsfunktion(en), die mit der zu messenden Software interagieren, b) Per Definition gibt es eine Abgrenzung zwischen jedem identifizierten Schichtenpaar, wo die Software einer Schicht die Anwenderfunktion der anderen Software darstellt und dann die letztere zu messen ist, c) Es gibt stets eine Abgrenzung zwischen zwei Softwarebereichen, in denen jeweils zwei Komponenten die gegenüberliegende Seiten darstellen. In diesem Fall kann jeder Softwarebereich und/oder jede Komponente eine Anwendungsfunktion der anderen darstellen.

Die Abgrenzung erlaubt eine klare Unterscheidung zwischen irgendeinem zu messenden Softwareteil (d. h. die Software innerhalb der Abgrenzung) und irgendeiner Anwendungsfunktionalität der Umgebung (d. h. die Anwendungsfunktionen außerhalb der Abgrenzung). Dabei wird die persistente Speicherung nicht als Anwendungsfunktion betrachtet und gehört zur Software innerhalb der Abgrenzung.

2.4 Bestimmung des Niveaus der Granularität

2.4.1 Die Notwendigkeit eines Standard-Niveaus der Granularität

Wenn beispielsweise ein Projekt mit dem Design und Konstruktion eines neuen Hauses startet, muss zunächst der Architekt einen „high level“ Plan zeichnen, der die Hauptmerkmale skizziert und nur wenig Details beinhaltet. Im Projektfortschritt – der eigentlichen Konstruktionsphase – sind für die Planung mehr Details (als „low level“) notwendig.

Das gleiche gilt auch für Software. Am Anfang sind die funktionalen Anforderungen (FUR) allgemein spezifiziert und beinhalten nur wenige Details. Im Projektverlauf werden die FUR verfeinert (z. B. in den Versionen 1, 2, 3 usw.) und zeigen mehr Details eines „niedrigeren“ Niveaus. Diese unterschiedlichen Detailgrade der FUR werden als unterschiedliche Granularitätsniveaus bezeichnet (Siehe auch die Beschreibung anderer Begriffe im Abschnitt 2.4.3, die durchaus gegensätzlich zum Konzept des hier definierten Granularitätsniveaus sein können).

Definition des Granularitätsniveau
Das Granularitätsniveau bzw. die Granularitätsstufe ist (irgend-) ein Grad des Ausmaßes der Beschreibung eines einfachen Softwarebereichs (d. h. die Angabe der Anforderungen oder die Beschreibung der Struktur des Softwarebereichs) in der Form, dass bei einer Niveauerhöhung dieses Ausmaßes der Funktionalität dieses Softwarebereichs sich auch das Detailniveau erhöht bzw. einheitlich ausrichtet. Hinweis: Die Softwaremesspersonen sollten darauf gefasst sein, dass im Falle der relativ frühen Änderung der Anforderungen im Softwareprojekt unterschiedliche Teile der geforderten Software-Funktionalität somit in unterschiedlichen Granularitätstufen dokumentiert vorliegen.

Ein Plan eines Hausbaus ist in unterschiedlicher Standardeinteilung gezeichnet und ist einfach in anderen Skalenformen maßstabsgetreu zu überführen. Im Gegensatz dazu existieren keine Standardeinteilungen für Software für die verschiedenen Granularitätsniveaus ihrer Spezifikation. Damit ist es schwer zu garantieren, dass zwei FUR-Beschreibungen dasselbe Granularitätsniveau besitzen. Ohne eine Übereinkunft zu einem Standard für das Messen (oder welche Messeinteilung zu verwenden ist) ist es unmöglich, eine vergleichbare Funktionsumfangsmessung zu gewährleisten. Außerdem haben die Messpersonen häufig ihre eigene Methodik der Skalenbestimmung für ein bestimmtes Granularitätsniveau entwickelt.

Um diese Probleme zu verdeutlichen betrachten wir eine weitere Analogie. Bei einer Anzahl von Straßenkarten werden im Allgemeinen drei Granularitätsniveaus unterschieden:

1. Karte A zeigt alle Verkehrswege als Autobahnen,

2. Karte B zeigt zu den Autobahnen auch noch die Bundesstraßen,

3. Karte C zeigt alle Straßen einschließlich ihrer Namen (beispielsweise als regionale bzw. Ortskarten).

Wenn wir die Tatsache unterschiedlicher Granularitätsniveaus außer Acht lassen, würden diese drei Karten unterschiedliche Umfänge eines Landesstraßennetzes dokumentieren. Natürlich versteht jeder die unterschiedlichen Niveaustufen und kann die unterschiedlichen (Straßennetz-) Umfänge jeweils klar erkennen. Die Abstraktion als unterschiedliche Granularitätsniveaus drückt sich in den verschiedenen Skalierungen dieser unterschiedlichen Karten aus.

Für die Software-Messung gibt es nur ein Standardniveau für die Granularität, das eindeutig definierbar ist. Das ist das Granularitätsniveau eines einzelnen, identifizierten Funktionsprozesses und dessen definierten Datenbewegungen. Die Messung sollte stets in diesem Niveau durchgeführt werden bzw. auf diese Basis überführt werden.

2.4.2 Die Klarstellung des Granularitätsniveaus

Für die weitere Beschreibung der Messstrategiephase insgesamt ist es notwendig, jedes Missverständnis in der Bedeutung des Granularitätsniveaus bei der COSMIC-Methode

auszuräumen. Wie wir bereits definiert haben, erweitert sich die Beschreibung der Software beim Übergang in ein weiteres Detaillierungsniveau – einschließlich eines Zoom-Effektes – ohne den Messbereich zu verändern. Die Prozess sollte ***nicht*** gleichgesetzt werden mit einem der folgenden Effekte:

- Vergrößerung des beschriebenen Software-Artefaktes, um unterschiedliche Teilmengen der Funktionalität für verschiedene Nutzer zu zeigen und somit die Funktionalität unzulässig zu verringern
- Erweiterung der Software-Darstellung, um weitere Komponenten, Teilkomponenten usw. (als unterschiedliche Stufen der Dekomposition – siehe auch Abschnitt 2.2.2 oben) darzustellen. Eine derartige Erweiterung kann notwendig werden, wenn der Zweck des Messens als Messbereich die technologische Struktur der Software erfordert.
- Veränderung der Softwarespezifikation im Verlauf der Entwicklung, d. h. Änderung der Anforderungen des logischen und physischen bzw. technologischen Design usw. Unabhängig vom Stand der Entwicklung sind wir nur an den FUR für den Messzweck interessiert.

Das Konzept des Granularitätniveaus wurde dafür entwickelt, ausschließlich die funktionalen Nutzeranforderungen von Software zu bestimmen. Andere Formen der Erweiterung oder des Zuschneidens oder der verschiedenen Beschreibungen von Software können ebenfalls für die Bestimmung des Funktionalen Umfangs von Bedeutung sein. Diese Aspekte werden jedoch in dem COSMIC-Dokument „Advanced and Related Topics, v3.0“ im Kapitel „Ensuring the Comparability of Size Measurements“ behandelt.

2.4.3 Das Standard-Niveau der Granularität

Das Standard-Niveau der Granularität für die Messung ist das Funktionsprozessniveau, welches wie folgt definiert ist.

Definition des Funktionsprozessniveaus der Granularität
Als Granularitätsniveau der Beschreibung bzw. Spezifikation der Funktionalität eines Softwareanteiles bzw. -bereichs • sind die Individuen (als Nutzer) oder technische Geräte oder (andere) Softwareteile (aber nicht Gruppierungen dieser) ***und*** • die das einfache Auftreten von Ereignissen, die diese Softwareteile zu liefern haben (und nicht irgendeine Stufe in der Zusammenfassungen von Ereignissen definiert sind), bestimmen. Hinweis 1: In der Praxis beschreiben die die funktionalen Nutzeranforderungen beschreibenden Dokumentationen die Funktionalität in unterschiedlichen Granularitätsstufen, insbesondere wenn diese Dokumentationen gerade entwickelt werden. Hinweis 2: Gruppierungen derartiger Nutzerfunktionen können beispielsweise in einer Abteilung auftreten, in denen die Mitarbeiter mit unterschiedlichen Funktionsprozessformen umzugehen haben oder bei einem Steuerpult mit unterschiedlichen Instrumententypen oder aber auch bei einem zentralen System.

Hinweis 3: Gruppierungen von ereignisbasierten FUR können beispielsweise in einer relativ hohen Granularitätsstufe durch einen Eingabestrom für ein Verwaltungssoftwaresystem für Verkaufstransaktionen oder bei einem Eingabestrom für ein Flugleitsystem für Pilotenanweisungen auftreten.

Auf der Grundlage dieser Definition können wir nun die folgenden Regeln und Anwendungsempfehlungen konstatieren.

Regeln für das Funktionsprozessniveau der Granularität

a) Funktionale Umfangsmessung sollte beim Granularitätsniveau der Funktionsprozessbeschreibung durchgeführt werden.

b) Für den Fall einer Funktionsumfangsbestimmung zu einer FUR für die die Funktionsprozesse noch nicht mit allen detaillierten Datenbewegungen definiert ist, sollte erst einmal die bis dahin vorliegende Funktionalität des Garnularitätsniveaus des Funktionsprozesses gemessen werden (siehe hierzu das COSMIC-Dokument „Advanced and Related Topics“ mit den Hinweisen zu Methoden einer näherungsweisen Umfangsbestimmung auf der Grundlage einer „frühen“ Schätzung im FUR-Bestimmungsprozess).

Neben diesen Regeln empfiehlt COSMIC[3], dass das Funktionsprozessniveau der Granularität stets in einer Richtung einheitlich auszurichten ist, der die Funktionsumfangsbestimmung dient bzw. die sie durch Benchmark-Dienst- und Software-Tool-Verteiber für die Funktionsumfangsmessung letztlich ausgerichtet ist, beispielsweise zur Aufwandschätzung von Projekten.

Beispiel: Funktionalität unterschiedlicher Granularitätsstufen

Dieses Beispiel ist aus dem Bereich der Business-Applikationen und soll ein Warenbestellsystem im Internet sein, welches wir Everest-System nennen wollen (Das COSMIC-Dokument „Advanced and Related Topics“ enthält ein Beispiel aus dem Bereich der Echtzeitsystemsoftware.). Die unten angegebene Beschreibung wurde für unseren Zweck der Granularitätsniveaubeschreibung stark vereinfacht. Diese Beschreibung deckt auch nur den Bereich der für den Everest-Systemnutzer verfügbaren Funktionalität ab. Das schließt zum Beispiel die Funktionalitäten aus, die für ein vollständiges Warenvertriebssystem erforderlich wären, wie die Unterstützung für die Everest-Mitarbeiter, für die Warenlagerung, für den Zahlungsverkehr usw.

Der Messbereich ist daher definiert als den „Teil der Everest-Applikation über den die Kunden über das Internet zugreifen“. Als Messzweck soll die Bestimmung des Funktionsumfangs der für den Kunden verfügbaren Applikationsteil angenommen werden. Als höchste „Granularitätsstufe 1“ der Beschreibung der FUR des

[3] Der Grund für eine Empfehlung zum Funktionprozessniveau der Granularität und nicht einer weiteren Regel liegt darin, dass es dem COSMIC-Methodenanwender letztlich überlassen bleibt, die jeweilige Tool-Unterstützung oder anderer Dienstanbieter in dieser Richtung zu wählen. Für diese erweiterte Community der Tool-Hersteller kann COSMIC lediglich eine Empfehlung aussprechen.

Gesamtsystems des Everest-Warenbestellsystems wäre dann eine mögliche Darstellung etwa die folgende.
„Das Everest-System muss dem Kunden ermöglichen, einen Auftrag für eine Ware zu öffnen, auszuwählen, zu bestellen und zu erhalten aus dem Everest-Produktkatalog, der auch Produkte anderer Anbieter enthält.“

Eine Erweiterung bzw. Detallierung dieser höchsten Stufe kann in der folgenden Abbildung als die nächst niedrigere Granularitätsstufe 2 mit vier Teilsystemen angenommen werden.

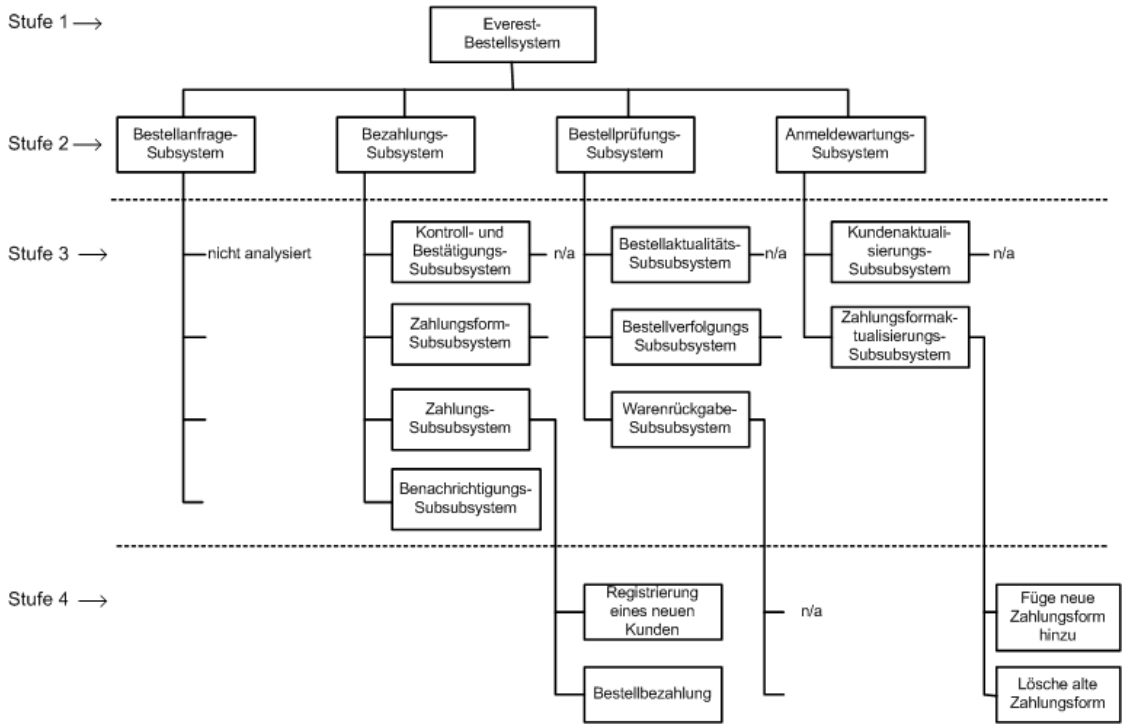

Abb. 2.5: Schichtenanalyse des Beispiels eines Everest-Bestellsystems

Diese vier Teilsysteme sind:

- Das *Bestellanfrage-Subsystem*, das ermöglicht, dass ein Kunde ein Produkt in der Everest-Datenbasis findet und auch seinen Preis, die Verfügbarkeit und das Hinzufügen eines ausgewählten Produktes in einen Warenkorb für den eigentlichen Kauf. Dieses Teilsystem unterstützt auch den Kauf von Sonderangeboten, gibt Bewertungen zu ausgewählten Waren an und ermöglicht allgemeine Anfragen zu Lieferzeiten usw. Es ist ein sehr komplexes Teilsystem und ist nicht weiter spezifiziert als es für dieses Beispiel notwendig ist.

- Das *Bezahlungs-Subsystem,* welches dem Kunden die Bestellung bestätigt und die Bezahlung der Waren aus dem Warenkorb ermöglicht.

- Das *Bestellüberprüfungs-Subsystem,* das dem Kunden über den Stand der Bestellung im Lieferprozess informiert, Änderung der Bestellung (z. B. eine andere Zustelladresse) oder die Rückgabe nicht zufrieden stellender Ware ermöglicht.

- Das *Anmeldewartungs-Subsystem,* welches verschiedene Details existierender Kunden pflegt, wie zum Beispiel die Heimatanschrift, die Zahlungsform usw.

Abbildung 2.5 deutet einige Details weiterer Verfeinerungen an für die nächsten beiden Granularitätsstufen für das Bezahlungssystem, das Bestellüberprüfungssubsystem und das Anmeldewartungssystem. Bei diesem Verfeinerungsprozess ist es wichtig zu beachten, dass

- nicht der zu messende Bereich der Applikation verändert wird,

- alle Beschreibungsstufen des Everest-Systems stets die verfügbare Funktionalitäten für den Kunden (als Nutzeranforderungen) aufzeigen. Ein Kunde muss stets die Systemfunktionalität in all diesen Stufen erkennen können.

Abbildung 2.5. verdeutlicht ebenfalls die Verfeinerung in weitere Granularitätsstufen dieser drei Subsysteme, wie beispielsweise spezielle Funktionsprozesse in Stufe 3 (für die beiden Bereiche des Bestellprüfungssubsystem) und in Stufe 4 für die beiden anderen Subsysteme. Dieses Beispiel zeigt, dass im Falle einer Funktionsbestimmung nach dem Top-Down-Ansatz nicht davon ausgegangen werden kann, dass die beschriebene Funktionalität einer spezielle Stufe stets mit derselben Granularitätsstufe übereinstimmt, wie es das Konzept der COSMIC-Methode definiert (Diese Definitionsform fordert, dass die Funktionalität in einer Granularitätsstufe stets ein vergleichbares Detailniveau besitzt.).

Weiterhin kommt es vor, dass Systemanalytiker die Diagramme unterschiedlich zeichnen, die jeweils unterschiedliche Funktionsgruppierungen in jedem Diagrammniveau aufzeigen. Das ist nicht der richtige Weg der Funktionsanalyse und –beschreibung derartiger komplexer Systeme.

Da diese Variationen unvermeidlich in der Praxis auftauchen, muss die Messperson genau die verschiedenen Stufen dieser Spezifikationsdiagramme prüfen, um den zu messenden Funktionsprozess exakt zu identifizieren. Wenn das in der Praxis nicht möglich ist, beispielsweise im Fall der noch nicht erreichten Stufen der Funktionsprozessbeschreibung, sollte die oben genannte Regel (b) angewandt werden. Um dies zu illustrieren wollen wir den Fall beim „Kundenaktualisierungs-Subsubsystem“ (siehe Abb. 2.5) im Zweig des Anmeldewartungs-Subsystem überprüfen.

Für eine erfahrene Messperson ist das Wort Wartung ein unveränderlicher Begriff, der eine Gruppe von Ereignissen und somit eine Gruppe von Funktionsprozessen einschließt. In diesem Sinne können wir sicherlich annehmen, dass dieses Wartungs-Subsubsystem die drei Funktionsprozesse, bezeichnet als „Anzeige der Kundendaten“, „Änderung der Kundendaten“ und „Lösche Kunden“, beinhaltet (Der Prozess „eröffne neuen Kunden“ muss offensichtlich auch existieren, aber dieser Prozess tritt in einem anderen Bereich des Systems auf, und zwar wenn ein Kunde Waren das erste Mal bestellt. Aber das ist außerhalb des Bereichs in diesem einfachen Beispiel.).

Eine erfahrene Messperson sollte in der Lage sein, den Umfang dieses Subsubsystems in der CFP-Einheit zu schätzen, in dem er die angenommene Funktionsprozessanzahl (drei in diesem Falle) aufnimmt und diese Zahl mit dem Durchschnittsumfang eines Funktionsprozesses multipliziert. Diesen Durchschnittsumfang erhält man durch Kalibrierung anderer Systemteile oder vergleichbarer Systeme. Beispiele für diesen Kalibrierungsprozess sind im COSMIC-Dokument „Avanced and Related Topics“ als näherungsweise Umfangsbestimmung beschrieben welches auch weitere Ansätze für diese Mess- bzw. Schätzform beinhaltet.

Natürlich haben diese Näherungsverfahren ihre Grenzen. Wenn wir einen derartigen Ansatz bereits in der Stufe 1 für die FUR-Bestimmung anwenden („Das Everest-System muss dem Kunden ermöglichen, einen Auftrag für eine Ware zu öffnen, auszuwählen, zu bestellen und zu erhalten . . .“) so würden wir nur wenige Funktionsprozesse identifizieren können. Eine weitere Spezifizierung würde erkennbar machen, das die Anzahl der Funktionsprozesse eines derartigen komplexen Systems viel größer sein muss. Das liegt daran, dass Funktionsumfänge gewöhnlich zu mehr Details führen als die beschriebenen Anforderungen erkennen lassen auch ohne Änderung der Messbereichs. Diese Näherungsmethoden müssen deshalb sehr sorgfältig und in einem möglichst hohem Granularitätsniveau angewandt werden, insbesondere gerade dann, wenn wenig Details verfügbar sind.

Das COSMIC-Dokument „Advanced and Related Topics“ beschreibt unterschiedliche Beispielarten der Umfangsbestimmung verschiedener Granularitätsstufen für den Fall in dem die Software Teil einer Architektur eines Telekommunikationssystems darstellt. Dieses Beispiel ist komplexer als das Everest-Beispiel bei dem die Nutzerfunktionen eines zu messenden Architekturteils auch alle anderen Softwareteile derselben Architektur mit einschließt. Ebenso enthält dieses Beispiel verschiedene Dekompositionsstufen der zu messenden Software und ihre dafür notwendigen Nutzerfunktionen.

2.5 Abschließende Bemerkungen zur Messstrategiephase

Die folgenden vier Elemente der Messstrategiephase sollten genau betrachtet werden, bevor die Messung selbst beginnt um auch ein nachvollziehbares Ergebnis der Umfangsbestimmung zu erreichen. Diese vier Elemente sind:

a) die genau Beschreibung des *Zweckes* der Messung,

b) die Festlegung des allgemeinen *Bereichs* zu messender Software und die separat zu messenden Teile, wobei auch die Stufen und zusammenhängenden Komponenten der Software-Architektur betrachtet werden müssen,

c) die Definition der *Nutzerfunktionen* des zu messenden Softwareteiles,

d) die Festlegung des *Granularitätsniveaus* der zu messenden Software-Artefakte und wie, wenn notwendig, der Umfang an einem einheitlichen Funktionsprozessniveaus der Granularität auszurichten ist.

Einigen Iterationen der Schritte (b) bis (d) werden notwendig sein, wenn die Anforderungen sich ändern bzw. neue Details eine Verfeinerung der Bereichsbestimmung erfordern.

Die Mehrheit funktionaler Umfangsbestimmung wird für den Zweck der Entwicklungsaufwandsbestimmung durchgeführt, d. h. für die Entwicklerleistung oder Projektabschätzung. Dafür sollte die Messstrategiebestimmung vor allem nachvollziehbar erfolgen. Der Zweck und Bereich sind gewöhnlich einfach zu bestimmen, die Nutzerfunktionen sind an der Systemnutzung auszurichten während das Granularitätsnieau der Messung sich an der Nutzerfunktion als einfache Ereignisse orientieren sollte.

Jedoch nicht alle Messungen sind nach einem gleichen Schema durchführbar, so dass die Messstrategie stets an der jeweiligen Situation auszurichten ist.

3. Die Abbildungsphase

Dieses Kapitel beschreibt die Regeln und die Methode des Abbildungsprozesses. Die allgemeine Methodik für die Abbildung der Software auf das Generische COSMIC-Software-Modell zeigt die folgende Abbildung 3.1.

