VI. Die Bedeutung der Komplexität

Nahezu alle bekannten funktionsorientierten Umfangsmetriken haben ein Problem mit der Berücksichtigung von Komplexität. Die COSMIC-Methode orientiert sich generell nicht an der Komplexität und zählt alle Datenbewegungen mit dem gleichen Punktwert. Die unterschiedliche Komplexität von Anwendungsfällen nivelliert sich zwar mit zunehmender Anzahl, jedoch bleibt eine Unschärfe, wenn nur in einer Dimension gemessen wird, d.h. nur die Anzahl der Datenelemente Berücksichtigung findet.

Komplexität gar nicht zu berücksichtigen erscheint fragwürdig. Das Dilemma ist jedoch, dass es unterschiedliche Arten der Komplexität gibt, deren Eignung für eine Umfangsmetrik nicht ohne Weiteres erkennbar ist.

Die Komplexität einer Implementierung

Es gibt eine Vielzahl von Metriken, die Aspekte der Komplexität einer konkreten Implementierung bewerten. Eine der bekanntesten Komplexitätsmetriken wurde von Thomas J. McCabe im Jahre 1976 entwickelt. Sie misst die zyklomatische Komplexität eines Programms, d.h. die Anzahl der konditionalen Zweige in seinem Kontrollflussgraphen [McCabe 1976]. Dies entspricht der Anzahl der binären Entscheidungen plus 1, sodass man die zyklomatische Komplexität auch durch das Zählen von Anweisungen, die neue Zweige im Kontrollflussgraphen einführen, ermitteln kann. Dies sind Anweisungen wie „If", Iterationskonstrukte, logische Operatoren, usw.

Ein einfaches Beispiel dafür, wie unterschiedlich die zyklomatische Komplexität verschiedener Implementierungen derselben Anwendungsfälle sein kann, zeigen die Abbildungen 12 und 13. Beide Programme sind eine Implementierung der Anforderung „Berechne das Produkt zweier ganzer Zahlen a und b". Die zyklomatische Komplexität des Programms in Abbildung 12 beträgt 1, die des Programms

in Abbildung 13 beträgt 6 (Basiswert 1 plus 3 If-Anweisungen plus 2 Schleifen). So absurd der in Abbildung 13 dargestellte Lösungsansatz auch anmuten mag – die Praxis zeigt, dass die Komplexität der Implementierung einer funktionalen Anforderung nicht immer den niedrigsten Idealwert erreicht, denn sie ist abhängig beispielsweise von den Kenntnissen der Programmiersprache wie auch der Erfahrung des Entwicklers.

```
c = a * b
```

Abbildung 12: Programmbeispiel 1

```
If b = 0 Then
      c = 0
ElseIf b > 0 Then
      c = 0
      For i = 1 To b
            c = c + a
      Next i
ElseIf b < 0 Then