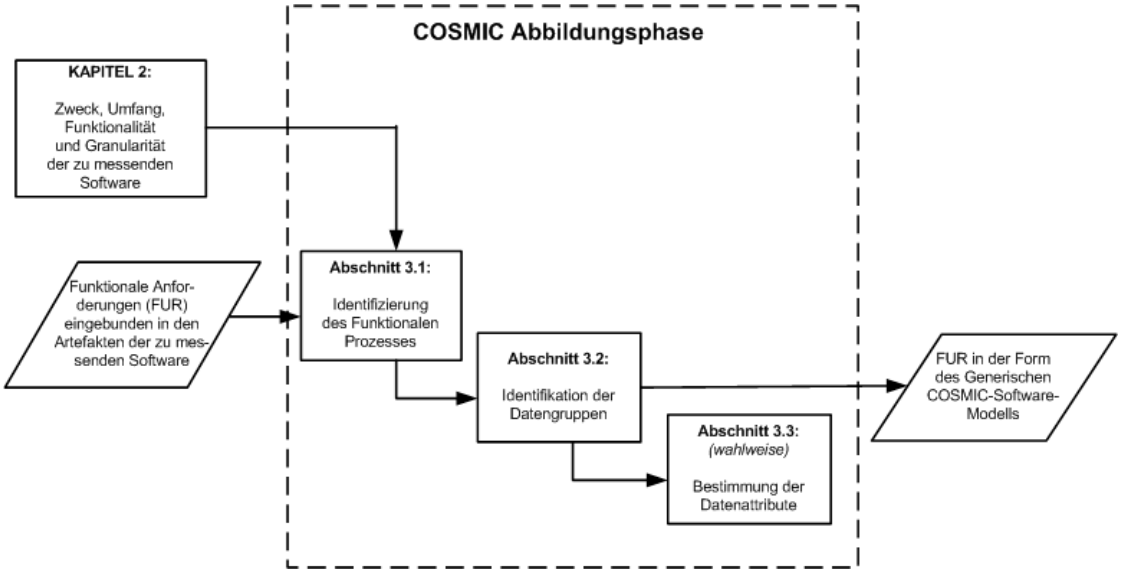

Abb. 3.1: Allgemeines Vorgehen beim COSMIC-Abbildungsprozess

Jeder Schritt dieser Methode ist Inhalt des jeweiligen Abschnittes (in der Abbildung entsprechend gekennzeichnet), wobei die Definitionen und Regeln einschließlich einiger unterstützender Prinzipien und Beispiele angegeben sind.

Die in der Abbildung 3.1 gezeigte allgemeine Methodik soll für einen möglichst großen Bereich an Software-Artefakten anwendbar sein. Ein mehr systembezogener und detaillierterer Prozess würde zwar präzise Abbildungsregeln für eine große Kollektion von spezifischen Artefakten bereitstellen, allerdings auch den Grad der Eindeutigkeit während der Bildung des Generischen COSMIC-Software-Modells verringern. Ein derartiger Prozess würde auch per Definition eine hohe Abhängigkeit von der Spezifik der Artefakte, die in hohem Maße auch von der softwaretechnischen Methodik der jeweiligen Organisation abhängen, bedeuten.

Das COSMIC-Dokument „Guideline for Sizing Business Application Software Using COSMIC“ gibt eine Anleitung für die Abbildung unterschiedlicher Methoden zur Bestimmung der Datenspezifikationen und Anforderungen nach dem Konzept der COSMIC-Methode, wie sie im Bereich der Business-Applikationen zur Anwendung kommen.

3.1 Die Anwendung des Generischen Software-Modells

Prinzipien für die Anwendung des Generischen COSMIC-Software-Modells
Das Generische COSMIC-Software-Modell sollte für jeden Software-Teil, für den ein separater Messbereich definiert wurde, auch separat auf die funktionalen Anforderungen angewandt werden. Die Anwendung des Generischen COSMIC-Software-Modells bedeutet auch, die Menge der auslösenden Ereignisse, die für jeden Nutzerfunktionstyp, die in der FUR bestimmt wurden und die dann den jeweiligen funktionalen Prozess, die relevanten Objekte, die Datengruppen und die für diese Ereignisse unterstützenden Datenbewegungen, zu bestimmen.

Die unten angegebenen Abbildungen 3.2 und 3.3 illustrieren die Anwendung des Prinzips (g) des Software-Kontextmodells und die Prinzipien des Generischen Software-Modells sowohl für den Business- als auch für den Echtzeitapplikationsbereich.

Während die Abbildungen 2.2 und 2.3 die aktuelle physische Sicht der Software-Schichtenarchitektur zeigen, geben die Abbildungen 3.2 und 3.3 die logische Sicht der unterschiedlichen Konzepte im COSMIC-Modell an. Bei dieser logischen Sicht interagieren die Benutzer mit der zu messenden Software entlang einer (System-) Abgrenzung über eine Eingangs- (Entry) und Ausgangsdatenbewegung (Exit). Die Software bewegt dann Daten von und zu einer persistenten Speicherung über eine Eingabe- (Read) und Ausgabedatenbewegung (Write). Bei dieser logischen Sicht wird die gesamte für diesen Austausch notwendige Hard- und Software in die Messung einbezogen, während der Benutzer und die persistente Speicherung hierbei ignoriert werden.

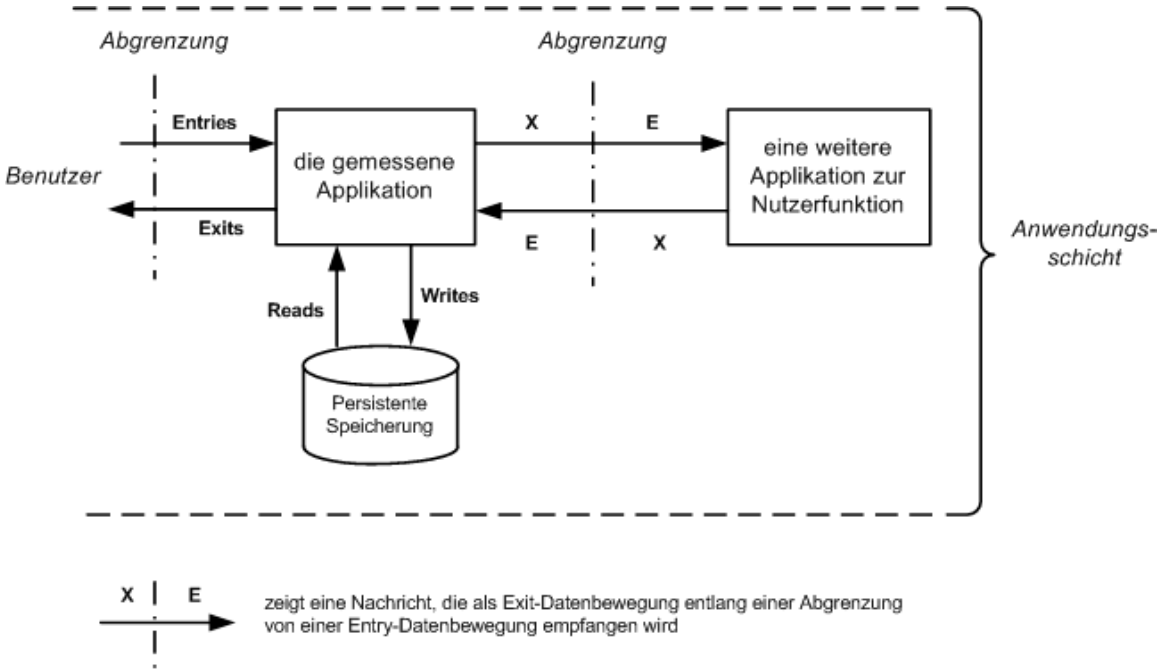

Abb. 3.2: Business-Applikation mit dem Nutzer und einer weiteren Applikation sowie deren Nutzerfunktionen aus logischer Sicht

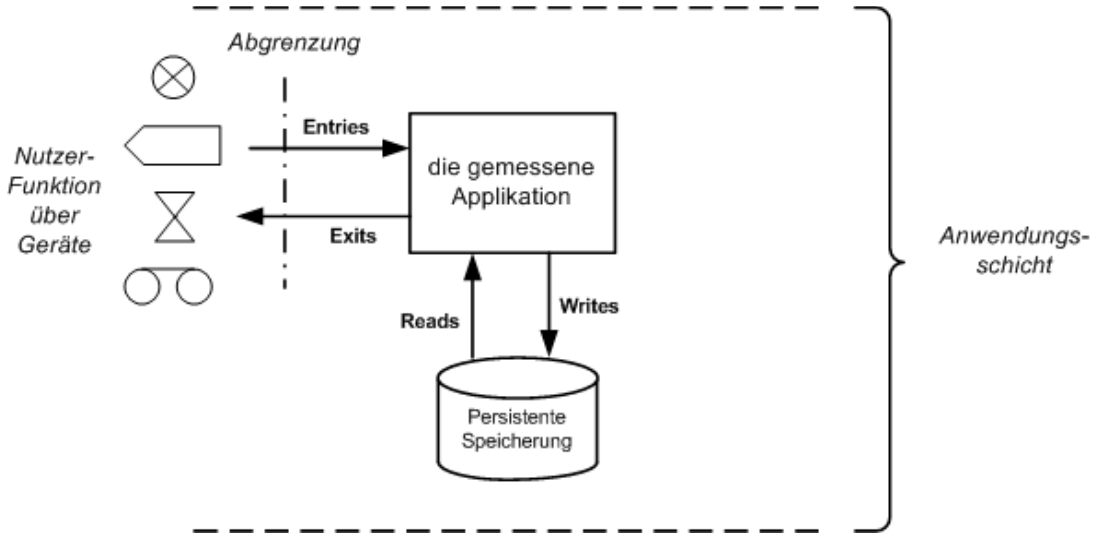

Abb. 3.3: Echtzeit-Software-Applikation mit unterschiedlichen Geräten sowie deren Nutzerfunktionen aus logischer Sicht

4.3 Die Identifizierung der funktionalen Prozesse

Dieser Abbildungsschritt besteht in der Identifikation der in dem zu messenden Software-Teil involvierten Menge an funktionalen Prozessen aus den funktionalen Anforderungen (FUR).

3.2.1 Definitionen

Definition des funktionalen Prozesses
Ein funktionaler Prozess ist eine elementare Komponente einer Menge funktionaler Anforderungen, die eine Menge eindeutiger, ganzheitlicher und unabhängiger Datenbewegungen einschließt. Sie wird ausgelöst durch eine Datenbewegung (einem Eingang (Entry)) von einem Benutzer, der einen Software-Teil darüber informiert, dass eine Nutzerfunktion ein auslösendes Ereignis identifiziert hat. Sie ist vollständig, wenn sie alle notwendigen Berechnungen für die Beantwortung des auslösenden Ereignisses erbracht hat. Hinweis: Zusätzlich zur Information des Software-Teiles bei dem das Ereignis aufgetreten ist kann der auslösende Eingang (Entry) Daten des einbezogenen Objektes, welches mit diesem Ereignis verbunden ist, enthalten.

Definition des auslösenden Ereignisses
Ein Ereignis bzw. irgendein Geschehnis, welches eine Nutzerfunktion eines Software-Teiles für die Initiierung („Auslösung“) eines oder mehrerer funktionaler Prozesse hervorruft. In einer Menge funktionaler Anforderungen kann jede einen funktionalen Prozess auslösende Nutzerfunktion - für diese FUR nicht weiter unterteilt werden, - eintreffen oder nicht eintreffen. Hinweis: Zeitbezogene Ereignisse können beispielsweise solche auslösenden Ereignisse sein.

Den Zusammenhang zwischen auslösendem Ereignis, Nutzerfunktion und dem die Entry-Datenbewegung auslösenden funktionalem Prozess ist in Abbildung 3.4 skizziert. Die Interpretation dieses Diagramms lautet: *ein Ereignis wird durch eine Nutzerfunktion ausgelöst und diese Nutzerfunktion wiederum löst einen funktionalen Prozess aus.*

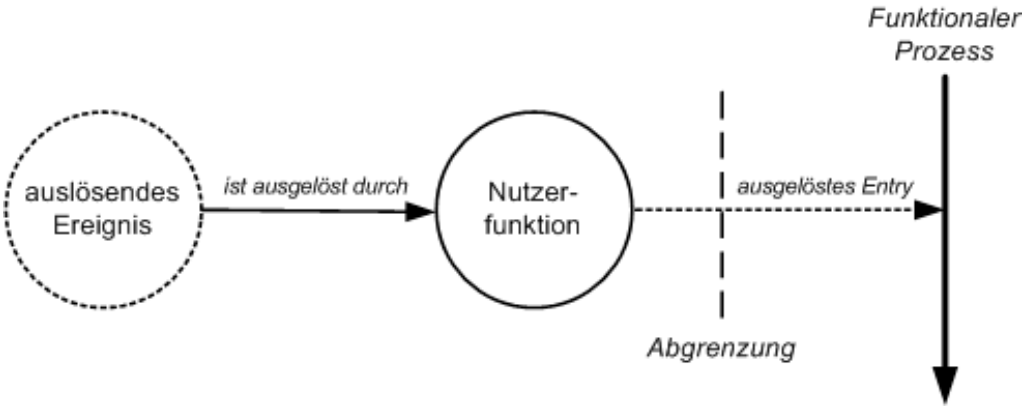

Abb. 3.4: Zusammenhang zwischen auslösendem Ereignis, Nutzerfunktion und funktionalem Prozess

Der auslösende Eintritt (Entry) ist normalerweise eine positive, eindeutige Nachricht, die die Software darüber informiert, dass eine Nutzerfunktion ein auslösendes Ereignis identifiziert hat. Dieser auslösende Eingang enthält oftmals auch Daten, die mit einem einbezogenen Objekt zusammenhängen. Nachdem der auslösende Eingang empfangen wurde, kann ein funktionaler Prozess weitere Eingänge (Entries), die weitere/andere einbezogene Objekte beschreiben, empfangen.

Wenn eine Nutzerfunktion inkorrekte Daten sendet, beispielsweise wegen einer Sensor-Nutzerfehlfunktion oder einer falschen Benutzereingabe, ist es normalerweise Aufgabe des funktionalen Prozesses zu bestimmen, ob das Ereignis wirklich aufgetreten ist, die Daten valide sind oder wie darauf zu reagieren ist.

3.2.2 Der Identifikationsansatz für die funktionalen Prozesse

Der Identifikationsansatz für die funktionalen Prozesse hängt von den Software-Artefakten ab, die dem Messpersonal zu Verfügung stehen sowie vom Zeitpunkt der Messung im Software-Lebenszyklus und von der für die Analyse, dem Design und der Implementation zugrunde liegenden Methode. Da insbesondere das Letztere in der jeweiligen Software-Domäne enorm variiert, ist der Sinn dieses Messhandbuches einige allgemeine Vorgehensweisen für die Bestimmung der funktionalen Prozesse bereitzustellen.

Die „Guideline for Sizing business application software using COSMIC" auch im Abschnitt 4.4 behandelt, enthält weitere Regeln und Beispiele wie die funktionalen Prozesse zu identifizieren bzw. zu unterscheiden sind für den Bereich der Business-Software-Applikationen.

Der wichtigste und allgemeinste Hinweis ist der, dass es stets sinnvoll ist, separate Ereignisse im Bereich der Nutzerfunktionen derart zu identifizieren, dass diese die Software „beantworten" muss, da jedes dieser Ereignisse bzw. Ereignistypen gewöhnlich zu einem oder mehreren funktionalen Prozessen bzw. Prozesstypen führt. Ereignisse können als Zustandsdiagramme oder so genannten Life-Cycle-Entity-Diagrammen bestimmt werden, da jede Transition die Software mit einem Ereignis verbindet.

Für die zuverlässige Identifikation von möglichen funktionalen Prozessen sollten die folgenden Regeln Anwendung finden.

Regeln für die Festlegung des funktionalen Prozesses
a) Ein funktionaler Prozess sollte von der weitestgehend identifizierbaren funktionalen Anforderung im zugestandenen Messbereich abgeleitet werden.
b) Ein funktionaler Prozess (-Typ) bearbeitet werden, wenn ein auslösendes Ereignis bzw. Ereignistyp auftritt.
c) Ein spezifisches Ereignis oder Ereignistyp sollte stets ein oder mehrere funktionale Prozess bzw. Prozesstypen auslösen, die auch parallel ausgeführt werden können. Ein spezifischer funktionaler Prozess oder Prozesstyp kann auch durch mehr als ein Ereignis bzw. Ereignistyp ausgelöst werden.
d) Ein funktionaler Prozess sollte mindestens zwei Datenbewegungen einschließen, einen Eingang (Entry) und einen Ausgang (Exit) oder einer Ausgabe /Write.
e) Ein funktionaler Prozess sollte sich vollkommen im Messbereich eines Software-Teils und in einer und nur einer Schicht befinden.
f) Im Kontext von Echtzeitanwendungen sollte ein funktionaler Prozess bei einem selbstinduzierten Wartezustand auch terminieren (d. h. der funktionale

Prozess hat die Anforderungen als Antwort auf ein auslösendes Ereignis erfüllt und wartet bis zum Auftreten eines nächsten auslösenden Eingangs).

g) Ein funktionaler Prozess bzw. Prozesstyp sollte so bestimmt werden, dass die FUR auch gestatten, dass ein funktionaler Prozess in unterschiedlichen Teilmengen mit maximalem Input von Datenattributen und ebenso durch derartige Variationen und/oder Unterschiede der Eingabedatenwerte zu unterschiedlichen Prozesswegen innerhalb des funktionalen Prozesses führen kann.

h) Separate Ereignisse bzw. Ereignistypen und die dazugehörigen funktionalen Prozesse bzw. Prozesstypen sollten in folgende Weise unterscheiden:

- wenn Entscheidungen in unterschiedlichen Ereignissen ergeben, dass sie zeitunabhängig sind (d. h. ob heute oder später eine Datenanforderung bestätig würde, sollte diese Anforderung auch eine separate Entscheidung erfordern bzw. sollte dies unterschiedliche funktionale Prozesse erkennen lassen),

- wenn die Verantwortlichkeiten für die Aktivitäten separat vorliegen (d. h. in einem Personalsystem, wo die Verantwortlichkeit für die Wartung von Grundpersonaldaten separat zur Verantwortlichkeit für die Wartung der Gehaltsdaten ist, liegen unterschiedliche funktionale Prozesse vor; oder für ein implementiertes Software-Paket, bei der die Funktionalität dem Systemadministrator für die Wartung der Paketparameter verfügbar ist, im Gegensatz zur separaten Verfügbarkeit der Funktionalität für den eigentlichen Benutzer).

4.3.1 Auslösende Ereignisse und funktionale Prozesse in Business-Applikationen

a) Auslösende Ereignisse einer online-Business-Applikation treten gewöhnlich im Bereich der die Applikation nutzenden Personen (als Benutzer) auf. Der Benutzer löst das Auftreten eines Ereignisses durch die Eingabe von Daten zu diesem Ereignis aus.

Beispiel 1: In einer Firma wurde ein Auftrag erhalten (als auslösendes Ereignis), welches einen Mitarbeiter erfordert (als Nutzerfunktion), um die Auftragsdaten einzugeben (als auslösender Eingang als Datenerfassung zum interessierenden Objekt eines Auftrags) mit der ersten Datenbewegung der „Auftragseingabe“ als funktionalen Prozess.

b) Manchmal kann zu einer Applikation A eine zugeordnete Applikation B auftreten, die das (Über-) Senden von Daten oder das Empfangen von Daten von der Applikation A

erfordert. In diesem Fall, wenn die Applikation B einen funktionalen Prozess auslöst, wenn es notwendig ist, Daten zu senden oder von der Applikation A zu empfangen, stellt die Applikation B die Nutzerfunktion zur Applikation dar.

Beispiel 2: Angenommen, dass der Auftrag in Beispiel 1 eine Bestätigung erfordert, dann muss die Auftragsapplikation dem Kunden Details zur zentralen Kundenregistrierungsapplikation senden, die zu messen ist. Jetzt wird die Auftragsapplikation die Nutzerfunktion gegenüber dieser zentralen Applikation. Die Auftragsapplikation nimmt das Ereignis der Bestätigung der Kundendaten auf und löst den funktionalen Prozess in der zentralen Kundenregistrierungsapplikation für die Speicherung dieser Daten aus, in dem es Daten zum interessierenden Objekt – dem Kunden – als auslösenden Eingang für die zentrale Applikation sendet.

c) Dies ist kein Widerspruch zum Prinzip der Spezifikation des funktionalen Prozesses, ob er nun online oder im Batch-Prozess erfolgt. Die nächtliche Bearbeitung ist eine technische Implementationsentscheidung, die erst dann als erfolgreiches Ereignis vorliegt, wenn dieser nächtliche Bearbeitungsprozess vollständig abgeschlossen ist. Eine Batch-Prozessfolge besteht aus einem oder mehreren funktionale Prozessen bzw. Prozesstypen, wobei jeder als „end-to-end“ spezifiziert werden sollte, unabhängig von anderen funktionalen Prozessen in derselben Folge. Jeder funktionale Prozess hat seine eigene auslösende Eingabe, die zu messen ist.

Beispiel 3: Unter der Annahme, dass der Auftrag im Beispiel 1 online eingegeben wird, wird dieser auch für eine mögliche Übernacht-Batch-Verarbeitung gespeichert. Die Nutzerfunktion wird durch eine Person, welche den Auftrag online erteilt, vorgenommen; die auslösende Eingabe ist schlechthin der Auftrag selbst. Es gibt einen funktionalen Prozess für die Eingabe und die Bearbeitung des Auftrages.

d) Periodische Signale von einer Uhr (als Zeitangaben) können physikalisch einen funktionalen Prozess auslösen.

Beispiel 4: Angenommen, dass eine FUR für einen Jahresend-Batch-Prozess, der die Business-Jahresergebnisse zeigt und sich auf den Start für das neue Jahr einstellt, zu bestimmen ist. Technisch besteht ein durch ein Betriebssystem ausgelöstes Zeitsignal für diesen Batch-Prozess aus einem oder mehreren funktionalen Prozessen. Derartige funktionale Prozesse in einer Folge, die Eingaben enthält, sollten ganz normal betrachtet werden (d. h. die Eingaben für irgendeinen funktionalen Prozess bestehen aus einem oder mehreren Entries bei der der erste den auslösenden Eingang darstellt).

Ein anderer Fall ist die Annahme eines speziellen funktionalen Prozesses ohne Eingabeanforderung für diese Berichtsausgabe. Technisch muss hier der Benutzer das auslösende Ereignis für diesen funktionalen Prozess dem Betriebssystem mitteilen. Da jeder funktionale Prozess ein auslösendes Ereignis besitzen muss, können wir davon ausgehen, dass das Jahresend-Zeitsignal diese Rolle für diesen Prozess übernimmt. Dieser funktionale

Prozess kann dann etliche Reads und viele Exits für die Erzeugung dieses Berichts erfordern. Aus logischer Sicht unterscheidet sich die Analyse dieses Beispiels nicht, wenn der Benutzer die Erzeugung eines oder mehrerer Berichte per Mausklick innerhalb eines Online-Menüs auslöst und danach dem Betriebssystem die Berichtserstellung in einem Batch-Stream zukommen lässt.

e) Ein einzelnes Ereignis kann durch ein oder mehrere funktionale Prozesse, die unabhängig voneinander ablaufen, ausgelöst werden.

Beispiel 5: Ein Wochenend-Zeitsignal kann den Start einer Berichterstellung auslösen als auch den Ablauf einer Zeiteinheit innerhalb eines Workflow-Prozesses.

f) Ein einfacher funktionaler Prozess kann durch mehr als einen Typ eines auslösenden Ereignisses initiiert werden.

Beispiel 6: In einem Banksystem kann eine Organisationsanweisung oder eine spezielle Kundenanforderung einen Monatsendbericht auslösen.

Zu weiteren Beispielen unterschiedlicher, auslösender Ereignisse und funktionaler Prozesse in so genannten Batch-Streams siehe die „Guideline for Sizing Business Application Software using COSMIC“ im Abschnitt 4.6.3.

3.2.4 Auslösende Ereignisse und funktionale Prozesse in Echtzeitanwendungen

a) Ein auslösendes Ereignis wird typischerweise durch einen Sensor wahrgenommen.

Beispiel 1: Wenn beispielsweise die Temperatur einen speziellen Wert erreicht hat (als auslösendes Ereignis), sendet der Sensor (als Benutzer) ein Signal (als auslösende Eingabedatenbewegung) für die Änderung des Warnlichts (als funktionalen Prozess).

Beispiel 2: Ein Militärflugzeug hat beispielsweise einen Sensor für das Ereignis eines Abschusses. Der Sensor stellt dann den Benutzer der eingebetteten Software dar, der eine Bedrohung erwidert. Bei dieser Software tritt ein Ereignis nur auf, wenn ein Sensor etwas bemerkt, wobei dieser Sensor (als Benutzer) die Software durch die Sendung einer Nachricht (auslösendes Entry) informiert, z.B. „Sensor 2 hat Abschuss bemerkt“, zuzüglich weiterer Daten über die Geschwindigkeit und die Koordinaten. Der Abschuss ist dann das interessierende Objekt.

b) Periodische Signale einer Uhr (als ein Zeitticker) können einen funktionalen Prozess auslösen.

Beispiel 3: In einigen Steuerungssoftware-Prozessen kann ein Moment (auslösendes Ereignis) eines Zeitgebers (Benutzer) ein Signal senden (als auslösendes Entry) als Bit-Information für einen funktionalen Prozess für die Wiederholung normaler Steuerzyklen. Der funktionale Prozess liest dann verschiedene Sensoren hinsichtlich des Empfangs von Daten zum interessierenden Objekt und wählt die jeweils benötigte Aktion aus. Dabei gibt es keine anderen Daten zu diesem Zeitgeber.

c) Ein einfaches Ereignis kann ein oder mehrere funktionale Prozesse, die unabhängig der parallel bearbeitet werden auslösen.

Beispiel 4: Eine Gefahrensituation in einer nuklearen Anlage kann unabhängige funktionale Prozesse in unterschiedlichen Bereichen, wie Drosseln der Steuerung, Starten einer Notkühlung, Schließen der Ventile, Auslösen des Bedieneralarms usw., auslösen.

d) Ein einfacher funktionaler Prozess kann mehr als ein auslösendes Ereignis initiieren.

Beispiel 5: Die Rücknahme des Schubs in einem Flugzeug kann entweder durch den Bodensensor oder durch ein Pilotenkommando ausgelöst werden.

3.2.5 Mehr über die Identifikation funktionaler Prozesse

Software unterscheidet Ereignisse und erfüllt die damit verbundenen funktionalen Prozesse ausschließlich über seine FUR. Wenn der Software-Umfang bestimmt werden soll, ist es manchmal schwierig zu entscheiden, welches die jeweils separaten Ereignisse sind, die notwendigerweise zu betrachten sind. Das ist insbesondere dann der Fall, wenn die ursprünglichen FUR nicht mehr verfügbar sind und wo zum Beispiel der Entwickler es für effizienter hielt, grundlegende Anforderungen zusammenzufassen. Dabei kann es hilfreich sein, die Datenorganisation der Eingaben (s. u.) zu prüfen oder das Menü der installierten Software als Hilfe für die Unterscheidung separater Ereignisse, die die Software für den jeweiligen funktionalen Prozess bedienen muss, auszuwerten.

Beispiel 1: Wenn es eine funktionale Nutzeranforderung zur Steuererleichterung für ein zusätzliches Kind und ebenso für die Steuerberechnung bei niedrigem Einkommen gibt, so sind dies Anforderungen für die Software zur Reaktion auf zwei unterschiedliche Ereignisse, die auch für den Benutzer separat auftreten können. Daher sollten auch zwei unterschiedliche funktionale Prozesse auch bei einer Steuerberechnung, deren Daten beide Fälle abdeckt, zugrunde gelegt werden.

Beispiel 2: Wenn das Einkommen einer Instanz (für eine Person) die Steuererleichterungsgrenze überschreitet und in einer anderen Instanz (für eine andere Person) nicht überschreitet, führt dieser Unterschied nicht zu zwei verschiedenen funktionalen Prozessen, sondern stellt separate Bedingungen für einen einfachen funktionalen Prozess dar.

Beispiel 3: Als Beispiel für die Anwendung der Regel (g) gilt, wenn ein spezielles Ereignis eines Eingangs (Entry) einer Datengruppe bestehend aus den Datenattributen A, B und C und die FUR es gestattet, dass auch beim selben Entry eine Datengruppe bestehend nur aus den Datenattributen A und B auftreten, so führt das nicht zu zwei funktionalen Prozesse bzw. –typen. Hierbei sind nur ein Entry und ein funktionaler Prozess zu identifizieren als Bewegung bzw. Manipulation der Datenattribute A, B und C.

Einmal identifiziert kann jeder funktionale Prozess in einer separaten Zeile registriert werden in der zutreffenden Schicht oder Komponente in der Matrix des Generischen Software-Modells (Anhang A) mit den entsprechenden Kennzeichnungen.

3.2.6 Funktionale Prozesse von Komponentenpaaren

Wenn es der Zweck einer Messung ist, jede Komponente separat zu messen, so ist für jede Komponente auch ein separater Messbereich zu definieren. In diesem Fall erfolgt die Umfangsbestimmung eines funktionalen Prozesses für jede Komponente nach den üblichen Regeln wie sie bereits beschrieben sind.

Bezüglich der Vorgehensweise für jede Messung (Messbereich definieren, dann Benutzer und Abgrenzung bestimmen usw.) gilt, dass der zu messende Software-Teil aus zwei oder mehreren Komponentenpaaren besteht, bei der es keine Überlappung zwischen dem Messbereich jeder Komponente gibt. Der Messbereich einer jeden Komponente muss als vollständige Menge funktionaler Prozesse definiert werden. So kann beispielsweise ein funktionaler Prozess nicht unterschiedlichen Messbereichen angehören. Ebenso hat ein funktionaler Prozess in einem Messbereich auch keinerlei Kenntnis über funktionale Prozesse in anderen Komponenten, auch wenn zwei Komponenten Nachrichten austauschen.

Der/die Benutzer einer jeden Komponente werden durch Analyse bestimmt dort wo ein Ereignis einen funktionalen Prozess der analysierten Komponenten initiiert. (Auslösende Ereignisse können nur im Bereich der Benutzer auftreten.)

Abbildung 4.4 zeigt funktionale Prozesse von Komponentenpaaren und die dabei ausgetauschten Datenbewegungen.

3.3 Die Identifizierung der interessierenden Objekte und Datengruppen

3.3.1 Definitionen und Prinzipien

Dieser Schritt besteht in der Bestimmung der Datengruppen, die durch einen zu messenden Software-Anteil referenziert werden. Für die Identifizierung dieser Datengruppen, beispielsweise für den Business-Applikationsbereich, ist es hilfreich, die zu interessierenden Objekte und wahrscheinlich auch ihre Attribute zu bestimmen. Datengruppen werden innerhalb von Datenbewegungen, die im nächsten Kapitel behandelt werden, übertragen.

Definition des interessierenden Objektes
Stellt irgendein identifiziertes „Ding“ aus Sicht der Funktionalen

Nutzeranforderungen dar. Es kann sich dabei um ein physisches Ding, als auch um ein konzeptuelles Objekt oder gar Teil eines konzeptuellen Objektes aus dem Benutzerbereich handeln, mit dem die Software arbeitet bzw. über den sie Daten speichert.

Hinweis: Bei der COSMIC-Methode wird der Begriff „interessierendes Objekt“ verwendet, um spezielle Termini softwaretechnischer Methoden zu vermeiden. Bei diesem Begriff handelt es sich nicht um das „Objekt“ im Sinne der objektorientierten Methoden.

Definition einer Datengruppe

Eine Datengruppe ist eine wohlunterschiedene, nichtleere, ungeordnete und redundanzfreie Menge von Datenattributen, wobei jedes enthaltene Datenattribut einen komplementären Aspekt ein und desselben interessierenden Objektes beschreibt.

Definition der persistenten Speicherung

Eine persistente Speicherung ist eine Speicherungsform, die ein Speichern im Verlauf eines funktionalen Prozesses gewährleistet und/oder eine durch einen anderen funktionalen Prozess gespeicherte Datengruppe, die auch bereits zuvor im selben oder anderen funktionalen Prozess gespeichert wurde, empfängt.

Hinweis 1: Im COSMIC-Modell befindet sich eine persistente Speicherung stets auf der anderen, vom Benutzer abgewandten Seite einer Abgrenzung.

Hinweis 2: Ein Beispiel irgend eines anderen Prozesses könnte eine ausschließliche Lese-Speicherungsform darstellen.

Einmal identifiziert, muss die bestimmte Datengruppe folgende Prinzipien erfüllen:

Prinzipien zur Datengruppe

a) Jede identifizierte Datengruppe sollte eindeutig und wohlunterschieden hinsichtlich seiner Datenattribute sein.

b) Jede Datengruppe sollte sich direkt auf ein interessierendes Objekt aus den funktionalen Nutzeranforderungen beziehen.

c) Eine Datengruppe sollte innerhalb eines die jeweilige Software unterstützenden Computersystems „materialisiert“ (s. u.) sein.

Einmal identifiziert, kann jede Datengruppe in einer eigenen Spalte mit einer referenzierenden Kennzeichnung der Generischen Software-Modellmatrix eingetragen sein (siehe Anhang A).

3.3.2 Über die Materialisierung der Datengruppen

In der Praxis kann die Materialisierung einer Datengruppe beispielsweise folgende Formen annehmen:

a) Als eine physische Datensatzstruktur auf einem persistenten Speichergerät (als Datei, Datenbanktabelle, ROM-Speicherung usw.).
b) Als eine physische Struktur innerhalb eines temporären Speichers (als dynamischer Datenstrukturbezug oder als initial referenzierter Speicherblock).
c) Als eine Darstellungsgruppierung von funktional zusammenhängenden Datenattributen eines Ein-/Ausgabe-Gerätes (Bildschirminhalt, Druckliste, Steuer-Display usw.).
d) Als Übertragungsnachricht zwischen peripheren Geräten und dem Computer der über einem Netzwerk usw.

3.3.3 Über die Identifizierung der interessierenden Objekte und Datengruppen

Die Definition und die zugehörigen Prinzipien eines interessierenden Objektes und den Datengruppen sind weit gefasst, um für einen größtmöglichen Softwarebereich angewandt werden zu können. Jedoch ist die Qualität der dabei erreichten Ergebnisse bei der Anwendung der Definition und der Prinzipien für einen speziellen Software-Anteil manchmal schwierig umzusetzen. Daher sind im Folgenden Beispiele beschrieben, die die Anwendung dieser Prinzipien für die spezielle Situation unterstützen sollen.

Im Falle einer notwendigen Analyse einer Datengruppe bei der sich die Datenattribute innerhalb oder außerhalb eines funktionalen Prozesses bewegen oder zu bzw. von einem persistenten Speicher bewegt werden, ist es besonders wichtig zu entscheiden, dass die Attribute der Daten sämtlichst zu einem interessierenden Objekt gehören, da letztlich die Anzahl separater Datengruppen gemäß der COSMIC-Methode zu bestimmen sind. Zum Beispiel, wenn das Datenattribut den Input eines funktionalen Prozesses darstellt und die Attribute zu drei unterschiedlichen, interessierenden Objekten gehören, müssen wir auch drei separate Eingaben (Entry) als Datenbewegungen identifizieren.

Interessierenden Objekte und Datengruppen im Business-Applikationsbereich

Beispiel 1: Im Anwendungsbereich einer Business-Software könnte ein interessierenden Objekt ein „Angestellter“ (physisch) oder eine „Rangfolge“ (konzeptuell) sein, angenommen die Software soll Daten über Angestellte oder einer Rangfolge speichern. Im Falle einer Rangfolge kann gleichzeitig aus den mehrzeilig geordneten FUR zwei interessierende Objekte abgeleitet werden: die Rangfolge und die Ordnungszeile. Die korrespondierenden Datengruppen könnten dann als „Ordnungsdaten“ und „Ordnungszeilendaten“ bezeichnet werden.

Datengruppen werden gebildet, wo immer es einen unmittelbaren Anlass einer Nachfrage nach Daten über irgendein „Ding“ gibt, die nicht persistent gespeichert sind, welche allerdings von persistenten Speicher abgeleitet werden könnten. Die Eingabedatenbewegung (Entry) einer unmittelbaren Anfrage (als Parameterauswahl für die Herleitung der erforderlichen Daten) und die Ausgabedatenbewegung (als Exit, das die notwendigen Attribute enthält) sind beides Datengruppen eines derartigen „Dinges“. Es handelt sich hierbei um temporäre Datengruppen, die nach den Berechnungen im funktionalen Prozess nicht mehr vorhanden sind. Es sind allerdings reale Datengruppen, da sie die Abgrenzung zwischen der Software und den Benutzern passieren.

Beispiel 2: Gegeben sei eine unmittelbare Anfrage zu einer Personaldatenbasis als Erstellung einer Namensliste der Angestellten mit einem Alter von über 35 Jahren. Die Eingabe (Entry) bewegt eine Datengruppe, die die Auswahlparameter enthält. Die Ausgabe (Exit) bewegt eine Datengruppe, die das einfache Attribut „Name“ des interessierenden Objektes als „alle Angestellten über 35 Jahre“ beinhaltet. Dabei ist es wichtig, bei der Beschreibung des funktionalen Prozesses den Namen der temporären Datengruppe bezüglich des interessierenden Objektes eindeutig anzugeben sowie auch den Bezug zum interessierenden Objekt durch welches das Ergebnis dieser unmittelbaren Anfrage abgeleitet wird.

Für eine detaillierte Diskussion von Methoden zur Analyse des zu bestimmenden, interessierenden Objektes und den unterschiedlichen Datengruppen sei der Leser auf die „Guideline for Sizing Business Application Software using COSMIC“ verwiesen.

Interessierenden Objekte und Datengruppen im Echtzeit-Software-Bereich

Beispiel 3: Datenbewegungen, die Eingaben (Entries) physikalischer Geräte darstellen, enthalten typischerweise Daten über den Zustand eines interessierenden Objektes, ob etwa ein Ventil offen oder geschlossen ist, welche zeitliche Gültigkeit eine Datenform besitzt, wie die Gültigkeit temporärer Speicherungen charakterisiert ist oder enthalten Daten, die ein aufgetretenes, kritisches Ereignis und die Ursache der Unterbrechung anzeigen. In derselben Weise führt eine Ausgabe (Exit) zu einem Umschaltkommando einer Warnlampe als ein oder aus bezüglich der Daten eines einfachen, interessierenden Objektes.

Beispiel 4: Ein Nachrichtenverteiler kann eine Nachrichtendatengruppe als eine Eingabe ohne Änderung als eine Ausgabe weiterreichen, im Sinne einer FUR eines speziellen Software-Anteils, Die Attribute der Nachrichtendatengruppe können beispielsweise ein „Sender, der Nachrichtenroute und –inhalt empfängt“ und das interessierende Objekt die „Nachricht“ darstellen.

Beispiel 5: Eine gemeinsame Datenstruktur, die das interessierende Objekt darstellt und die durch die funktionalen Nutzeranforderungen definiert ist, kann durch einen funktionalen Prozess gewartet werden, und kann auch zu den meisten funktionalen Prozessen der zu messenden Software zugreifen bzw. sich beziehen.

Beispiel 6: Eine Referenzdatenstruktur, die beispielsweise Graphen oder Tabellen von Werten aus den funktionalen Nutzeranforderungen darstellt, die permanent gespeichert sind (z. B. ROM-Speicher) und sich auf die meisten funktionalen Prozesse der zu messenden Software beziehen.

Beispiel 7: Dateien, die zum Beispiel gemeinsam als „flache“ Strukturen entworfen wurden, können das interessierende Objekt gemäß den funktionalen Nutzeranforderungen darstellen, die sich auf einem persistenten Speichergerät befinden.

3.3.4 Daten oder Datengruppierungen, die keine Kandidaten für die Datenbewegungen darstellen

Irgendwelche Daten als Ein- oder Ausgabebildschirminhalte bzw. Berichte, die keinerlei Beziehung zum interessierenden Benutzerobjekt haben, sollten auf keinen Fall als Datenbewegung identifiziert und auch nicht gemessen werden.

Beispiel 1: Allgemeine Anwendungsdaten, wie Kopf- oder Fußdaten (als Firmenname, Systemname usw.), die auf allen Bildschirmseiten auftreten, sollten nicht berücksichtigt werden.

Beispiel 2: Steuerkommandos (lediglich als Konzept eines Business-Anwendungsbereichs), die den Benutzer die Anwendung erleichtern, wie beispielsweise Navigationskommandos oder „o.k“-Klicks einer speziellen (Fehler-) Nachricht sollten ebenfalls nicht berücksichtigt werden (siehe auch 4.1.10).

Das Generische COSMIC-Software-Modell geht davon aus, dass sämtliche Datenmanipulationen in den funktionalen Prozessen mit einer der vier Datenbewegungstypen verbunden sind (siehe Abschnitt 4.1.6). Folglich können keine anderen Datenbewegungen oder –manipulationen innerhalb funktionaler Prozesse als Kandidaten für zu messende Datenbewegungen identifiziert werden, die eine Addition zu den Entries, Exits, Reads und Writes bedeuten würde. (Zum Beispiel können Datenbewegungen bzw. –manipulationen falsch verstanden oder interpretiert werden; siehe hierzu die Abschnitte 4.1.4 und 4.1.5 zu den Prinzipien für das Read und das Write.)

3.3.5 Die Anwendungsfunktionalität als interessierendes Objekt

In vielen einfachen Echtzeit-Software-Systemen, wie sie beispielsweise im Beispiel 3 des Abschnitts 3.3.3 beschrieben wurden, ist ein physikalisches Gerät als Benutzer nicht vom interessierenden Objekt der Datenbewegung, die es erzeugt oder empfängt, unterscheidbar. In diesem Fall erweitert es das interessierende Objekt nur gering als wenn man es separat zum Benutzer betrachten würde. Wichtig hierbei ist nur, dass es hilfreich sein kann, stets separate Datengruppen und ebenso separate Datenbewegungen klar zu unterscheiden.

Beispiel: Wir nehmen einen Temperatursensor „A“ an, der innerhalb eines funktionalen Prozesses die aktuelle Temperatur an diesen sendet. Der Benutzer ist der Sensor A, die Eingabenachrichtenbezeichnung könnte „aktuelle Temperatur von A“ sein und als interessierendes Objekt dieser Nachricht könnte der „Sensor A“ betrachtet werden. Genau genommen ist „Sensor A“ nicht das interessierende Objekt, sondern „das die Temperatur messende Material bzw. Gerät des Sensors A“. In der Praxis ist dies für eine Unterscheidung nicht so wesentlich bzw. lohnt nicht bei der speziellen Identifizierung des interessierenden Objektes.

3.4 Die Identifizierung der Datenattribute (optional)

Dieser Abschnitt diskutiert die Identifikation der Datenattribute, die durch einen zu messenden Software-Anteil referenziert werden. Bei der vorliegenden Version der Messmethode ist es nicht unbedingt notwendig, die Datenattribute zu bestimmen. Jedoch kann es hilfreich sein, die Datenattribute im Prozess der Datengruppen- und interessierenden Objektbestimmung zu analysieren. Datenattribute können auch als Untereinheit einer geforderten Umfangsmessung angesehen werden wie sie im Abschnitt 4.5 „Erweiterungen der COSMIC-Messmethode“ beschrieben werden.

3.4.1 Definitionen

Definition des Datenattributes
Ein Datenattribut ist der kleinste Teil einer Information innerhalb einer identifizierten Datengruppe, deren Bedeutung sich aus Sicht der funktionalen Nutzeranforderungen ergibt.

Beispiel 1: Datenattribute im Kontext des Business-Applikationsbereichs sind beispielsweise in einem Repository eingetragene Datenelemente oder Datenattribute, die in einem konzeptuellen bzw. logischen Datenmodell auftreten.

Beispiel 2: Datenattribute im Kontext von Echtzeit-Software-Applikationen sind beispielsweise Datenattribute eines empfangenen Signals von einem Sensor oder die Datenattribute von Übertragungsnachrichten.

3.4.2 Über die Assoziation von Datenattributen und Datengruppen

In der Theorie genügt es bereits, wenn eine Datengruppe nur ein Datenattribut aus Sicht der funktionalen Nutzeranforderungen besitzt, um ein interessierendes Objekt zu beschreiben. In der Praxis treten solche Fälle aber nur bei Echtzeit-Applikationen auf (beispielsweise die Eingabe eines Zeitsignals von einer Echtzeituhr), in Business-Applikationen jedoch eher seltener.

4. Die Messphase

Dieses Kapitel beschreibt die Regeln und die Methodik der Messphase aus dem COSMIC-Messprozess.

Die allgemeine Vorgehensweise für die Messung eines Software-Teiles auf der Grundlage bereits formulierte funktionaler Nutzeranforderungen nach dem Generischen COSMIC-Software-Modell ist in der folgenden Abbildung 4.1 zusammengefasst.

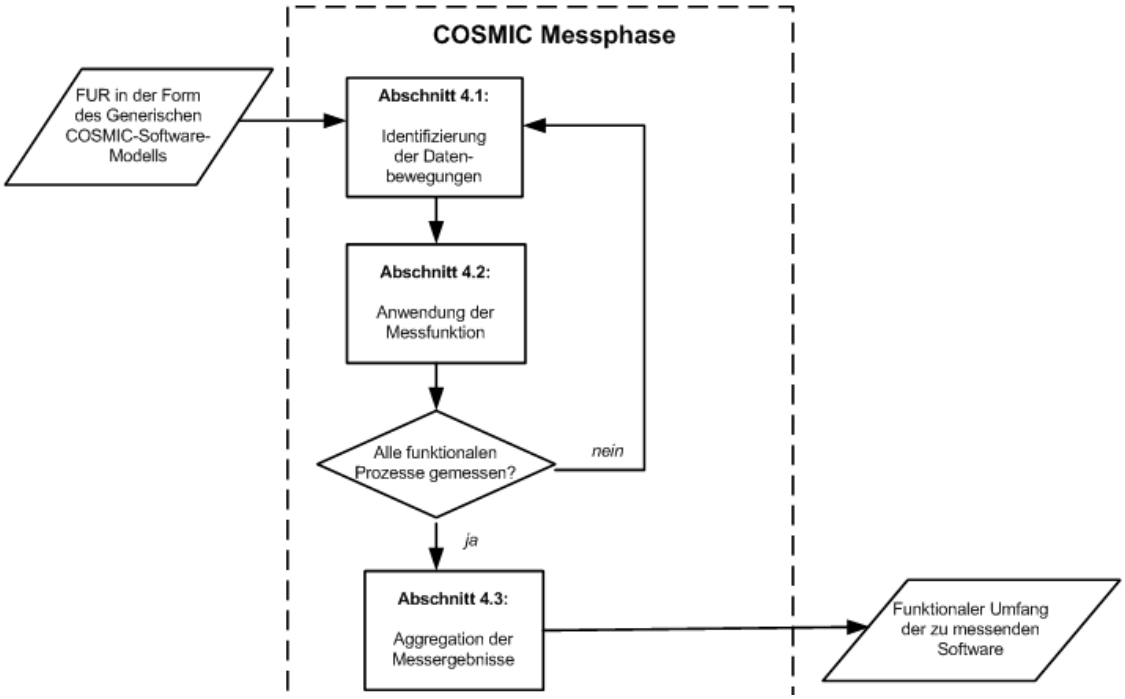

Abb. 4.1: Allgemeiner Prozess zur COSMIC-Messphase

Jeder Schritt in dieser Methode ist gleichzeitig auch die Bezeichnung eines speziellen Abschnittes in diesem Kapitel, in dem die Definitionen und anzuwendenden Prinzipien angegeben sind sowie einige Regeln und schließlich auch wieder Beispiele.

4.1 Die Identifizierung der Datenbewegungen

Dieser Schritt besteht aus der Identifikation der Datenbewegungen (Entry, Exit, Read und Write) für jeden funktionalen Prozess.

4.1.1 Definitionen der Datenbewegungstypen

Definition der Datenbewegung
Der Datenbewegungstyp stellt eine Grundfunktionskomponente dar, die einen einfachen Datengruppentyp bewegt. Hinweis 1: Es gibt genau vier Untertypen eines Datenbewegungstyps: Entry, Exit, Read und Write (-typen). Hinweis 2: Für den Messzweck ist es sinnvoll, dass jeder Datenbewegungstyp auch gewisse, eingeschlossene Datenmanipulationen enthält (siehe auch 4.1.6 für mehr Details). Hinweis 3: Genauer gesagt ist es das Auftreten einer Datenbewegung und nicht ein Datenbewegungstyp das jeweils die auftretenden Datengruppen bewegt (und nicht die Typen). Das gilt auch für die Definitionen von Entry, Exit, Read und Write.

Definition des Einganges (Entry E)
Ein Eingang (Entry bzw. kurz „E“) ist eine Datenbewegung bei der eine Datengruppe eines funktionalen Prozesses außerhalb der Abgrenzung in den Prozess, der diese Funktionalität anfordert, übertragen wird. Hinweis: Ein Eingang hat stets einen Bezug zu den betrachteten Datenmanipulationen (siehe auch Abschnitt 4.1.6).

Definition des Ausganges (Exit X)
Ein Ausgang (Exit bzw. kurz „X“) ist eine Datenbewegung bei der eine Datengruppe eines funktionalen Prozesses, die entlang einer Abgrenzung zum Benutzer, der diese Funktionalität anfordert, übertragen wird. Hinweis: Ein Ausgang hat stets einen Bezug zu den betrachteten Datenmanipulationen (siehe auch Abschnitt 4.1.6).

Definition der Eingabe (Read R)
Die Eingabe stellt eine Datenbewegung dar, die eine erforderliche Datengruppe aus einer persistenten Speicherung für den funktionalen Prozess bereitstellt. Hinweis: Ein Read-Typ hat stets einen Bezug zu den betrachteten Datenmanipulationen (siehe auch Abschnitt 4.1.6).

Definition der Ausgabe (Write W)
Die Ausgabe stellt eine Datenbewegung dar, die eine erforderliche Datengruppe aus einem funktionalen Prozess in die persistente Speicherung schreibt. Hinweis: Ein Write-Typ hat stets einen Bezug zu den betrachteten Datenmanipulationen (siehe auch Abschnitt 4.1.6).

Abbildung 4.2 illustriert die allgemeinen Beziehungen zwischen den vier Typen einer Datenbewegung, den funktionalen Prozess und die Abgrenzung der zu messenden Software.

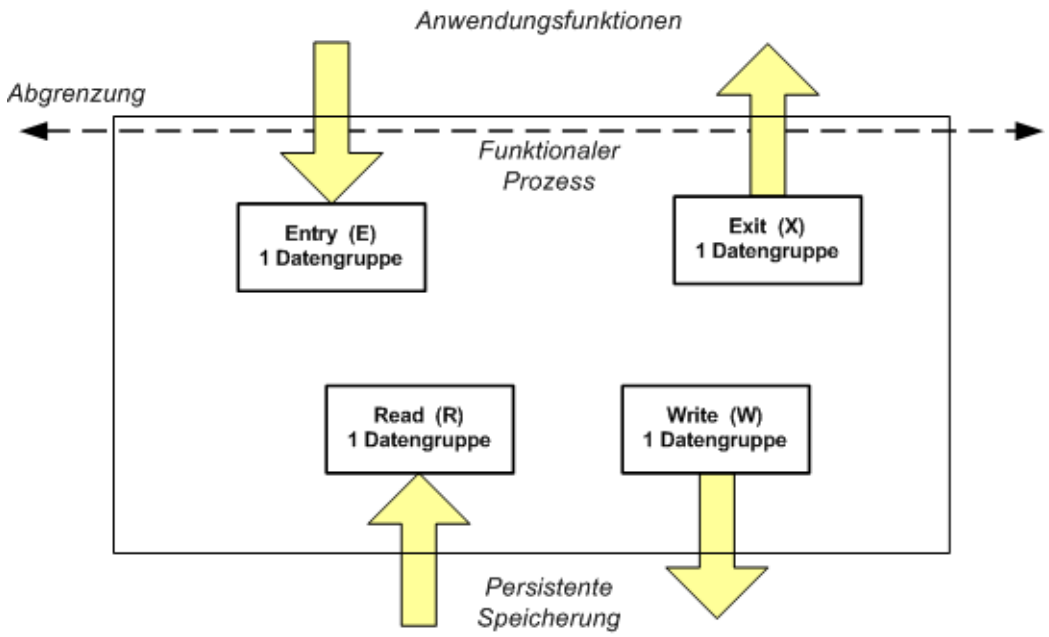

Abb. 4.2: Die vier Typen einer Datenbewegung und ihr Zusammenhang mit dem funktionalen Prozess und den Datengruppen

4.1.2 Die Identifikation der Eingänge (Entries E)

Einmal identifiziert, muss eine Entry-Datenbewegung die folgenden Prinzipien erfüllen:

Prinzipien des Einganges (Entry E)
a) Eine Eingabe soll stets eine einfache Datengruppe, die ein einfaches interessierendes Objekt beschreibt. Die Übertragung erfolgt dabei von einem funktionalen Prozess über eine Abgrenzung in den Prozess, der den Entry für die Funktionalität benötigt. Erfordert der Input eines funktionalen Prozesses mehrere Datengruppen, soll ein Entry eindeutig für jede Datengruppe des Input bestimmt werden.

b) Eine Eingabe soll keine Daten über die Abgrenzung hinweg ausgeben oder auch nicht lesen oder schreiben.

c) Wenn ein funktionaler Prozess Daten vom Benutzer benötigt, dieser aber die Daten nicht senden kann, oder der Benutzer kann auf eine ankommende Nachricht nicht reagieren, so sollte dennoch ein Entry zum funktionalen Prozess für die zu erhaltenen Daten identifiziert werden. Allerdings soll in diesem Fall die Anfragenachricht des funktionalen Prozesses zum Benutzer nicht als Exit gezählt werden.

d) Wenn jedoch ein funktionaler Prozess Benutzerdaten erhalten muss und der Benutzer gibt dann Daten zu dieser Anforderung zurück, wird ein Exit für die Anforderung und ein Entry für die Rückgabe der geforderten Daten gezählt (sieche auch Abschnitt 4.1.9).

Die folgenden Regeln helfen bei der Festlegung einer Entry-Datenbewegung:

Regeln des Einganges (Entry E)

a) Die Datengruppe einer auslösenden Eingabe kann aus nur einem Datenattribut bestehen, welches einfach die Software „informiert“, dass „ein Ereignis Y aufgetreten ist“. Oftmals, insbesondere bei Business-Applikationen, besteht die Datengruppe des auslösenden Ereignisse aus vielen Datenattributen, so dass die Software zum Beispiel darüber „informiert“ wird, dass „ein Ereignis Y aufgetreten ist, das die folgenden Daten zu diesem Ereignis besitzt“.

b) Zeitsignale auslösender Ereignisse sollen stets extern zur gemessenen Software betrachtet werden. Daher wird beispielsweise ein Zeitsignal, das alle 3 Sekunden auftritt, mit einem Entry als Datenbewegung einer Datengruppe eines Datenattributes verbunden. Zu beachten ist, dass es keinen Unterschied macht, ob ein auslösendes Ereignis periodisch durch Hardware oder durch einen Softwareteil außerhalb der Abgrenzung der gemessenen Software erzeugt wird.

c) Sofern ein spezifischer funktionaler Prozess eine Zeit von einer Systemuhr erhält, wird das nicht als ein Entry betrachtet.

d) Wenn ein spezifisches Ereignis auftritt, das ein Entry einer Datengruppe mit bis zu *n* Datenattributen eines speziellen interessierenden Objektes erzeugt und die FUR auch ein weiteres Auftreten desselben Ereignisses erlaubt bzw. festlegt, dass ein Entry zu einer Datengruppe mit Attributwerten eine Teilmenge der *n* Attribute des betrachteten Objektes betrifft, dann wird das Entry so identifiziert, dass es die *n* Datenattribute beinhaltet.

Beispiel: Der Regel (c) führt beispielsweise ein funktionaler Prozess eines Zeitsignals nicht zu einem Entry, der ein erhaltenen Systemzeitwert identifizieren würde.

Einmal identifiziert kann jede Entry-Datenbewegung durch Eintrag in die zugehörige Position der Generischen Software-Modell-Matrix (Anhang A) mit einem „E“ gezählt werden.

4.1.3 Die Identifikation der Ausgänge (Exits X)

Bezüglich der Festlegung möglicher Exit-Datenbewegungen müssen folgende Prinzipien berücksichtigt werden:

Prinzipien des Ausganges (Exit X)
a) Ein Ausgang bzw. Exit bewegt eine einfache Datengruppe, die ein einfaches interessierendes Objekt eines funktionalen Prozesses beschreibt, als Ausgang entlang der Abgrenzung zum Benutzer. Gibt der funktionale Prozess mehr als eine Datengruppe aus, so wird ein Exit für jede eindeutige, ausgegebene Datengruppe gezählt (Siehe auch Abschnitt 4.1.7 zur Eindeutigkeit der Datenbewegungen). b) Ein Exit soll keine Daten über die Abgrenzung hinaus übertragen, oder Daten lesen oder schreiben.

Die folgenden Regeln sollten die Erkennung möglicher Exit-Datenbewegungen unterstützen.

Regeln des Ausganges (Exit X)
a) Alle erzeugten Nachrichten und Software-Ausgaben (außer Nutzerdaten), wie beispielsweise Fehlernachrichten, werden als Werte eines Attributes eines betrachteten Objektes ansehen (bezeichnet als „Fehleranzeige“). Dafür wird ein einfaches Exit, das all diese Nachrichten repräsentiert, die in einem funktionalen Prozess durch die FUR notwendig sind. b) Wenn ein Exit eines funktionalen Prozesses eine Datengruppe bis zu *n* Datenattributen eines speziellen interessierenden Objektes bewegt und die FUR funktionale Prozesse, die bei einem Exit Datengruppen mit Teilmengen aus Objektattributen erlaubt, dann wird genau ein Exit festgelegt, das alle *n* Datenattribute berücksichtigt.

Beispiele zur Regel (a) sind folgende:

Beispiel 1: Bei einem Mensch-Maschine-Dialog sind Beispiele für derartige Nachrichten, die bei einer Datenprüfung auftreten, „Formatfehler“, „Kunde nicht gefunden“, „Fehler: Ausdrücke und Bedingungen getestet“, „Kreditgrenze überzogen“ usw. All diese Fehlernachrichten werden als Auftreten eines Exit zu jedem funktionalen Prozess, wo derartige Nachrichten auftreten, die als „Fehlernachricht“ bezeichnet werden können.

Beispiel 2: Fehlernachrichten zum Benutzer, die nicht durch Software-Applikationen erzeugt werden, werden bei der Messung der Applikation ignoriert. Ein Beispiel einer derartigen Nachricht kann die Betriebssystemnachricht „Drucker X ist nicht verfügbar“ sein.

Beispiel 3: Bei einem Mensch-Maschine-Dialog ist im Falle einer Nachricht im Fehlerfall, die Nutzerdaten enthält, ein Exit im auftretenden funktionalen Prozess zu zählen. Ein konkretes Beispiel einer derartigen Nachricht kann sein „Warnung: der gewünschte Betrag überschreitet die Grenze von 100 EUR“, wobei die 100 EUR eine berechnete Variable darstellt. In diesem Beispiel enthält das Exit eine Datengruppe über ein Kundenbankkonto.

Beispiel 4: Bei einem Echtzeitsystem, bei dem ein funktionaler Prozess kontinuierlich die korrekte Funktion aller Hardware-Geräte prüft, kann eine Nachricht sein „Sensor X hat Fehlfunktion“, wo „X“ eine Variable darstellt. Diese Nachricht wird als ein Exit eines funktionalen Prozesses bezogen auf den Wert von „X“ identifiziert.

Beispiel 5: Es seien A und B funktionale Prozesse. „A“ kann möglicherweise zwei verschiedene Bestätigungsnachrichten und fünf Fehlernachrichten zum Benutzer und „B“ kann acht Fehlernachrichten zu seinem Benutzer haben. In diesem Beispiel wird ein Exit im funktionalen Prozess von „A“, der 5+2=7 Nachrichten verarbeitet und ein separates Exit innerhalb des funktionalen Prozesses „B“, der 8 Nachrichten verarbeitet, identifiziert.

Einmal identifiziert kann jede Exit-Datenbewegung durch Eintrag in die zugehörige Position der Generischen Software-Modell-Matrix (Anhang A) mit einem „X“ gezählt werden.

4.1.4 Die Identifikation der Eingaben (Reads R)

Für die Festlegung möglicher Read-Datenbewegungen müssen folgende Prinzipien berücksichtigt werden:

Prinzipien der Eingabe (Read R)
a) Ein Einlesen bzw. ein Read ist eine einfache Datengruppe, die ein betrachtetes Objekt vom persistenten Speicher zum funktionalen Prozess, der das Read erfordert, bewegt. Empfängt der funktionale Prozess mehr als eine Datengruppe vom persistenten Speicher, so wird ein Read für jede eindeutige, empfangene Datengruppe gezählt (Siehe auch Abschnitt 4.1.7 zur Eindeutigkeit der Datenbewegungen). b) Ein Read soll keine empfangenen oder ausgegebenen Daten üner die Abgrenzung hinweg zum Benutzer oder geschriebene Daten darstellen. c) Bewegungen oder Manipulationen von Konstanten oder Variablen während eines funktionalen Prozesses, die intern in diesem Prozess sind und die nur durch den Programmierer geändert werden können oder sich als Resultat einer unmittelbaren Berechnung bzw. von gespeicherten Daten durch einen funktionalen Prozess nur aus der Implementation und nicht aus der FUR ergeben, werden nicht als Read-Datenbewegung betrachtet.

d) Eine Read-Datenbewegung enthält stets irgendeine Read-Anforderung bzw. Funktionalität (eine separate Datenbewegung wird nicht als Read-Funktionalität gezählt) (siehe auch Abschnitt 4.1.9).

Einmal identifiziert kann jede Read-Datenbewegung durch Eintrag in die zugehörige Position der Generischen Software-Modell-Matrix (Anhang A) mit einem „R“ gezählt werden.

4.1.5 Die Identifikation der Ausgaben (Writes W)

Für die Festlegung möglicher Write-Datenbewegungen müssen folgende Prinzipien berücksichtigt werden:

Prinzipien der Ausgabe (Write W)
a) Ein Ausschreiben bzw. Write soll eine einfache Datengruppe, die ein einfaches interessierendes Objekt eines funktionalen Prozesses, der ein Write in einen persistenten Speicher einträgt, darstellen. b) Ein Write soll keine Daten über die Abgrenzung hiweg zum bzw. vom Benutzer empfangen oder ausgeben oder auch Daten einlesen. c) Eine Anforderung für das Löschen einer Datengruppe vom persistenten Speicher wird als eine einfache Write-Datenbewegung gezählt. d) Bewegungen oder Manipulationen von Daten während eines funktionalen Prozesses, die nicht persistent nach einem vollständigen, funktionalen Prozess sind oder Variablen aktualisieren, die intern zum funktionalen Prozess sind bzw. unmittelbar in einer Berechnung entstehen, werden nicht als Write-Datenbewegungen betrachtet.

Einmal identifiziert kann jede Write-Datenbewegung durch Eintrag in die zugehörige Position der Generischen Software-Modell-Matrix (Anhang A) mit einem „W“ gezählt werden.

4.1.6 Über die mit den Datenbewegungen verbundenen Manipulationen

Teilprozesse, wie sie im Prinzip (d) des Generischen Software-Modells (siehe Abschnitt 1.4) erklärt sind, sind entweder Datenbewegungen oder Datenmanipulationen. Allerdings wird die eigenständige Existenz von Teilprozessdatenbewegungen in den gegenwärtigen COSMIC-Dokumenten nicht betrachtet.

Definition der Datenmanipulation
Datenmanipulation stellt jedwede Behandlung von Daten dar, die anders als die Bewegung von Daten in oder aus einem funktionalen Prozess oder zwischen funktionalem Prozess und persistenter Speicherung ist

Das folgende Prinzip bestimmt, wie die COSMIC-Methode die Datenmanipulation versteht.

Prinzipien für die mit den Datenbewegungen verbundenen Manipulationen
Alle Datenmanipulationen eines funktionalen Prozesses sollten mit den vier Typen der Datenbewegung (E, X, R und W) verbunden sein. Einvernehmlich wird eine Datenmanipulation eines funktionalen Prozesses stets durch eine Datenbewegung dieses Prozesses repräsentiert.

Die Notwendigkeit, die Art der Datenmanipulation und die involvierten Datenbewegungstypen zu bestimmen, besteht nur bei der Messung von *Änderungen* der Software (siehe Abschnitt 4.4). Als typische Änderungsanforderung können Attribute bewegt oder zu einer Manipulation gehörend eine spezielle Datenbewegung betreffen, die nur die Manipulation nicht aber die Datenbewegung selbst beeinflussen. Eine derartige Änderung muss identifiziert und gemessen werden. Wenn es also eine Änderungsanforderung zu einer Datenmanipulation im funktionalen Prozess gibt, muss die Messperson identifizieren, welche Datenbewegung mit dieser Datenmanipulation zusammenhängt.

Es folgen einige allgemeine Regeln zur Identifikation einer Datenmanipulation für die jeweilige Datenbewegung.

Eingangsdatenbewegungen

Ein Eingang (Entry) enthält alle Datenmanipulationen

- um zu ermöglichen, dass eine Datengruppe durch einen Benutzer Eingang findet (d. h. Manipulationen formatiert und darstellt) und bestätigt wird,

- aber nicht Manipulationen, die andere Datenbewegungen enthalten oder die nach einer bereits eingegebenen und bestätigten Datengruppe stattfinden.

Beispiel: Ein Entry enthält alle Manipulationen, die zu eingegebenen Daten auf einem Bildschirm formatiert und dargestellt sind. Ausgenommen(!) irgendwelche Eingabedaten (als Reads) die zur Prüfung der Kodierung oder zugeordneter Darstellungsformen benötigt werden.

Eine Entry-Datenbewegung kann eine Enter-Anforderung enthalten, ausgenommen, wenn der funktionale Prozess den Benutzer darüber informieren muss, welche Daten zu senden sind (siehe Abschnitt 4.1.9 zu den zu sendenden Daten und 4.1.10 zur Behandlung eines leeren Eingabebildschirms).

Ausgangsdatenbewegungen

Ein Exit enthält alle Datenmanipulationen

- um Datenattribute einer Datengruppe, die ausgegeben werden sollen, zu definieren und/oder

- die Ausgabe einer Datengruppe und die Zuordnung zum Benutzer zu ermöglichen (d. h. Manipulationen formatiert und darstellt),

- aber nicht solche Mainpulationen, die andere Datenbewegungen enthalten.

Beispiel: Ein Exit enthält alle Prozesse der Formatierung und Aufbereitung zu druckender Datenattribute einschließlich der Kennzeichnungen für die Lesbarkeit der Daten, ausgenommen(!) irgendwelcher Reads oder Entries die für die Unterstützung der Druckwerte bzw. -beschreibungen der zu druckenden Datenattribute erforderlich sein könnten.

Eingabedatenbewegungen

Eine Eingabe (Read) enthält alle Prozesse und/oder Berechnungen, die für den Empfang einer Datengruppe von einem persistenten Speicher erforderlich sind, allerdings nicht irgendwelche Manipulationen, die andere Datenbewegungstypen oder irgendwelche Manipulationen nach erfolgreichem Read-Abschluss beinhalten.

Beispiel: Ein Read enthält alle mathematischen Berechnungen und logischen Prozesse, die für den Empfang einer Datengruppe aus einem persistenten Speicher erforderlich sind, aber nicht die Manipulationen derartiger Attribute, die nach dem Erhalt der Datengruppe erfolgen.

Ein Read enthält auch stets eine Read-Anforderungsfunktionalität (siehe Abschnitt 4.1.9).

Ausgabedatenbewegungen

Eine Ausgabe (Write) enthält alle Prozesse und/oder Berechnungen zur Erzeugung einer auszugebenden Datengruppe, ausgenommen irgendwelcher Manipulationen, die andere Datenbewegungstypen oder Manipulationen nach erfolgreichem Write-Abschluss enthalten.

Beispiel: Ein Write enthält alle mathematischen Berechnungen und logischen Prozesse zur Erzeugung oder Aktualisierung einer ausgegebenen Datengruppe oder deren Löschen, ausgenommen(!) irgendwelcher Reads der Entries, die erforderliche Werte von Datenattributen betreffen, die in der zu schreibenden oder löschenden Datengruppe enthalten sind.

4.1.7 Eindeutigkeit der Datenbewegungen und mögliche Ausnahmen

Das Generische Software-Modell setzt voraus, dass normalerweise in jedwedem funktionalen Prozess alle Daten für das interessierende Objekt, die der funktionale Prozess erfordert, bezüglich der Eingabe im Entry-Datenbewegungenstyp und/oder des Einlesens im Read-Datenbewegungstyp und/oder des Schreibens im Write-Datenbewegungstyp und/oder der Ausgabe im Exit-Datenbewegungstyp angegeben sind. Weiterhin setzt das Modell voraus,

dass alle Datenmanipulationen, die sich insgesamt aus den Datenattributen der bewegten Datengruppen ergeben, eindeutig einer Datenbewegung zugeordnet sind.

Beispiel: Zur Illustrierung des Letztgenannten wird das zweifache Auftreten eines gegebenen funktionalen Prozesses betrachtet. Angenommen, dass das erste Auftreten die Werte zu bewegender Attribute zu einem Datenmanipulations-teilprozess 'A' führt und dass das andere Auftreten desselben funktionalen Prozesses der Attributwerte zu einem anderen Datenmanipulationsteilprozess 'B' führt. In diesem Fall sind beide Datenmanipulationsteilprozesse 'A' und 'B' mit derselben Datenbewegung verbunden und werden folglich auch nur als eine Datenbewegung identifiziert und für diesen funktionalen Prozess gezählt.

Dabei kann sich jedoch eine Ausnahmesituation in der Form ergeben, dass bei unterschiedlichen Datengruppentypbeschreibungen zu einem interessierenden Objekt gemäß der FUR eine Datenbewegung desselben Typs (E, R, W oder X) im selben funktionalen Prozess erfolgt. Alternativ dazu und untypisch kann auch dieselbe Datengruppe als denselben Datengruppentyp (E, R, W oder X) im selben funktionalen Prozess bewegt werden, allerdings mit unterschiedlichen Datenmanipulationen.

Die folgenden Regeln und Beispiele beziehen sich auf die üblichen Gegebenheiten, einschließlich der Ausnahmen und möglichen Beispiele eines korrekt erscheinenden aber doch fehlerhaften Vorgehens.

Regeln für die Eindeutigkeit der Datenbewegungen und mögliche Ausnahmen
a) Sofern die Funktionalen Nutzeranforderungen es nicht anders spezifizieren, werden alle beschriebenen Datenattribute eines interessierenden Objektes und alle zugehörigen Datenmanipulationen, die in einen funktionalen Prozess Eingang finden müssen, als ein Entry identifiziert und gezählt. (Hinweis: Ein funktionaler Prozess kann natürlich mehrere Entry-Typen behandeln, wobei jedes Entry eine Datengruppe eines unterschiedlichen interessierenden Objektes beschreibt.) Dieselbe äquivalente Regel gilt auch für die Read-, Write- oder Exit-Datenbewegung eines gegebenen funktionalen Prozesses. b) Mehr als eine Entry-Datenbewegung liegen vor, wenn jede bewegte Datengruppe eines interessierenden Objektes in einem gegebenen funktionalen Prozess identifiziert und gezählt wird, weil es auch diese multiplen Entries in den Funktionalen Nutzeranforderungen gibt. In gleicher Weise haben wir mehr als ein Entry für eine Bewegung derselben Datengruppe im selben funktionalen Prozess, allerdings mit unterschiedlichen, zugeordneten Datenmanipulationen, weil es wiederum auch diese multiplen Entries in den Funktionalen Nutzeranforderungen gibt. Derartige FUR können in einem funktionalen Prozess auftreten, während die multiplen Entries von unterschiedlichen Benutzern, die verschiedene Datengruppen zu selben interessierenden Objekt beschreiben, herrühren.

Dieselbe äquivalente Regel kann auch für die Read-, Write- oder Exit-Datenbewegung eines gegebenen funktionalen Prozesses angewandt werden.

c) Wiederholtes Auftreten von Datenbewegungstypen (d. h. jedes Auftreten bewegt dieselbe Datengruppe mit derselben Datenmanipulation) wird in einem funktionalen Prozess nicht mehrfach identifiziert und gezählt.

d) Wenn beim mehrfachen Auftreten von Datenbewegungstypen in einem gegebenen funktionalen Prozess sich diese hinsichtlich der zugeordneten Manipulationen aufgrund unterschiedlicher Werte der Datenattribute einer Datengruppe, die sich durch verschiedene Prozessverläufe ergibt, unterscheiden, wird die Datenbewegung in diesem Prozess nicht mehr als einmal identifiziert und gezählt.

Die folgenden Beispiele illustrieren die oben genannten Regeln.

Beispiel 1: (Bezüglich Regel (a)) In einem funktionalen Prozess können alle Reads für die Daten, die ein spezielles interessierendes Objekt beschreiben, logisch als Rückgabe der erforderlichen Attribute dieses Objektes betrachtet werden (d. h. als insgesamt erforderlicher „Zustandsvektor“ dieses interessierenden Objektes). Somit wird nur ein Read zum interessierenden Objekt des funktionalen Prozesses benötigt und wird auch so identifiziert.

Beispiel 2: (Bezüglich der Regeln (a) und (b)) Analog dem Beispiel 1, wenn gelesene Daten durch eine Write-Anweisung persistent gespeichert werden können, so ist der Normalfall (nach Regel (a)), dass auch genau ein Write identifiziert wird, wenn eine bewegte Datengruppe, die alle Datenattribute eines interessierenden Objektes enthält, in einem gegebenen funktionalen Prozess persistent gespeichert wird. Ausgenommen jedoch, wenn eine FUR existiert, die für einen einfachen funktionalen Prozess zwei unterschiedliche Datengruppen, die dasselbe interessierende Objekt beschreiben, (aus-) schreibt, d. h. für die spätere Verwendung zweier unterschiedlicher Benutzer in anderen funktionalen Prozessen. Ein Beispiel der Anwendung der Regel (b) ist, wenn ein einfacher funktionaler Prozess A zwei Teilmengen von Daten eines Bankkontofiles für die spätere Verwendung durch separate Programme extrahieren muss. Die erste Teilmenge seien die „Kontoüberziehungsdetails“ (welche das negative Kontostandsattribut enthalten). Die zweite Teilmenge seien die „Kontostandshöchstwertdetails“ (welche nur den Kontobesitzernamen und die Adresse für mögliche Marketing-Benachrichtungen beinhaltet). Der funktionale Prozess besitzt dann zwei Writes, jeder einzelne für die jeweilige Teilmenge.

Beispiel 3: (Bezüglich der Regel b)) Es ist durchaus möglich, dass es einen FUR für einen funktionalen Prozess gibt, der zwei oder mehr Exits für die Bewegung unterschiedlicher Datengruppen zum selben interessierenden Objekt und zwar für unterschiedliche Benutzer, besitzt. Zum Beispiel, wenn ein neuer Angestellter in der Firma eingestellt wird, so ist ein Bericht mit seinen von ihm bestätigten Personendaten zu erstellen und die Fimen-Security über seine Berechtigung des Betretens des Firmengebäudes zu benachrichtigen.

Beispiel 4: (Bezüglich der Regel (c)) Wenn ein Read in einer FUR viele Rückhol- bzw. Wiederholereignisse erfordert, im Gegensatz zu einer einfachen Suche in einer Datei, so wird für die Umfangsbestimmung nur ein Read gezählt.

Beispiel 5: (Bezüglich der Regel (c)) Angenommen in einem funktionalen Echtzeitprozess erfordert die FUR die Eingabe von Daten, zum Beispiel eines (Hardware-) Gerätes, und zwar zweimal zu einem festen Zeitpunkt, um den Änderungsanteil oder eine Wertänderung im Verlauf dieses Prozesses überhaupt zu testen. Wenn dabei zwei Entries bezüglich der bewegten Datengruppe und der zugehörigen Datenmanipulation identisch sind, wird nur ein Entry identifiziert (siehe Abschnitt 4.1.6 bezüglich der Datenmanipulationstypen, die sich auf das Entry beziehen).

Beispiel 6: (Bezüglich der Regel (c)) Siehe Abschnitt 4.1.2 Regel (d) zu Entries und Abschnitt 4.1.3 Regel (b) zu Exits.

Beispiel 7: (Bezüglich der Regel (d)) Im Falle eines funktionalen Prozesses, bei dem unterschiedliche Datenmanipulationen erforderlich sind, die von den jeweiligen Werten der Datenattribute eines Entry abhängen, wird für die Umfangsbestimmung nur ein Entry gezählt.

Beispiel 8: Angenommen, dass eine FUR ein Read erfordert, der Entwickler aber entscheidet, dies durch zwei Befehle zu implementieren, die unterschiedliche Teilmengen von Datenattributen desselben betreffenden Objektes vom persistenten Speicher zu unterschiedlichen Stellen des funktionalen Prozesses herauslesen, so wird nur ein Read gezählt.

4.1.8 Funktionale Datenbewegungen von und zur persistenten Speicherung

Dieser Abschnitt widmet sich den betreffenden Datenbewegungen, dass der funktionale Prozess eines Software-Teils Daten vom oder zum persistenten Speicher bewegt und der Speicher entweder lokal oder entfernt ist. Die Beispiele zeigen auch, wie Applikationen, die Speicher benötigen, durch andere Software verarbeitet werden, die Applikationen anderer Schichten unterstützen, wie beispielsweise die persistente Speicherung von Gerätetreiber-Software.

Die Beispiele illustrieren die Anwendung des Prinzips (g) des Software-Kontextmodells and der Prinzipien des Generischen Software-Modells. Der Schlüssel zum Verständnis der Beispiele ist der, dass diese Prinzipien separat zu jedem zu messenden Software-Teil angewandt werden müssen.

Praktisch bedeutet dies, dass die Beispiele auch für Messaufgaben für zwei Software-Teile, die entweder hierarchisch (d. h. wenn die beiden Teile sich in unterschiedlichen Schichten befinden) oder als Client/Server-Beziehung (d. h. wenn die beiden Teile einander zugeordnet sind) verbunden sind. Die Beispiele zeigen auch, wie die Datenbewegungen, die jeweils zwischen zwei zu messenden Software-Teilen ausgetauscht werden, zu modellieren sind.
Die Beispiele sind jeweils nach den Konventionen für Sequenzdiagramme dargestellt. Die Notation dieser Diagramm ist folgende:

- Eine dicke, senkrechte Linie mit einem Pfeil symbolisiert einen funktionalen Prozess.

- Horizontale, mit einem Pfeil versehene Linien stellen die Datenbewegungen dar, gekennzeichnet jeweils mit E, X, R oder W für den Entry, den Exit, das Read oder das Write.
- Eine gestrichelte senkrechte Linie symbolisiert jeweils eine Abgrenzung.

Beispiel 1: Ein funktionaler Prozess bewegt Daten von oder zu einem persistenten Speicher

Dieses Beispiel schließt zwei Software-Teile ein, einen Teil einer Software-Applikation bezeichnet mit 'A' und einen separaten, weiteren Software-Teil 'B', der einen Gerätetreiber für ein persistentes Speichergerät darstellt, den die Applikation benötigt (Wir ignorieren dabei der Einfachheit halber das wahrscheinlich vorhandene Betriebssystem, das üblicherweise die hier betrachtete Datenübertragung effizient realisiert).

Unser Konzept der Schichten verweist darauf, dass die beiden Software-Anteile sich in unterschiedlichen Schichten befinden: die *Applikationsschicht* und die *Gerätetreiberschicht*. Physisch liegt hier eine hierarchische Beziehung zwischen den Software-Komponenten vor (das Betriebssystem ignorierend) und eine Schnittstelle zwischen der Software der beiden Schichten (bereits im Beispiel in Abbildung 2.2 dargestellt).

Normalerweise spezifiziert die FUR der Applikation A die funktionalen Prozesse, die die Reads und Writes zum persistenten Speicher einschließen, betrachtet aber nicht, wie diese Reads und Writes durch andere Infrastruktur-Software behandelt werden.

Unter Anwendung des COSMIC-Modells für die zwei Software-Teile könnten die Benutzer der Software A in der Anwendungsschicht beispielsweise Personen sein, während der Benutzer der Software B in der Geräteschicht den Applikationsteil der Software A (das Betriebssystem ignorierend) darstellt.

Unter der Annahme, dass ein funktionaler Anfrageprozess 'FP A' der Software A in der Applikationsschicht eine Read-Datenbewegung erfordert, zeigt Abbildung 4.3 (a) das COSMIC–Modell für die Anfrageapplikation. Die physische Abfrage geforderter Daten vom persistenten Speicher wird durch den funktionalen Prozess 'FP B' der Software B in der Gerätetreiberschicht realisiert. Abbildung 4.3. (b) zeigt das Modell des funktionalen Prozesses des Grätetreibers.

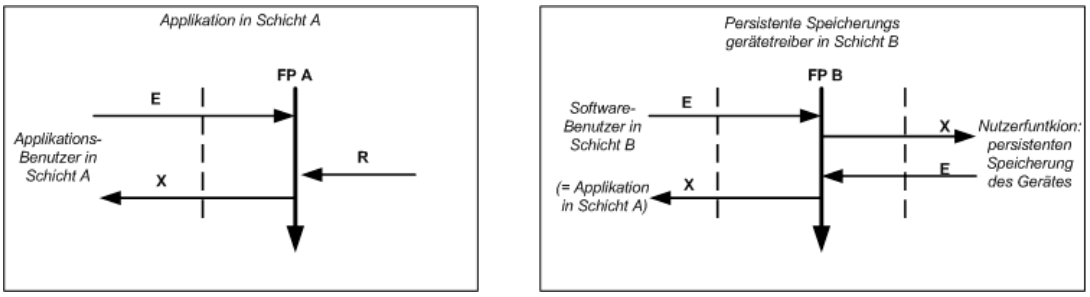

Abb. 4.3: Lösung für ein Read aus der Software A innerhalb der Applikationsschicht und aus B in der Gerätetreiberschicht (links (a) und rechts(b))

Abbildung 4.3 (a) zeigt die durch ein Entry ausgelöste Abfrage, gefolgt durch Read und dann einem Exit mit dem Abfrageergebnis. FP A hat dabei keine Kenntnis, woher die Daten empfangen werden noch darüber, dass praktisch das Read sich auf irgendwelche Gerätetreiber bezieht.

Abbildung 4.3.(b) zeigt wiederum, dass das von der Applikation A geforderte Read als auslösendes Entry des funktionalen Prozesses FP B, das die erforderlichen Daten vom persistenten Speicher über ein Exit/Entry-Paar erhält und die Daten der Applikation als ein Exit zurückgibt, empfängt. Die Applikation A und das persistente Speicherungs-(Hardware)-Gerät sind somit die Benutzer der Gerätetreiber-Software B.

Der scheinbare Widerspruch zwischen der Anzahl der Datenbewegungen des „Read“ der Software der Applikationsschicht und den Entry/Exit-Paaren der Software der Gerätetreiberschicht ist darin begründet, dass per Konvention eine Read-Datenbewegung stets auch eine „Leseanforderungsfunktion“ beinhalten soll.

Ein exaktes analoges Modell ist anzuwenden, wenn der funktionale Prozess FP A einige Daten über eine Write-Datenbewegung persistent machen soll. Bei diesem Beispiel würde das Exit des funktionalen Prozesses FP B der Gerätetreiber-Software der Applikation A ein „Return-Code“ oder eine Fehlernachricht enthalten.

Beispiel 2: Ein funktionaler Prozess erhält Daten von einem zugeordneten Software-Anteil

Bei diesem Beispiel wird angenommen, dass die beiden zu messenden Software-Anteile eine Client/Server-Beziehung haben, d. h. die eine Software, der Client, erhält Dienste und/oder Daten von der anderen Software, dem Server in *derselben* Schicht. Abbildung 4.4 zeigt ein Beispiel einer derartigen Beziehung, wobei die beiden Software-Anteile die Hauptkomponenten derselben Applikation darstellen. Dieselbe Beziehung würde bestehen, wenn dasselbe Diagramm auf zwei Software-Anteile separater Applikationen angewandt wird, wobei die eine Applikation Daten von der anderen erhalten würde.

Physisch gesehen könnten die beiden kommunizierenden Komponenten auf separaten Prozessoren abgearbeitet werden. In diesem Fall würden sie Daten mit dem jeweiligen Betriebssystem austauschen oder mit irgendeiner anderen Zwischenschicht dieser Prozessoren in einer Software-Architektur, wie sie zum Beispiel Abbildung 2.2 zeigt. Allerdings würden die beiden Komponenten nach dem COSMIC-Modell logisch Daten über Entry/Exit-Paare austauschen. Dabei wurde die gesamte dazwischen liegende Soft- und Hardware in diesem Modell ignoriert (wie auch in der rechten Seite der Abbildung 3.2 gezeigt).

Die Abbildung 4.4. zeigt, wie ein funktionaler Prozess FP C1 der Client-Komponente C1 durch ein Entry vom Benutzer ausgelöst wird, das beispielsweise die Parameter einer Anfrage beinhaltet. Die FUR der Komponente C1 ist so zu betrachten, dass diese Komponente die Server-Komponente C2 nach den geforderten Daten fragt bzw. zum Ausdruck bringen muss.

Um die erforderlichen Daten zu erhalten, muss der FP C1 ein Exit zur Komponente C2 ausgeben, das die geforderten Anfrageparameter enthält. Die Komponente C1 ist jetzt der Benutzer der Komponente C2. Das bedeutet, dass eine Abgrenzung zwischen diesen beiden Komponenten existiert. Die Exit-Datenbewegung kreuzt die Abgrenzung zwischen C1 und C2 und erhält das auslösende Entry des funktionalen Prozesses FP C2 der Komponente C2. Der funktionale Prozess FP C2 der Komponente C2 erhält die erforderlichen Daten via einem Read und sendet die Daten zurück zu C1 mit einem Exit. Der funktionale Prozess FP C1 der Komponente C1empfängt die Datenbewegung als ein Entry. FP C1 passiert dann die Daten

als ein Exit zur Erfüllung der Benutzeranfrage. Dieses Beispiel 2 erfordert für die Anfrage 7 Datenbewegungen, um die geforderte Anfrage der Applikationsschicht zu erfüllen. Dies verglichen mit den 3 Datenbewegungen (1 × E, 1 × R und 1 × X) würde in der Applikationsschicht erforderlich sein, wenn die Komponente C1 möglicherweise die Daten vom *lokalen* persistenten Speicher erhielte, wie es in Abbildung 4.3 (a) gezeigt ist.

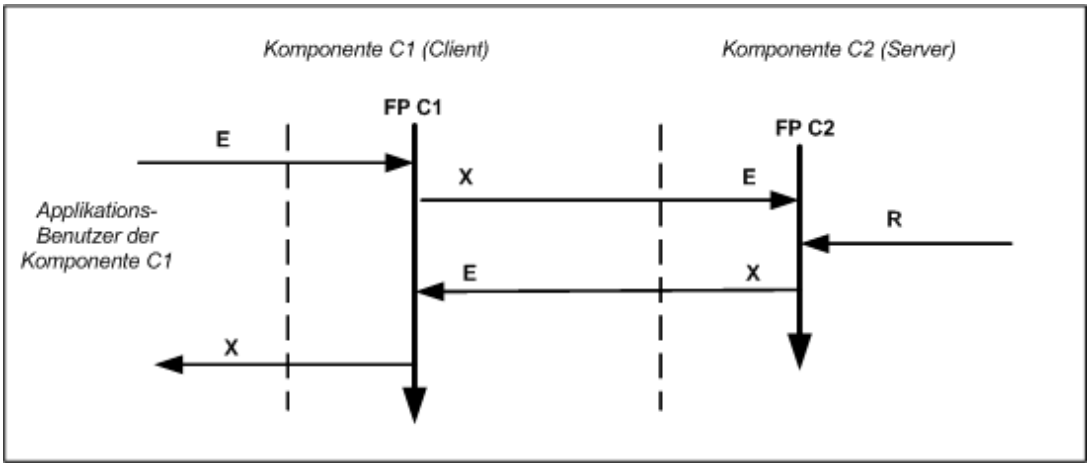

Abb. 4.4: Datenaustausch zwischen Komponentenpaaren

Die Komponente C2 wird sicherlich die Dienste der Gerätetreiber-Software vom persistenten Speicher einer unteren Schicht der Software-Architektur für den Empfang der Daten von der Hardware nutzen, wie es in Beispiel 1 beschrieben wurde.

Beim Vergleich der Beispiele 1 und 2, sieht man im Beispiel 1, dass die Modelle der Applikation A und des Gerätetreibers B nicht wie in Beispiel 2 kombiniert werden können. Das liegt daran, dass ein Read nicht über eine Abgrenzung geht. Abbildung 4.3 (b) zeigt, dass die Applikation A ein Benutzer der Gerätetreiber-Software darstellt. Aber die Umkehrung gilt nicht, was die hierarchische Eigenschaft der Software in unterschiedlichen Schichten verdeutlicht.

Im Gegensatz dazu zeigt Abbildung 4.4 die beiden Komponenten in einem Modell, da ihr Datenaustausch als Komponentenpaar oder gleiche Komponenten in derselben Schicht stattfindet. Die Komponente C1 ist der Benutzer der Komponente C2 und umgekehrt sind sie über eine gemeinsame Abgrenzung aufgeteilt.

Man beachte, dass wir in diesen beiden Beispielen der Einfachheit halber die Erzeugung einer Exit-Fehlernachricht durch die Applikation A bzw. durch die Komponente C1 ignoriert haben (als weiteres Ergebnis der Abfrage für das Exit), das als Ergebnis eines Rückkehr-Codes zu einer Read-Datenbewegung gehören kann.

4.1.9 Funktionsprozessbezogene Datenanforderungen aus der Nutzungsfunktion

Bezüglich des Prinzips (c) für ein Entry (siehe Abschnitt 4.1.2) gibt es in dem Fall, wenn der funktionale Prozess Daten vom Benutzer erhalten muss, zwei Fälle. Wenn der funktionale Prozess dem Nutzer nicht mitteilen muss, welche Daten gesendet wurden, ist ein einfaches Entry pro betrachtetes Objekt ausreichend. Muss der funktionale Prozess dem Benutzer mitteilen, welche Daten gesendet wurden, dann ist ein Exit/Entry-Paar erforderlich. Dabei werden die folgenden Regeln angewandt.

Regeln für die funktionsprozessbezogenen Datenanforderungen aus der Nutzungsfunktion
a) Ein funktionaler Prozess soll eine Datengruppe über eine Entry-Datenbewegung von einem Benutzer erhalten. Wenn es nicht notwendig ist, dass der Benutzer mitteilt, welche Daten er sendet, liegt einer der folgenden vier Fälle vor: • ein Benutzer sendet ein auslösendes Ereignis, das einen funktionalen Prozess initiiert, • ein funktionaler Prozess, der ein auslösendes Entry empfangen hat, erwartet die Ankunft eines weiteren Entry von einem Benutzer (dieser Fall kann auch auftreten, wenn eine Benutzerperson Daten in eine Business-Applikation eingibt), • ein gestarteter funktionaler Prozess erfordert die Funktionalität, dass im Falle vorhandener Daten diese genau jetzt zu senden sind, was dann vom Benutzer auch erfolgt, • ein gestarteter funktionaler Prozess prüft erst die Situation des Benutzers und empfängt dann die geforderten Daten. In den letzten beiden Fällen, die typischerweise bei Echtzeit-Polling-Software auftreten, wird per Konvention kein Exit für den Erhalt der erforderlichen Daten identifiziert. Der funktionale Prozess benötigt lediglich eine Bestätigungsnachricht zum Benutzer, wobei die Funktionalität dieser Bestätigungsnachricht Teil des Entry darstellt. Der funktionale Prozess weiß dann, welche Daten zu erwarten sind. Nur ein Entry wird in diesem Falle gezählt. b) Wo ein funktionaler Prozess die Dienste eines Benutzers benötigt (zum Beispiel zum Erhalt von Daten) und der funktionale Prozess mitteilen muss, welche Daten zu senden sind (typischerweise wenn ein Benutzer ein anderer Software-Teil außerhalb des betrachteten, zu messenden Software-Bereichs darstellt), wird ein Exit/Entry-Paar identifiziert. Das Exit enthält die Anforderung spezifischer Daten und das Entry enthält die zurückgegebenen Daten.

Beispiel 1: (Bezüglich der ersten beiden Punkte der Regel (a)) Wenn ein funktionaler Prozess ein Bildschirminhalt für die Dateneingabe durch eine benutzende Person

bereitstellt, so ist dieser für die einzugebenden Daten formatiert und anderen Falls leer (mit Ausnahme vordefinierter Werte). Das ist wie in Online-Business-Applikationen, wo die Anzeige eines leeren Bildschirms nicht als ein Exit gezählt wird. Nur ausgefüllte Bildschirminhalte werden als ein oder mehrere Entries gezählt (siehe auch Abschnitt 4.1.10).

Beispiel 2: (Bezüglich der letzten beiden Punkte der Regel (a)) Gegeben sei der funktionale Prozess eines Echtzeitsteuerungs-Softwaresystems, das ein Feld einfachster Sensoren abfragt. Der funktionale Prozess erhält Daten über einen Entry, weil die Sensoren identisch sind und auch im Falle des mehrfachen Auftretens nur ein Entry identifiziert und gezählt wird. Es gelte weiterhin, dass das Entry praktisch Gerätetreiber-Software in einer unteren Schicht der Software-Architektur passieren muss und da von einem Sensor die erforderlichen Daten erhält, wie in Abbildung 2.3 gezeigt. Die funktionalen Prozesse der Steuerungs-Software und der Gerätetreiber-Software für die einfachen Sensoren zeigt dann die Abbildung 4.5 (a) und (b).

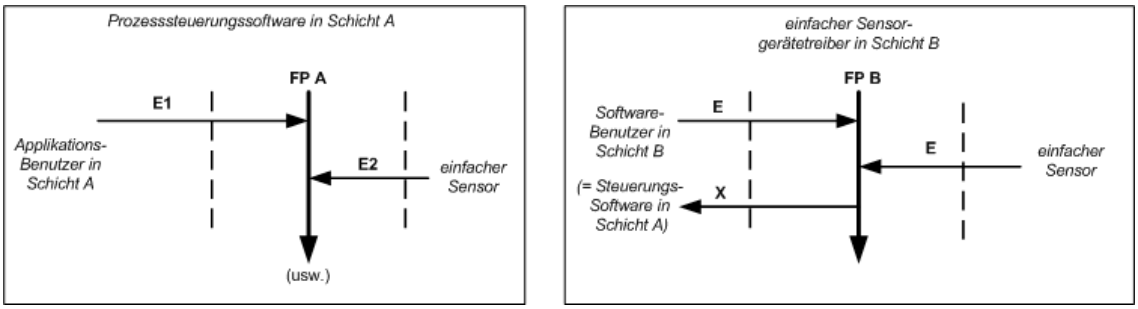

Abb. 4.5: Lösung für einen Eingang von der Software 'A' innerhalb der Steuerapplikationssoftware 'B' in der Gerätesensorschicht (links (a) und rechts(b))

Dabei zeigt die Abbildung 4.5 (a) die Steuerungs-Software des funktionalen Prozesses FP A, die durch das Ereignis E1 ausgelöst wurde, beispielsweise durch ein Zeitsignal. Der funktionale Prozess erhält dann den Entry E2 vom Sensorfeld, das die mehrfach auftretenden Sensorabfragen empfängt. Die Sensoren sind auch Benutzer der Steuerungs-Software.

Die Abbildung 4.5. (b) zeigt, dass die Software, die den Sensor betreibt, ein Entry (in der Realität sicherlich durch ein Betriebssystem) als Auslöser eines funktionalen Prozesses FP B empfängt. Dieser funktionale Prozess erhält ein Entry von seinem Benutzer, dem Sensor, um die Sensordaten, die zum Prozess durch die Software-Steuerung als Exit zurückgegeben werden, zu erhalten. Der durch die Steuerungs-Software definierte funktionale Prozess wird dann mit der Verarbeitung der Sensordaten fortgesetzt. Die Tatsache, dass in dieser zyklischen Datenübertragung dieselben Sensoren mehrfach einbezogen sind, ist für unser Modell irrelevant.

Der scheinbare Widerspruch zwischen dem einen Entry eines Steuerungs-Software-Prozesses und dem Entry/Exit-Paar für die Gerätetreiber-Software liegt an der Übereinkunft, dass ein Entry auch eine Enter-Anforderungsfunktionalität im Falle der durch den Benutzer nicht möglichen Beantwortung einer Nachricht vom funktionalen Prozess enthält.

Beispiel 3: (Bezüglich der Regel (b) Wir nehmen an, dass ein funktionaler Prozess zu einem seiner Benutzer als „intelligentes“ Gerät oder anderer Software einige Parameter für eine Anfrage oder einer Berechnung oder auch Datenkomprimierung sendet. Die Antwort des Benutzers wird dann über ein Exit/Entry-Paar erhalten wie es im Abschnitt 4.1.8 Beispiel 2 beschrieben wurde.

4.1.10 Steuerungskommandos

Ein „Steuerkommando“ ist eine spezielle Kategorie von Datenbewegung, die nur in Business-Applikationsbereichen betrachtet werden und die im funktionalen Prozess bei der Messung eigentlich zu ignorieren sind. Die Definition ist im Folgenden angegeben.

Definition eines Steuerkommandos
Ein Steuerkommando ist ein Kommando, dass dem Benutzer die Steuerung der Software gestattet, die allerdings keinerlei Datenbewegungen bezüglich des betrachteten Objektes einschließt. Hinweis: Der Ausdruck „Steuerkommando“ wird nur(!) im Kontext gemessener Business-Applikations-Software verwendet. In diesem Kontext ist ein Steuerkommando keine Datenbewegung, weil dieses Kommando keine Daten vom interessierenden Objekt bewegt. Beispiele sind Seiten-Scroll-Kommando, Tabellensprung oder Enter-Eingabe oder die OK-Bestätigung einer vorhergehenden Aktion usw.

Regeln für Steuerkommandos im Business-Applikationsbereich
Im Business-Applikationsbereich werden Steuerkommandos ignoriert, wenn sie keine Datenbewegungen zum interessierenden Objekt beinhalten.

Beispiele: Steuerkommandos im Business-Applikationsbereich sind Funktionen, die dem Benutzer beispielsweise die Steuerung des Bildschirms der Ausgabehinweise für Berechnungen, dem Navigieren (auf und ab bzw. zwischen Bildschirminhalten), der OK-Bestätigung von Fehlernachrichten oder der Bestätigung eingegebener Daten usw. dienen. Daher sind Steuerkommandos auch häufig Menükommandos, die dem Benutzer das Navigieren zu einem oder mehreren spezifischen, funktionalen Prozessen ermöglichen, die allerdings nicht selbst einen funktionalen Prozess initiieren oder Kommandos für die Anzeige leerer Bildschirminhalte für die Dateneingabe darstellen.

Anmerkung: Außerhalb des Business-Applikationsbereichs hat das Konzept der Steuerkommandos keine spezielle Bedeutung und ein Signal oder Bewegung von Daten zu einem interessierenden Objekt eines Benutzers muss hierbei gezählt bzw. gemessen werden.

4.2 Die Anwendung der Messfunktion

Dieser Schritt beinhaltet die Anwendung der COSMIC-Messfunktion für jede der in einem funktionalen Prozess identifizierten Datenbewegung.

Definition der COSMIC-Messfunktion
Die COSMIC-Messfunktion ist eine mathematische Funktion, die eindeutig einen Wert zu seinem Argument auf der Grundlage des COSMIC-Messstandards zuordnet. Das Argument der COSMIC-Messfunktion ist die Datenbewegung.

Definition des COSMIC-Messstandards
Der COSMIC-Messstandard, als 1 CFP (Cosmic Function Point), ist definiert als der Umfang einer Datenbewegung.

Bezüglich dieser Messfunktion ist jede Instanz einer Datenbewegung (Entry, Exit, Read oder Write) notwendigerweise hinzuzufügen, zu ändern oder abzuziehen und besitzt gemäß Abschnitt 4.1 jeweils den numerischen Umfang von 1 CFP.

4.3 Die Aggregation von Messergebnissen

Dieser Schritt beschäftigt sich mit der Aggregation der Ergebnisse der COSMIC-Messung wie sie auf alle identifizierten Datenbewegungen angewandt wurde. Er soll auf der Grundlage der folgenden Regeln durchgeführt werden.

4.3.1 Allgemeine Aggregationsregeln

Regeln für die Aggregation von Messergebnissen

a) Für jeden funktionalen Prozess wird der Funktionsumfang aller einzelnen Datenbewegungen in folgender Weise die einzelnen Funktionsumfangswerte berücksichtigend mit der Einheit CFP arithmetisch zusammengefasst.

$$Umfang(funktionaler\ Prozess_i) = \sum Umfang(Entries_i) + \sum Umfang(Exits_i)$$
$$\sum Umfang(Reads_i) + \sum Umfang(Writes_i)$$

b) Für jeden funktionalen Prozess wird der Funktionsumfang von Änderungen auf der Grundlage der Funktionalen Nutzeranforderungen den Umfang der Datenbewegungen, die im funktionalen Prozess hinzugefügt, modifiziert oder gelöscht wurden und einen geänderten Umfang in der CFP-Einheit ergeben, nach folgender Formel aggregiert.

$$Umfang(Änderung(funktionaler\ Prozess_i)) = \sum Umfang(hinzugefügter\ Datenbewegungen_i) + \sum Umfang(modifizierter\ Datenbewegungen_i) + \sum Umfang(gelöschter\ Datenbewegungen_i)$$

Weiteres über die Aggregation des Funktionsumfangs ist im Abschnitt 4.3.2 angegeben. Bezüglich der Messung geänderter Software siehe Abschnitt 4.4.

c) Der Umfang eines Software-Anteils in einem definierten Systembereich wird durch die Aggregation aller Umfänge in den funktionalen Prozessen dieses Anteils (unter Beachtung der unten angegebenen Regeln (e) und (f)) gebildet.

d) Der Umfang der Änderungen eines Software-Anteils in einem definierten Systembereich wird durch die Aggregation der Umfänge aller Änderungen in den funktionalen Prozessen dieses Anteils (unter Beachtung der unten angegebenen Regeln (e) und (f)) gebildet.

e) Der Umfang eines Software-Teils oder dessen Änderungen innerhalb von Schichten kann nur innerhalb desselben Funktionsprozessniveaus in der Granularität der FUR zusammengefasst werden.

f) Der Umfang eines Software-Teils und/oder dessen Änderungen innerhalb einer Schicht oder von unterschiedlichen Schichten wid nur zusammengefasst, wenn es bezüglich des Zwecks der Messung auch sinnvoll ist.

g) Der Umfang eines Software-Teils kann nicht durch Zusammenfassung der Umfänge seiner Komponenten (bezüglich der Dekomposition der Einheiten) gebildet, es sei denn, die Umfangsaufteilung bezüglich der Datenbewegungen zusammenhängender Komponenten wurde eliminiert.

h) Wenn die lokalen Erweiterungen der COSMIC-Methode angewandt werden (zum Beispiel, wenn Umfangsaspekte, die nicht im Standard enthalten sind, gemessen werden sollen), muss der jeweils gemessene Umfang separat wie in Abschnitt 5.1 beschrieben, ausgewiesen werden. Er kann nicht(!) einfach nach dem üblichen Vorgehensweise addiert und in CFP ausgedrückt werden (siehe auch Abschnitt 4.5).

Beispiel 1: (Bezüglich der Regeln (b) und (c)) Eine notwendige Änderung für eine Software kann sein: „füge eine neue Funktionalität mit einem Umfang von 6 CFP hinzu und in einem anderen funktionalen Prozess addiere eine Datenbewegung, modifiziere drei andere Datenbewegungen und lösche zwei Datenbewegungen“. Der Gesamtumfang der erforderlichen Änderung ergibt dann 6 + 1 + 3 + 2 = 12 CFP.

Beispiel 2: (Bezüglich der Regel (f)) Wenn verschiedene, aber wesentliche Teile der Software mit unterschiedlichen Technologien entwickelt wurden, beispielsweise

durch unterschiedliche Projektteilteams, so kann man praktisch die einzelnen Umfänge nicht einfach addieren.

Beispiel 3: (Bezüglich der Regel (g)) Wenn ein Software-Teil

- zuerst als Ganzes gemessen wurde, beispielsweise innerhalb eines Bereiches,
- dann als Zweites der Umfang jeder seiner Komponenten separat gemessen wird, also in seinem jeweiligen eigenen Bereich,

so wird der Gesamtumfang durch die Addition des Umfangs aller separaten Komponenten (also im zweiten Fall) dem Umfang der Messung der Software als Ganzes (also im ersten Fall) durch die Einbeziehung aller Datenbewegungen als Umfang der Komponentenbeziehungen übersteigen. Diese Datenbewegungen der Komponentenbeziehungen sind bei der Messung der Software als Ganzes nicht sichtbar und müssen daher eliminiert werden, wenn man „nur“ den Gesamtumfang erhalten will. Dazu siehe auch den Abschnitt zur Messung in unterschiedlichen Granularitätsstufen in Software-Architekturen im COSMIC-Dokument „Advanced and Related Topics“.

Es ist darauf hinzuweisen, dass innerhalb einer festgelegten Schicht die Aggregationsfunktion auch korrekt skaliert. Dafür können Teilumfänge für individuelle funktionale Prozesse oder für die gesamte Software in dieser Schicht gebildet werden, in Abhängigkeit vom Zeck und Bereich jeder Messung gemäß den obigen Regeln (d), (e) und (f).

4.3.2 Mehr über die Aggregation des Funktionsumfangs

In dem Kontext, wo der Funktionsumfang als eine Modellgröße angewandt wird, z. B. für die Aufwandsschätzung, und die zu messende Software sich über mehr als eine Schicht oder eine Komponente verteilt, muss die Aggregation korrekterweise pro Schicht oder pro Komponente durchgeführt werden, da sie zumeist nicht in derselben Technologie implementiert wurden.

Beispiel 1: Betrachtet sei eine Software, bei der die Applikationsschicht unter Verwendung von 3GL-Sprachen und einer Menge existierender Bibliotheken entwickelt wurde, während man die Geräteschicht unter Verwendung einer Assemblersprache implementiert hat. Der Aufwand für die Konstruktion beider Schichten ist sehr wahrscheinlich unterschiedlich, und konsequenterweise ist eine den Umfang berücksichtigende Aufwandschätzung auch separat für jede Schicht durchzuführen.

Beispiel 2: Wenn ein Projektteam eine Anzahl von wesentlichen Software-Teilen entwickelt hat und an der insgesamt vorhandenen Produktivität interessiert ist, so kann es die Arbeitsstunden für jeden entwickelten Software-Teil addieren. In gleicher Weise können auch die Umfänge zusammengefasst werden, vorausgesetzt, man beachtet die Erfüllung der oben angegeben Regeln.

Der Grund dafür, dass die Umfänge bzw. Größe wesentlicher Software-Teile von unterschiedlichen Schichten einer Standard-Schichtenarchitektur, die auf demselben Funktionsprozessniveau der Granularität gemessen wurden, einfach addiert werden können, ist darin begründet, dass eine derartige Architektur auch in dieser Weise die damit zusammenhängenden Benutzer definiert. Jede Schicht ist ein Benutzer der darunter liegenden Schichten und irgendeine Software kann wiederum Benutzer einer der Softwarekomponenten in einer Schicht sein. Die Anforderungen an eine solche Architektur beinhalten auch, dass die FUR der unterschiedlichen Software-Teile auch den Austausch von Nachrichten fordern. Es ist daher nur logisch und nachvollziehbar, dass auch der Umfang der unterschiedlichen Software-Teile addiert werden kann, wobei stets die obigen Regeln (d), (e) und (f) zu beachten sind. Im Gegensatz dazu wird der Umfang wesentlicher Software-Teile nicht durch einfache Addition erhalten, es sei denn, die wiederverwendeten Objekte oder die zwischen den Objekten vorhandenen Datenbewegungen sind im Sinne der Regel (f) eliminiert.

Die Aggregation der Messergebnisse nach den jeweiligen Datenbewegungstypen kann für die Analyse der Verteilung dieser Typen hinsichtlich des Gesamtumfangs einer Schicht in der Weise hilfreich sein, dass dadurch auch die Funktionalität der gemessenen Schicht überhaupt erklärt bzw. transparent wird.

4.4 Mehr zur Umfangsmessung von Software-Änderungen

Eine Funktionsänderung zu einer vorhandenen Software wird nach der COSMIC-Methode definiert als „irgendeine Kombination von Hinzufügen neuer Datenbewegungen oder Modifikationen oder Löschungen von existierenden Datenbewegungen“. Die Ausdrücke „Weiterentwicklung“ oder „Wartung“[4] werden häufig für die hier beschriebene Funktionsänderung verwandt.

Der Bedarf zur Änderung von Software kann entweder herrühren

- von einer neuen FUR (d. h. eine reine Funktionserweiterung), oder
- von einer Änderung FUR (vielleicht das Hinzufügen, Modifizieren oder Löschen eingeschlossen) oder
- von eine Wartung im Sinne einer notwendigen Fehlerkorrektur.

Die Regeln für die Umfangsbestimmung dieser Änderungen sind dieselben, aber die Messperson muss im Falle einer Leistungsmessung oder (Kosten-/Aufwand-) Schätzung genau die jeweiligen zu unterscheidenden Umstände berücksichtigen.

Wenn ein Software-Teil vollständig ersetzt wird, beispielsweise bei einer Neuimplementation dieses Teils mit oder ohne Funktionserweiterung, so ist der Funktionsumfang dieser Änderung der Umfang der ersetzten Software, die normalerweise nach den Regeln für neu erstellte Software gemessen wird. Dieser Fall wird in diesem Abschnitt nicht behandelt. Die Messperson sollte sich aber darüber im Klaren sein, dass im Falle von Performance-

[4] Eine übliche Vorgehensweise bei der Messung ist die, dass der Funktionsumfang eines Software-Teils sich nicht ändert, wenn nur eine Fehlerkorrektur im Sinne der FUR erforderlich ist. Der Funktionsumfang ändert sich allerdings in dem Falle, wenn eine Änderung die Korrektur eines Fehler in der FUR selbst betrifft.

Messungen oder Schätzungen die Unterschiede zwischen den Projekten, die jeweils neu entwickelt werden und Projekten die weiterentwickelt oder in Software-Teilen ersetzt werden, entsprechend zu berücksichtigen sind.

Oftmals wird ein veralteter Teil einer Applikation gelöscht („die Verbindung aufgehoben“ wäre eine bessere Beschreibung) durch Verlassen des Programm-Codes an dieser Stelle und Ersetzen der Verbindung zur veralteten Funktionalität. Wenn beispielsweise die Funktionalität des veralteten Software-Teils 100 CFP beträgt, aber dieser Teil durch eine Änderung von angenommen 2 Datenbewegungen entkoppelt wird, so sind dennoch die 100 und nicht die 2 Datenbewegungen für den Umfang der Änderung zu identifizieren. Wir messen den Umfang einer Anforderung und nicht den Umfang dessen, was implementiert wurde[5].

Man beachte dabei die hier diskutierte Differenz zwischen dem Umfang der Funktionsänderung und der Änderung des Funktionsumfangs der Software. Üblicherweise sind diese unterschiedlich. Der Umfang der letzteren wird in Abschnitt 4.4.3 behandelt.

4.4.1 Modifikation der Funktionalität

Jede Datenbewegung eines bestimmten Typs (E, X, R und W) schließt stets zwei Funktionstypen ein: sie bewegt eine einfache Datengruppe und sie hat spezielle, mit ihr verbundene Datenmanipulationen (siehe auch Abschnitt 4.1.6). Daher wird für den Zweck einer Messung die Datenbewegung funktional modifiziert, wenn

- die bewegte Datengruppe bewegt und/oder
- die mit ihr verbundene Datenmanipulation

in irgendeiner Weise modifiziert werden.

Eine Datengruppe gilt dabei als modifiziert, wenn

- neue Attribute zu dieser Datengruppe hinzugekommen sind und/oder
- bereits existierende Attribute aus einer Datengruppe entfernt wurden und/oder
- ein oder mehrere, bereits existierende Attribute modifiziert wurden (beispielsweise in ihrer Bedeutung oder Formatierung, ohne jedoch ihre Werte zu verändern).

Eine Datenmanipulation gilt als modifiziert, wenn in irgendeiner Weise ihre Funktionalität verändert wurde, beispielsweise durch Änderung der Berechnungen, spezieller Formatierungen, Darstellungen und/oder einer Datenprüfung. Dabei kann die „Darstellung“ z. B. die Schriftart, Hintergrundfarbe, Feldlänge oder Dezimalstellenanzahl usw. bedeuten.

[5] Man beachte, dass für Schätzungen die Verwendung unterschiedlicher Produktivitäten bei der Änderung sinnvoll sein kann, da Verbindungsauflösung völlig unterschiedlich zu „echten“ Löschfunktionen sein kann. Im Gegensatz dazu kann es beispielsweise sein, dass für die Schätzung der Umfang der Implementaion z. B. 2 CFP sein kann, anstelle der angenommenen 100 CFP im Beispiel. Wenn der „Projektumfang“ als 2 CFP gemessen wurde, sollte dies auch klar dokumentiert werden und dem Unterschied zur Messung der FUR (als Reduzierung um 100 CFP) klar herausstellen.

Steuerkommandos und allgemeine Applikationsdaten werden nicht in die Datenbewegungen mit einbezogen, sofern sie nicht das betrachtete Objekt verändern. Daher gehen auch die Steuerkommandos und die allgemeinen Applikationsdaten nicht in die Messung ein. Ein Beispiel dafür ist, wenn sich die Bildschirmfarbe für alle Bildschirminhalte verändert. Diese Änderungen sollten nicht gemessen werden (siehe Abschnitt 4.1.10 zur Beschreibung von Steuerkommandos und allgemeinen Applikationsdaten).

Regeln für die Modifikation einer Datenbewegung
a) Wenn eine Datenbewegung aufgrund der Änderung der assoziierten Datenmanipulation zu modifizieren ist und/oder sich die Anzahl oder der Typ der Datenattribute einer Datenbewegung ändert, so muss auch ein geändertes CFP gemessen werden, (wobei die aktuelle Anzahl der Modifikationen einer Datenbewegung zu berücksichtigen ist). b) Wenn eine Datengruppe zu modifizieren ist und diese Modifikation beeinflusst nicht die Funktionalität der Datenbewegungen für die modifizierte Datengruppe, so wird keine Datenbewegungsänderung identifiziert bzw. gezählt. Hinweis 1: Das Auftreten einer Modifikation eines Attributwertes beispielsweise in Form einer Änderung eines speziellen Code-Elements eines Attributs in einem Code-System stellt keine Änderung des Attributtyps dar. Hinweis 2: Eine Modifikation von in einem Ein- oder Ausgabebildschirm auftretende Daten, die sich nicht auf das für den Benutzer interessierende Objekt beziehen, wird nicht als CFP-Änderung identifiziert (siehe auch Abschnitt 3.3.4 bezüglich derartiger Datenbeispiele).

Beispiel: (Bezüglich der Regeln (a) und (b)) Es sei eine Anforderung als Hinzufügen oder Modifizieren der Datenattribute einer Datengruppe D_1 gegeben, die nach der Modifikation eine Datengruppe D_2 erzeugt. Für den funktionalen Prozess A, in dem diese Modifikation gefordert wird, werden dann alle diese Modifikation betreffenden Datenbewegungen auch als modifiziert identifiziert und gezählt. Gemäß der Regel (a) gilt, wenn die modifizierte Datengruppe D_2 persistent gemacht werden soll und/oder eine Ausgabe des funktionalen Prozesses A darstellt, so ist ein Write und/oder eine Exit-Datenbewegung als modifiziert zu identifizieren. Es ist jedoch auch möglich, dass andere funktionale Prozesse als Read oder Entry dieselbe Datengruppe D_2 betreffen, aber durch die Modifikation keine Funktionalität bewirken, da sie nicht die geänderten oder hinzugefügten Datenattribute verwenden. Diese funktionalen Prozesse verarbeiten die geänderte Datengruppe so wie es noch auf der Basis von D_1 üblich ist. Gemäß der Regel (b) werden diese Datenbewegungen anderer funktionaler Prozesse, die nicht durch die Modifikation der Datenbewegungen des funktionalen Prozesses beeinflusst sind, auch nicht(!) als modifiziert identifiziert bzw. gezählt.

4.4.2 Der Umfang funktional geänderter Software

Nach einer Änderung der Funktionalität eines Software-Teils ist der Gesamtumfang der ursprüngliche Umfang plus dem Funktionsumfang aller hinzugefügten Datenbewegungen und minus dem Funktionsumfang der entfernten Datenbewegungen. Modifizierte Datenbewegungen haben keinen Einfluss auf den Umfang eines Software-Teils, ganz gleich ob er vor oder nach eine Modifikation betrachtet wird.

4.5 Erweiterung der COSMIC-Messmethode

4.5.1 Einführung

Die COSMIC-Messmethode zurMessung des Funktionsumfangs beansprucht nicht, alle Aspekte der „Größe“ einer Software zu messen. Daher ist die COSMIC–Messmethode gegenwärtig nicht entworfen worden, um eine standardisierte Form des Umfangs spezieller Funktionaler Nutzeranforderungen, wie beispielsweise komplexe mathematische Algorithmen oder komplexe logische Ableitungen in Expertensystemen, zu berücksichtigen. So wird zum Beispiel auch der Einfluss der Anzahl der Datenattribute pro Datenbewegung für die Software-Größe nicht durch diese Methode behandelt. Der Einfluss des Umfangs von Datenmanipulationen in Teilprozessen wird auf der Grundlage von vereinfachenden Annahmen ermöglicht, die für spezielle Anwendungsbereiche gelten, wie sie in Abschnitt 1.1.1 zur Anwendbarkeit der COSMIC-Methode definiert sind.

Weitere Parameter, wie die Komplexität, könnten in ihrer Auswirkung auf den Funktionsumfang berücksichtigt werden, vorausgesetzt, die Komplexitätsart ist eindeutig definiert. Dazu sind allerdings konstruktive Diskussionen erforderlich, die ein Einvernehmen einer derartigen Definition voraussetzen würde. Das sind allerdings noch auch weitere Forschungsaufgaben.

Nichtsdestoweniger kann die COSMIC-Methode als gute Annäherung einer Methode für den genannten Zweck und Anwendungsbereich charakterisiert werden. Bei der Anwendung der COSMIC-Methode in einem speziellen Organisationsumfeld kann es notwendig sein, die Methode den lokalen Erfordernissen anzupassen. Für diese Anforderung besitzt die COSMIC-Methode spezielle Ansatzpunkte. Wird die Methode in dieser Weise verändert, so sollte allerdings die Darstellungsform nach den im Abschnitt 5.1 beschriebenen Konventionen erfolgen.

Die folgenden beiden Abschnitte beschreiben, wie die COSMIC-Methode zu einem lokalen Standard erweitert werde kann.

4.5.2 Lokale Erweiterungen mit komplexen Algorithmen

Wenn es notwendig ist, komplexe Algorithmen einzubeziehen, kann ein lokaler Standard für diese Funktionserweiterung vorgenommen werden. In dem funktionalen Prozess, wo es eine besonders komplexe Datenmanipulation funktionaler Teilprozesse gibt, kann die Messperson nach eigenem Ermessen lokalbezogene Function Points einführen.

Beispiel: Eine lokale Standarderweiterung könnte sein: „In unserer Organisation ist ein Local FP für mathematische Algorithmen in folgender Weise zugeordnet“ (Liste

von gut verständlichen Beispielen) „Zwei Local FP werden gezählt im Falle“ (andere Liste aussagekräftiger Beispiele) usw.

4.5.3 Lokale Erweiterungen mit Teileinheiten der Messung

Wenn mehr Genauigkeit bei der Messung der Datenbewegungen erforderlich ist, so können Teileinheiten unseres Maßes definiert werden. Zum Beispiel kann man ein Meter in 100 Zentimeter und 1000 Millimeter unterteilen. Analog könnte die Bewegung eines einzelnen Datenattributs beispielsweise als Teileinheit verwendet werden. COSMIC-Messungen kleiner Software-Beispiele haben im praktischen Umfeld gezeigt, dass bei den Messkomponenten die durchschnittliche Anzahl der Datenattribute pro Datenbewegung bei den einzelnen vier Datenbewegungstypen kaum variierten. Aus diesem Grund und für eine Vereinfachung der Messung überhaupt wurde die COSMIC-Maßeinheit, 1 CFP, für eine Datenbewegung festgelegt. Jedoch kann es beim Vergleich gemessener Umfänge in CFP für unterschiedliche Software-Teile auch zeigen, dass diese Durchschnittsanzahl der Datenattribute stark differiert.

Wenn jemand eine Verfeinerung der Maßeinheit bei der COSMIC-Methode wünscht, so kann dies auch ohne weiteres umgesetzt werden. Allerdings ist dann auch die damit verbundene Differenz zu den üblicherweise gemessenen COSMIC Function Points zu beachten.

5. Der Messbericht

Das Generische Software-Modell kann in einer Matrix dargestellt werden, bei der die Zeilen die funktionalen Prozesse (kann nach Schichten gruppiert sein) und die Spalten die Datengruppen darstellen, während die einzelnen Felder die identifizierten Teilprozesse (Entry, Exit, Read oder Write) beinhalten. Diese Darstellungsform des Generischen Software-Modells ist im Anhang A angegeben.

Die COSMIC-Messergebnisse sollten nach folgenden Konventionen dargestellt und archiviert werden.

5.1 Kennzeichnung

Wenn ein Messbericht zum COSMIC-Funktionsumfang angefertigt werden muss, so sollte er die folgenden, dem ISO/IEC14143-1:2007 Standard entsprechenden Konventionen berücksichtigen.

Regeln für die Kennzeichnung der COSMIC-Messung
Ein COSMIC-Messresultat sollte notiert sein in der Form „***x*** CFP (v.***y***)“, wobei • „***x***“ den numerischen Wert des Funktionsumfangs darstellt und • „v.***y***“ den Messstandard identifiziert und die Version der verwendeten COSMIC-Methode für die der Funktionsumfangswert „x“ erhalten wurde, kennzeichnet. Hinweis: Im Falle einer lokal erweiterten Messmethodenanwendung, allerdings unter Anwendung der generellen COSMIC-Methodik, sollte doch die zugrunde gelegte COSMIC-Version angegeben werden und die speziellen Erweiterungen anderweitig notiert werden (siehe auch Abschnitt 5.2).

Beispiel A: Ein erhaltenes Messergebnis unter Verwendung dieses Messhandbuchs wurde notiert als ’x CFP (v3.0.1)’.

Sollten lokale Erweiterungen angewandt worden sein, wie sie in Abschnitt 4.5 beschrieben wurden, sollen die Messergebnisse wie folgt beschrieben werden.

Regeln für die lokale COSMIC-Kennzeichnungserweiterung
Ein COSMIC-Messergebnis basierend auf lokalen Methoden-Erweiterungen sollte notiert werden als „***x*** CFP (v.***y***) + ***z*** Local FP, wobei • „***x***“ den numerischen Wert des Funktionsumfangs darstellt, den man erhält durch die Aggregation aller individuellen Messergebnisse gemäß dem COSMIC-Standard Version v.y,

- „v.***y***“ den Messstandard identifiziert und die Version der verwendeten COSMIC-Methode für die der Funktionsumfangswert „x“ erhalten wurde, kennzeichnet,

- „***z***“ den numerischen Wert darstellt, der sich durch Aggregation der individuellen Messergebnisse aus den lokalen Erweiterungen zur COSMIC-Methode ergibt.

5.2 Archivierung der COSMIC-Messergebnisse

Für die Archivierung der COSMIC-Messergbnisse sollten die folgenden Hinweise berücksichtigt werden, um stets interpretierbare Ergebnisse zu haben.

Regeln für den COSMIC-Messbericht

Als Ergänzung zu aktuellen Messungen, wie sie in 5.1. darzustellen sind, sollten auch die folgenden Attribute zu jeder Messung mit aufgenommen werden:

a) die Identifikation der gemessenen Software-Komponente (Name, Versions-ID oder Konfigurations-ID),

b) die Informationsquellen , die für die Identifikation der FUR für die Messung verwendet wurden,

c) der Software-Applikationsbereich,

d) eine Beschreibung des Zwecks der Messung,

e) eine Beschreibung des Messbereichs und seine Beziehung zum Kontext möglicher anderer, damit zusammenhängender Messungen (man verwende dabei die Bereichsbeschreibung vom Abschnitt 2.2),

f) die Benutzer der Software,

g) das Granularitätsniveau der FUR und das Niveau der Dekomposition der Software,

h) den Zeitpunkt im Projekt-Lifecycle, wann die Messung gemacht wurde (speziell wenn die Messung für die Aufwandschätzung auf der Grundlage unvollständiger FUR oder auf Basis der aktuell vorhandenen Funktionalität durchgeführt wurde),

i) das Ziel bzw. mögliche Fehlerannahmen für die Messung,

j) Hinweise, ob die (reine) COSMIC-Methode oder lokale Erweiterungen verwendet wurden (siehe Abschnitt 4.5 bzw. 5.1 und 5.2),

k) Hinweise darüber, ob entwickelte oder angepasste Funktionalität gemessen wurde (entwickelte Software ist die Neuentwicklung und angepasste Software schließt auch Wiederverwendung und Formen der Anpassen mit ein),

l) ein Hinweis, ob neu hinzugefügte Funktionalität oder Erweiterungen gemessen wurden (beispielsweise Zusammenfassungen von hinzugefügter, geänderter oder gelöschter Funktionalität (siehe 4.4)),

m) eine Beschreibung der Schichtenarchitektur in der die Messung durchgeführt wurde,

n) die Anzahl der wesentlichen bzw. Haupt-Komponenten, deren Umfang für die Bestimmung des Gesamtumfangs addiert wurde,

o) für jeden Bereich innerhalb des allgemeinen Messbereichs eine Messresultatsmatrix, wie sie im Anhang A beschrieben ist,

p) den Namen der Messperson und seine COSMIC-bezogenen (zertifizierten) Qualifikationen.

Anhang A

Dokumentierung der COSMIC-Umfangsmessung

Die unten angegebene Struktur kann als ein Repository für die Speicherung der Messergebnisse für jede identifizierte Komponente eines gemäß dem Generischen Software-Modell betrachteten Bereiches angewandt werden. Jeder Teilbereich des gesamten Messbereiches hat dabei seine eigene Matrix.

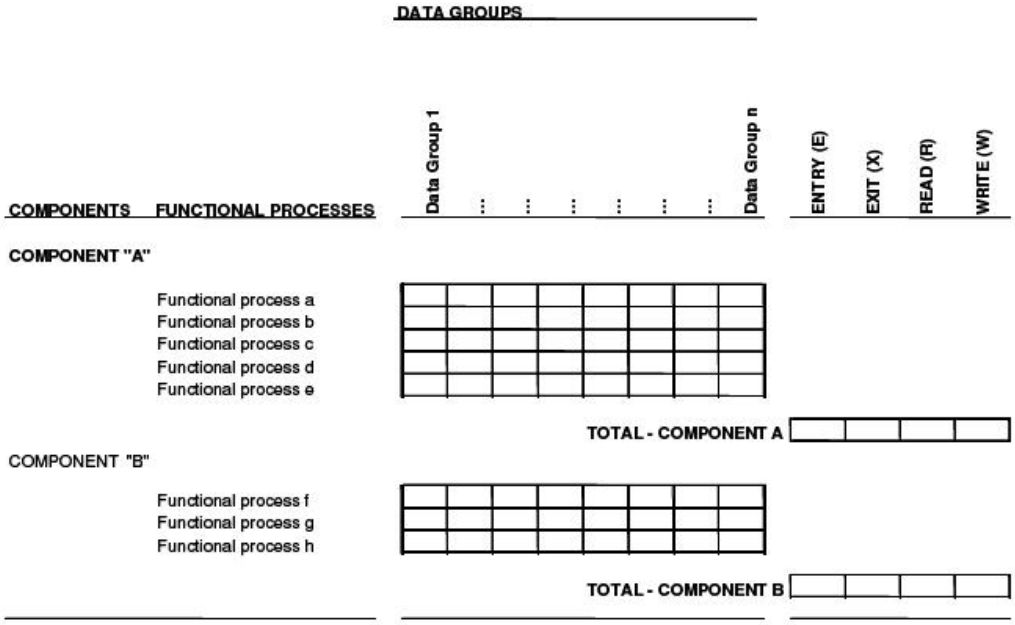

Abbildung A: Matrix zum Generischen Software-Modell

Abbildungsphase

- Jede identifizierte Datengruppe ist in einer Spalte registriert.
- jeder funktionale Prozess ist in einer speziellen Zeile registriert, die nach den identifizierten Komponenten gruppiert ist

Messphase

- Für jeden identifizierten, funktionalen Prozess sind die gezählten Datenbewegungen im jeweiligen Kästchen mit der Bezeichnung „E“ für Entry, „X“ für Exit, „R“ für Read und „W“ für Write eingetragen.
- Für jeden identifizierten, funktionalen Prozess sind dann die Datenbewegungen nach dem jeweiligen Typ in den rechten Matrixspalten zusammengefasst.
- Die Gesamtheit der Messungen kann dann in den „Total“-Kästchen zu jeder Komponente eingetragen werden.

Anhang B

Übersicht zu den COSMIC-Methodenprinzipien

Prinzipien des COSMIC-Software-Kontextmodells
a) Software ist stets mit Hardware verbunden.
b) Software ist typischerweise in **Schichten** (layers) strukturiert.
c) Eine Schicht kann aus ein oder mehreren verbundenen Software-Komponenten (**peer** pieces of software) bzw. die Software kann selbst aus separaten Teilkomponenten bestehen.
d) Eine zu messende Software-Einheit (peer component) soll stets durch ihre Messobjektcharakterisierung (measurement **scope**) definiert werden, welche sich vollständig in einem einfachen Layer befinden.
e) Die Messobjektcharakterisierung für die zu messende Software sollte sich auf den **Zweck der Messung** beziehen.
f) Die Nutzer- bzw. **Anwendungsfunktion** der Software aus den zu messenden funktionalen Anforderungen sollten als Sendung und/oder Empfang von Daten identifiziert werden.
g) Software interagiert mit einem Nutzer in Form von **Datenbewegungen** (data movements) im Rahmen von **Abgrenzungen** (boundaries) und kann Daten zu und von einer **persistenten Speicherung** innerhalb dieser Abgrenzung bewegen.
h) Die FUR der Software kann in verschiedenen **Stufen der Granularität** ausgedrückt werden.
i) Die Stufe bzw. das Niveau dieser Granularität, in der die Messung durchgeführt wird, sind die **funktionalen Prozesse**.
j) Wenn es nicht möglich ist, auf dem jeweiligen Granularitätsniveau zu messen, sollten die zu messenden FUR näherungsweise entsprechend bestimmt werden.

Prinzipien des Generischen COSMIC-Software-Modells
a) Software empfängt Eingabedaten des Nutzers und produziert **Ausgabedaten** und/oder andere Ausgaben bzw. Ausgabesignale für den Nutzer.
b) Funktionale Systemanforderungen der zu messenden Software kann in eindeutige Funktionsprozesse abgebildet werden.
c) Jeder funktionale Prozess besteht aus **Teilprozessen**.
d) Ein Teilprozess kann sowohl eine **Datenbewegung** (data movement) oder **Datenmanipulation** (data manipulation) sein.
e) Jeder funktionale Prozess ist mit einem **Eingang** (Entry) zur Datenbewegung aus einer Nutzerfunktion verbunden, der dem Prozess signalisiert, das in der Nutzerfunktion ein **Ereignis** stattfindet.

f) Eine Datenbewegung „bewegt“ eine einfache **Datengruppe**.

g) Eine Datengruppe besteht aus einer eindeutigen Menge von **Datenattributen**, die ein einfaches, zu betrachtendes Objekt beschreibt.

h) Es gibt vier Typen von Datenbewegungen bzw. Datenbehandlungen. Der **Eingang** (Entry) bewegt Daten von der Anwendung bzw. dem Nutzer in die Software. Der **Ausgang** (Exit) bewegt wiederum eine Datengruppe aus der Software zum Nutzer. Ein **Ausschreiben** (Write) bewegt eine Datengruppe aus der Software in eine persistente Speicherungsform. Ein **Einlesen** (Read) bewegt eine Datengruppe aus der persistenten Speicherung in die betrachtete Software.

i) Ein funktionaler Prozess sollte stets aus mindestens einem Eingang bzw. einem Einschreiben und einem dazugehörigen Ausgang bzw. Ausschreiben bestehen. Das bedeutet, dass ein funktionaler Prozess aus **mindestens zwei Datenbewegungen** besteht.

j) Als eine Näherung für den Messprozess sind Datenmanipulationen für die Teilprozesse nicht separat zu messen; die **Funktionalität einer Datenmanipulation** sollte **über** damit **zusammenhängende Datenbewegungen bestimmt** werden.

Prinzipien für die Schichtenbildung

e) Die Software einer Schicht tauscht Daten mit der Software einer anderen Schicht auf der Grundlage funktionaler Prozesse aus.

f) Die Hierarchiebezogene Abhängigkeit zwischen Schichten kann darin bestehen, dass eine Schicht die funktionalen Dienste der Software einer anderen, in der Hierarchie darunter liegenden Schicht, benutzt. Bei einer derartigen Anwendungsbeziehung bezeichnen wir die verwendete Softwareschicht als „übergeordnet“ und die anwendende Softwareschicht als „untergeordnet“. Die Software in der übergeordneten Schicht verlässt sich auf den Dienst der untergeordneten Schicht für einen korrekten Ablauf, wobei der korrekte Ablauf wiederum sich auf die darunter liegenden Schichten bezieht usw. Umgekehrt kann allerdings die Software der untergeordneten Schicht mit den davon abhängigen Schichten arbeiten, ohne die Dienste der übergeordneten Schichten zu benötigen.

g) Die Software einer Schicht muss nicht notwendigerweise alle Funktionalitäten der durch die untergeordnete Schicht angebotenen Dienste benutzen.

h) Die Daten, die als Austausch zwischen der Software zweier Schichten definiert ist, kann unterschiedlich bezüglich der FUR dieser beiden Software-Komponenten interpretiert werden. Dies bedeutet, dass bei den beiden Software-Komponenten unterschiedliche Attribute und/oder Teilgruppierungen der ausgetauschten Daten betrachtet werden können. Jedoch müssen dabei auch ein oder mehrere gemeinsame Datenattribute bzw. Datengruppen definiert sein, die von der jeweiligen Schicht im Sinne der bestehenden Funktionalität empfangen bzw. gesendet werden können.

Prinzipien der betrachteten Komponenten
d) In einer Menge betrachteter Komponenten eines Softwarebereichs in einer Schicht gibt es keine hierarchischen Abhängigkeiten zwischen diesen Komponenten als die durch die Schichten selbst bestimmt sind. e) Alle betrachteten Komponenten eines Softwarebereichs kooperieren miteinander in der Weise, dass der jeweilige Softwarebereich erfolgreich ausgeführt werden kann. f) Die FUR der betrachteten Komponenten einer Schicht, die gemeinsam Daten übertragen bzw. verteilen, definieren diese Daten identisch, d. h. sie erkennen dieselben Datenattribute und -strukturen dieser Daten, die sie weitergeben bzw. verteilen.

Prinzipien für die Anwendung des Generischen COSMIC-Software-Modells
Das Generische COSMIC-Software-Modell sollte für jeden Software-Teil, für den ein separater Messbereich definiert wurde, auch separat auf die funktionalen Anforderungen angewandt werden. Die Anwendung des Generischen COSMIC-Software-Modells bedeutet auch, die Menge der auslösenden Ereignisse, die für jeden Nutzerfunktionstyp, die in der FUR bestimmt wurden und die dann den jeweiligen funktionalen Prozess, die relevanten Objekte, die Datengruppen und die für diese Ereignisse unterstützenden Datenbewegungen, zu bestimmen.

Prinzipien zur Datengruppe
d) Jede identifizierte Datengruppe sollte eindeutig und wohlunterschieden hinsichtlich seiner Datenattribute sein. e) Jede Datengruppe sollte sich direkt auf ein interessierendes Objekt aus den funktionalen Nutzeranforderungen beziehen. f) Eine Datengruppe sollte innerhalb eines die jeweilige Software unterstützenden Computersystems „materialisiert“ (s. u.) sein.

Prinzipien des Einganges (Entry E)
a) Eine Eingabe soll stets eine einfache Datengruppe übertragen, die ein einfaches interessierendes Objekt beschreibt. Die Übertragung erfolgt dabei von einem funktionalen Prozess über eine Abgrenzung in den Prozess, der den Entry für die Funktionalität benötigt. Erfordert der Input eines funktionalen Prozesses mehrere Datengruppen, soll ein Entry eindeutig für jede Datengruppe des Inputs bestimmt werden. b) Eine Eingabe soll keine Daten über die Abgrenzung hinweg ausgeben oder auch nicht lesen oder schreiben. c) Wenn ein funktionaler Prozess Daten vom Benutzer benötigt, dieser aber die Daten nicht senden kann, oder der Benutzer kann auf eine ankommende Nachricht nicht reagieren, so sollte dennoch ein Entry zum funktionalen Prozess für die zu erhaltenen Daten identifiziert werden. Allerdings soll in diesem Fall die Anfragenachricht des funktionalen Prozesses zum Benutzer nicht als Exit gezählt werden. d) Wenn jedoch ein funktionaler Prozess Benutzerdaten erhalten muss und der Benutzer gibt dann Daten zu dieser Anforderung zurück, dann wird ein Exit für die Anforderung und ein Entry für die Rückgabe der geforderten Daten gezählt (sieche auch Abschnitt 4.1.9).

Prinzipien des Ausganges (Exit X)
a) Ein Ausgang bzw. Exit bewegt eine einfache Datengruppe, die ein einfaches interessierendes Objekt eines funktionalen Prozesses beschreibt, als Ausgang entlang der Abgrenzung zum Benutzer. Gibt der funktionale Prozess mehr als eine Datengruppe aus, so wird ein Exit für jede eindeutige, ausgegebene Datengruppe gezählt (Siehe auch Abschnitt 4.1.7 zur Eindeutigkeit der Datenbewegungen). b) Ein Exit soll keine Daten über die Abgrenzung hinaus übertragen, oder Daten lesen oder schreiben.

Prinzipien der Eingabe (Read R)
a) Ein Einlesen bzw. ein Read ist eine einfache Datengruppe, die ein betrachtetes Objekt vom persistenten Speicher zum funktionalen Prozess, der das Read erfordert, bewegt. Empfängt der funktionale Prozess mehr als eine Datengruppe vom persistenten Speicher, so wird ein Read für jede eindeutige, empfangene Datengruppe gezählt (Siehe auch Abschnitt 4.1.7 zur Eindeutigkeit der Datenbewegungen). b) Ein Read soll keine empfangenen oder ausgegebenen Daten über die Abgrenzung hinweg zum Benutzer oder geschriebene Daten darstellen.

c) Bewegungen oder Manipulationen von Konstanten oder Variablen während eines funktionalen Prozesses, die intern in diesem Prozess sind und die nur durch den Programmierer geändert werden können oder sich als Resultat einer unmittelbaren Berechnung bzw. von gespeicherten Daten durch einen funktionalen Prozess nur aus der Implementation und nicht aus der FUR ergeben, werden nicht als Read-Datenbewegung betrachtet.

d) Eine Read-Datenbewegung enthält stets irgendeine Read-Anforderung bzw. Funktionalität (eine separate Datenbewegung wird nicht als Read-Funktionalität gezählt) (siehe auch Abschnitt 4.1.9).

Prinzipien der Ausgabe (Write W)

a) Ein Ausschreiben bzw. Write soll eine einfache Datengruppe, die ein einfaches interessierendes Objekt eines funktionalen Prozesses, der ein Write in einen persistenten Speicher einträgt, darstellen.

b) Ein Write soll keine Daten über die Abgrenzung hinweg zum bzw. vom Benutzer empfangen oder ausgeben oder auch Daten einlesen.

c) Eine Anforderung für das Löschen einer Datengruppe vom persistenten Speicher wird als eine einfache Write-Datenbewegung gezählt .

d) Bewegungen oder Manipulationen von Daten während eines funktionalen Prozesses, die nicht persistent nach einem vollständigen, funktionalen Prozess sind oder Variablen aktualisieren, die intern zum funktionalen Prozess sind bzw. unmittelbar in einer Berechnung entstehen, werden nicht als Write-Datenbewegungen betrachtet.

Prinzipien für die mit den Datenbewegungen verbundenen Manipulationen

Alle Datenmanipulationen eines funktionalen Prozesses solln mit den vier Typen der Datenbewegung (E, X, R und W) verbunden sein. Einvernehmlich wird eine Datenmanipulation eines funktionalen Prozesses stets durch eine Datenbewegung dieses Prozesses repräsentiert.

Anhang C

Übersicht zu den COSMIC-Methodenregeln

Regeln für den Messbereich
c) Der Messbereich einer Funktionalen Größenmessung (FSM) sollte unmittelbar durch den Zweck der Messung bestimmt werden. d) Der Messbereich einer einzelnen Messung sollte nicht über eine Schicht der zu messenden Software hinausgehen.

Regeln zur Schichtenbildung
e) Wenn Software eine Schichtenarchitektur in dem hier betrachteten Sinne aufweist, sollte diese Architektur auch zur Identifikation der Schichten für den Messzweck angewandt werden. f) In den Anwendungsbereichen der MIS oder Business-Applikationen wird die oberste Schicht, d. h. die Schicht, die selbst keine Unterschicht darstellt, als *Applikationsschicht* bezeichnet. (Applikations-) Software dieser Schicht basiert auf den Diensten der Software in allen anderen Schichten für eine korrekte Funktionalität. Im Falle von Echtzeit-Software bezeichnet man die oberste Schicht als *System*, beispielsweise als Prozesssteuerungssystem-Software oder als Flugsteuerungssystem-Software. g) Im Zweifelsfall sollte man die Konzepte der Kopplung und Bindung zwischen verschiedenen Interaktionschichten anwenden (siehe Anhang D). h) Man sollte nicht annehmen, dass es möglich sei, irgendeine Software ohne genauere Betrachtung des architekturellen oder strukturellen Entwurfs in Schichten gemäß dem COSMIC-Modell einteilen zu können.

Regeln zur Bestimmung der Anwendungsfunktion
c) Die Anwendungsfunktion eines zu messenden Softwarebereichs sollte durch den Zweck der Messung bestimmt werden. d) Wenn der Zweck der Messung eines Softwarebereichs auf den Aufwand für die Entwicklung oder Modifikation der Software ausgerichtet ist, sollten die Anwendungsfunktionen auch entsprechend jeweils die neue bzw. modifizierte Funktionalität darstellen.

Regeln für die Abgrenzung
d) Identifiziere die Anwendungsfunktion(en), die mit der zu messenden Software interagieren, e) Per Definition gibt es eine Abgrenzung zwischen jedem identifizierten Schichtenpaar, wo die Software einer Schicht die Anwenderfunktion der anderen Software darstellt und dann die letztere zu messen ist, f) Es gibt stets eine Abgrenzung zwischen zwei Softwarebereichen, in denen jeweils zwei Komponenten die gegenüberliegende Seiten darstellen. In diesem Fall kann jeder Softwarebereich und/oder jede Komponente eine Anwendungsfunktion der anderen darstellen.

Regeln für das Funktionsprozessniveau der Granularität
c) Funktionale Umfangsmessung sollte beim Granularitätsniveau der Funktionsprozessbeschreibung durchgeführt werden. d) Für den Fall einer Funktionsumfangsbestimmung zu einer FUR für die die Funktionsprozesse noch nicht mit allen detaillierten Datenbewegungen definiert ist, sollte erst einmal die bis dahin vorliegende Funktionalität des Garnularitätsniveaus des Funktionsprozesses gemessen werden (siehe hierzu das COSMIC-Dokument „Advanced and Related Topics“ mit den Hinweisen zu Methoden einer näherungsweisen Umfangsbestimmung auf der Grundlage einer „frühen“ Schätzung im FUR-Bestimmungsprozess).

Regeln für die Festlegung des funktionalen Prozesses
i) Ein funktionaler Prozess sollte von der weitestgehend identifizierbaren funktionalen Anforderung im zugestandenen Messbereich abgeleitet werden. j) Ein funktionaler Prozess (-Typ) bearbeitet werden, wenn ein auslösendes Ereignis bzw. Ereignistyp auftritt. k) Ein spezifisches Ereignis oder Ereignistyp sollte stets ein oder mehrere funktionale Prozess bzw. Prozesstypen auslösen, die auch parallel ausgeführt werden können. Ein spezifischer funktionaler Prozess oder Prozesstyp kann auch durch mehr als ein Ereignis bzw. Ereignistyp ausgelöst werden. l) Ein funktionaler Prozess sollte mindestens zwei Datenbewegungen einschließen, einen Eingang (Entry) und einen Ausgang (Exit) oder einer Ausgabe /Write. m) Ein funktionaler Prozess sollte sich vollkommen im Messbereich eines Software-Teils und in einer und nur einer Schicht befinden.

n) Im Kontext von Echtzeitanwendungen sollte ein funktionaler Prozess bei einem selbstinduzierten Wartezustand auch terminieren (d. h. der funktionale Prozess hat die Anforderungen als Antwort auf ein auslösendes Ereignis erfüllt und wartet bis zum Auftreten eines nächsten auslösenden Eingangs).

o) Ein funktionaler Prozess bzw. Prozesstyp sollte so bestimmt werden, dass die FUR auch gestatten, dass ein funktionaler Prozess in unterschiedlichen Teilmengen mit maximalem Input von Datenattributen und ebenso durch derartige Variationen und/oder Unterschiede der Eingabedatenwerte zu unterschiedlichen Prozesswegen innerhalb des funktionalen Prozesses führen kann.

p) Separate Ereignisse bzw. Ereignistypen und die dazugehörigen funktionalen Prozesse bzw. Prozesstypen sollten in folgende Weise unterscheiden:

- wenn Entscheidungen in unterschiedlichen Ereignissen ergeben, dass sie zeitunabhängig sind (d. h. ob heute oder später eine Datenanforderung bestätig würde, sollte diese Anforderung auch eine separate Entscheidung erfordern bzw. sollte dies unterschiedliche funktionale Prozesse erkennen lassen),

- wenn die Verantwortlichkeiten für die Aktivitäten separat vorliegen (d. h. in einem Personalsystem, wo die Verantwortlichkeit für die Wartung von Grundpersonaldaten separat zur Verantwortlichkeit für die Wartung der Gehaltsdaten ist, liegen unterschiedliche funktionale Prozesse vor; oder für ein implementiertes Software-Paket, bei der die Funktionalität dem Systemadministrator für die Wartung der Paketparameter verfügbar ist, im Gegensatz zur separaten Verfügbarkeit der Funktionalität für den eigentlichen Benutzer).

Regeln des Einganges (Entry E)
a) Die Datengruppe einer auslösenden Eingabe kann aus nur einem Datenattribut bestehen, welches einfach die Software „informiert“, dass „ein Ereignis Y aufgetreten ist“. Oftmals, insbesondere bei Business-Applikationen, besteht die Datengruppe des auslösenden Ereignisses aus vielen Datenattributen, so dass die Software zum Beispiel darüber „informiert“ wird, dass „ein Ereignis Y aufgetreten ist, das die folgenden Daten zu diesem Ereignis besitzt“. b) Zeitsignale auslösender Ereignisse sollen stets extern zur gemessenen Software betrachtet werden. Daher wird beispielsweise ein Zeitsignal, das alle 3 Sekunden auftritt, mit einem Entry als Datenbewegung einer Datengruppe eines Datenattributes verbunden. Zu beachten ist, dass es

keinen Unterschied macht, ob ein auslösendes Ereignis periodisch durch Hardware oder durch einen Softwareteil außerhalb der Abgrenzung der gemessenen Software erzeugt wird.

c) Sofern ein spezifischer funktionaler Prozess eine Zeit von einer Systemuhr erhält, wird das nicht als ein Entry betrachtet.

d) Wenn ein spezifisches Ereignis auftritt, das ein Entry einer Datengruppe bis zu *n* Datenattributen eines speziellen interessierenden Objektes erzeugt und die FUR auch ein weiteres Auftreten desselben Ereignisses erlaubt bzw. festlegt, dass ein Entry zu einer Datengruppe mit Attributwerten eine Teilmenge der *n* Attribute des betrachteten Objektes betrifft, dann wird das Entry so identifiziert, dass es die *n* Datenattribute beinhaltet.

Regeln des Ausganges (Exit X)

a) Alle erzeugten Nachrichten und Software-Ausgaben (außer Nutzerdaten), wie beispielsweise Fehlernachrichten, werden als Werte eines Attributes eines betrachteten Objektes angesehen (bezeichnet als „Fehleranzeige“). Dafür wird ein einfaches Exit identifiziert, das all diese Nachrichten repräsentiert, die in einem funktionalen Prozess durch die FUR notwendig sind.

b) Wenn ein Exit eines funktionalen Prozesses eine Datengruppe mit bis zu *n* Datenattributen eines speziellen interessierenden Objektes bewegt und die FUR funktionale Prozesse, die bei einem Exit Datengruppen mit Teilmengen aus Objektattributen erlaubt, dann wird genau ein Exit festgelegt, das alle *n* Datenattribute berücksichtigt.

Regeln für die Eindeutigkeit der Datenbewegungen und mögliche Ausnahmen

a) Sofern die Funktionalen Nutzeranforderungen es nicht anders spezifizieren, werden alle beschriebenen Datenattribute eines interessierenden Objektes und alle zugehörigen Datenmanipulationen, die in einen funktionalen Prozess Eingang finden müssen, als ein Entry identifiziert und gezählt.

(Hinweis: Ein funktionaler Prozess kann natürlich mehrere Entry-Typen behandeln, wobei jedes Entry eine Datengruppe eines unterschiedlichen interessierenden Objektes beschreibt.)

Dieselbe äquivalente Regel gilt auch für die Read-, Write- oder Exit-Datenbewegung eines gegebenen funktionalen Prozesses.

b) Mehr als eine Entry-Datenbewegung liegen vor, wenn jede bewegte Datengruppe eines interessierenden Objektes in einem gegebenen funktionalen Prozess identifiziert und gezählt wird, weil es auch diese multiplen Entries in den Funktionalen Nutzeranforderungen gibt. In gleicher Weise haben wir mehr als ein Entry für eine Bewegung derselben

Datengruppe im selben funktionalen Prozess, allerdings mit unterschiedlichen, zugeordneten Datenmanipulationen, weil es wiederum auch diese multiplen Entries in den Funktionalen Nutzeranforderungen gibt.

Derartige FUR können in einem funktionalen Prozess auftreten, während die multiplen Entries von unterschiedlichen Benutzern, die verschiedene Datengruppen zu selben interessierenden Objekt beschreiben, herrühren.

Dieselbe äquivalente Regel kann auch für die Read-, Write- oder Exit-Datenbewegung eines gegebenen funktionalen Prozesses angewandt werden.

c) Wiederholtes Auftreten von Datenbewegungstypen (d. h. jedes Auftreten bewegt dieselbe Datengruppe mit derselben Datenmanipulation) wird in einem funktionalen Prozess nicht mehrfach identifiziert und gezählt.

d) Wenn beim mehrfachen Auftreten von Datenbewegungstypen in einem gegebenen funktionalen Prozess sich diese hinsichtlich der zugeordneten Manipulationen aufgrund unterschiedlicher Werte der Datenattribute einer Datengruppe, die sich durch verschiedene Prozessverläufe ergibt, unterscheiden, wird die Datenbewegung in diesem Prozess nicht mehr als einmal identifiziert und gezählt.

Regeln für die funktionsprozessbezogenen Datenanforderungen aus der Nutzungsfunktion

a) Ein funktionaler Prozess soll eine Datengruppe über eine Entry-Datenbewegung von einem Benutzer erhalten. Wenn es nicht notwendig ist, dass der Benutzer mitteilt, welche Daten er sendet, liegt einer der folgenden vier Fälle vor:

- ein Benutzer sendet ein auslösendes Ereignis, das einen funktionalen Prozess initiiert,
- ein funktionaler Prozess, der ein auslösendes Entry empfangen hat, erwartet die Ankunft eines weiteren Entry von einem Benutzer (dieser Fall kann auch auftreten, wenn eine Benutzerperson Daten in eine Business-Applikation eingibt),
- ein gestarteter funktionaler Prozess erfordert die Funktionalität, dass im Falle vorhandener Daten diese genau jetzt zu senden sind, was dann vom Benutzer auch erfolgt,
- ein gestarteter funktionaler Prozess prüft erst die Situation des Benutzers und empfängt dann die geforderten Daten.

In den letzten beiden Fällen, die typischerweise bei Echtzeit-Polling-Software auftreten, wird per Konvention kein Exit für den Erhalt der erforderlichen Daten identifiziert. Der funktionale Prozess benötigt lediglich eine Bestätigungsnachricht zum Benutzer, wobei die Funktionalität dieser Bestätigungsnachricht Teil des Entry darstellt. Der funktionale Prozess weiß dann, welche Daten zu erwarten sind. Nur ein Entry wird in diesem Falle gezählt.

b) Wo ein funktionaler Prozess die Dienste eines Benutzers benötigt (zum Beispiel zum Erhalt von Daten) und der funktionale Prozess mitteilen muss, welche Daten zu senden sind (typischerweise wenn ein Benutzer ein anderer Software-Teil außerhalb des betrachteten, zu messenden Software-Bereichs darstellt), wird ein Exit/Entry-Paar identifiziert. Das Exit enthält die Anforderung spezifischer Daten und das Entry enthält die zurückgegebenen Daten.

Regeln für Steuerkommandos im Business-Applikationsbereich

Im Business-Applikationsbereich werden Steuerkommandos ignoriert, wenn sie keine Datenbewegungen zum interessierenden Objekt beinhalten.

Regeln für die Aggregation von Messergebnissen

a) Für jeden funktionalen Prozess wird der Funktionsumfang aller einzelnen Datenbewegungen in folgender Weise die einzelnen Funktionsumfangswerte berücksichtigend mit der Einheit CFP arithmetisch zusammengefasst.

$$Umfang(funktionaler\ Prozess_i) = \sum Umfang(Entries_i) + \sum Umfang(Exits_i)$$
$$\sum Umfang(Reads_i) + \sum Umfang(Writes_i)$$

b) Für jeden funktionalen Prozess wird der Funktionsumfang von Änderungen auf der Grundlage der Funktionalen Nutzeranforderungen den Umfang der Datenbewegungen, die im funktionalen Prozess hinzugefügt, modifiziert oder gelöscht wurden und einen geänderten Umfang in der CFP-Einheit ergeben, nach folgender Formel aggregiert.

$$Umfang(Änderung(funktionaler\ Prozess_i)) = \sum Umfang(hinzugefügter\ Datenbewegungen_i) + \sum Umfang(modifizierter\ Datenbewegungen_i) + \sum Umfang(gelöschter\ Datenbewegungen_i)$$

Weiteres über die Aggregation des Funktionsumfangs ist im Abschnitt 4.3.2 angegeben. Bezüglich der Messung geänderter Software siehe Abschnitt 4.4.

c) Der Umfang eines Software-Anteils in einem definierten Systembereich wird durch die Aggregation aller Umfänge in den funktionalen Prozessen dieses Anteils (unter Beachtung der unten angegebenen Regeln (e) und (f)) gebildet.

d) Der Umfang der Änderungen eines Software-Anteils in einem definierten Systembereich wird durch die Aggregation der Umfänge aller Änderungen in den funktionalen Prozessen dieses Anteils (unter Beachtung der unten angegebenen Regeln (e) und (f)) gebildet.

e) Der Umfang eines Software-Teils oder dessen Änderungen innerhalb von Schichten kann nur innerhalb desselben Funktionsprozessniveaus in der Granularität der FUR zusammengefasst werden.

f) Der Umfang eines Software-Teils und/oder dessen Änderungen innerhalb einer Schicht oder von unterschiedlichen Schichten wird nur zusammengefasst, wenn es bezüglich des Zwecks der Messung auch sinnvoll ist.

g) Der Umfang eines Software-Teils kann nicht durch Zusammenfassung der Umfänge seiner Komponenten (bezüglich der Dekomposition der Einheiten) gebildet werden, es sei denn, die Umfangsaufteilung bezüglich der Datenbewegungen zusammenhängender Komponenten wurde eliminiert.

h) Wenn die lokalen Erweiterungen der COSMIC-Methode angewandt werden (zum Beispiel, wenn Umfangsaspekte, die nicht im Standard enthalten sind, gemessen werden sollen), muss der jeweils gemessene Umfang separat wie in Abschnitt 5.1 beschrieben, ausgewiesen werden. Er kann nicht(!) einfach nach dem üblichen Vorgehensweise addiert und in CFP ausgedrückt werden (siehe auch Abschnitt 4.5).

Regeln für die Modifikation einer Datenbewegung

a) Wenn eine Datenbewegung aufgrund der Änderung der assoziierten Datenmanipulation zu modifizieren ist und/oder sich die Anzahl oder der Typ der Datenattribute einer Datenbewegung ändert, so muss auch ein geändertes CFP gemessen werden, (wobei die aktuelle Anzahl der Modifikationen einer Datenbewegung zu berücksichtigen ist).

b) Wenn eine Datengruppe zu modifizieren ist und diese Modifikation beeinflusst nicht die Funktionalität der Datenbewegungen für die modifizierte Datengruppe, so wird keine Datenbewegungsänderung identifiziert bzw. gezählt.

Hinweis 1: Das Auftreten einer Modifikation eines Attributwertes beispielsweise in Form einer Änderung eines speziellen Code-Elements eines Attributs in einem Code-System stellt keine Änderung des Attributtyps dar.

Hinweis 2: Eine Modifikation von in einem Ein- oder Ausgabebildschirm auftretenden Daten, die sich nicht auf den Benutzer-Objekte beziehen, wird nicht als CFP-Änderung identifiziert (siehe auch Abschnitt 3.3.4 bezüglich derartiger Datenbeispiele).

Regeln für die Kennzeichnung der COSMIC-Messung
Ein COSMIC-Messresultat sollte notiert sein in der Form „***x*** CFP (v.***y***)“, wobei • „***x***“ den numerischen Wert des Funktionsumfangs darstellt und • „v.***y***“ den Messstandard identifiziert und die Version der verwendeten COSMIC-Methode für die der Funktionsumfangswert „x“ erhalten wurde, kennzeichnet. Hinweis: Im Falle einer lokal erweiterten Messmethodenanwendung, allerdings unter Anwendung der generellen COSMIC-Methodik, sollte doch die zugrunde gelegte COSMIC-Version angegeben werden und die speziellen Erweiterungen anderweitig notiert werden (siehe auch Abschnitt 5.2).

Regeln für die lokale COSMIC-Kennzeichnungserweiterung
Ein COSMIC-Messergebnis basierend auf lokalen Methoden-Erweiterungen sollte notiert werden als „***x*** CFP (v.***y***) + ***z*** Local FP, wobei • „***x***“ den numerischen Wert des Funktionsumfangs darstellt, den man erhält durch die Aggregation aller individuellen Messergebnisse gemäß dem COSMIC-Standard Version v.y, • „v.***y***“ den Messstandard identifiziert und die Version der verwendeten COSMIC-Methode für die der Funktionsumfangswert „x“ erhalten wurde, kennzeichnet, • „***z***“ den numerischen Wert darstellt, der sich durch Aggregation der individuellen Messergebnisse aus den lokalen Erweiterungen zur COSMIC-Methode ergibt.

Regeln für den COSMIC-Messbericht
Als Ergänzung zu aktuellen Messungen, wie sie in 5.1. darzustellen sind, sollen auch die folgenden Attribute zu jeder Messung mit aufgenommen werden: a) die Identifikation der gemessenen Software-Komponente (Name, Versions-ID oder Konfigurations-ID), b) die Informationsquellen , die für die Identifikation der FUR für die Messung verwendet wurden, c) der Software-Applikationsbereich,

d) eine Beschreibung des Zwecks der Messung,

e) eine Beschreibung des Messbereichs und seine Beziehung zum Kontext möglicher anderer, damit zusammenhängender Messungen (man verwende dabei die Bereichsbeschreibung vom Abschnitt 2.2),

f) die Benutzer der Software,

g) das Granularitätsniveau der FUR und das Niveau der Dekomposition der Software,

h) den Zeitpunkt im Projekt-Lifecycle, wann die Messung gemacht wurde (speziell wenn die Messung für die Aufwandschätzung auf der Grundlage unvollständiger FUR oder auf Basis der aktuell vorhandenen Funktionalität durchgeführt wurde),

i) das Ziel bzw. mögliche Fehlerannahmen für die Messung,

j) Hinweise, ob die (reine) COSMIC-Methode oder lokale Erweiterungen verwendet wurden (siehe Abschnitt 4.5 bzw. 5.1 und 5.2),

k) Hinweise darüber, ob entwickelte oder angepasste Funktionalität gemessen wurde (entwickelte Software ist die Neuentwicklung und angepasste Software schließt auch Wiederverwendung und Formen der Anpassen mit ein),

l) ein Hinweis, ob neu hinzugefügte Funktionalität oder Erweiterungen gemessen wurden (beispielsweise Zusammenfassungen von hinzugefügter, geänderter oder gelöschter Funktionalität (siehe 4.4)),

m) eine Beschreibung der Schichtenarchitektur in der die Messung durchgeführt wurde,

n) die Anzahl der wesentlichen bzw. Haupt-Komponenten, deren Umfang für die Bestimmung des Gesamtumfangs addiert wurde,

o) für jeden Bereich innerhalb des allgemeinen Messbereichs eine Messresultatsmatrix, wie sie im Anhang A beschrieben ist,

p) den Namen der Messperson und seine COSMIC-bezogenen (zertifizierten) Qualifikationen.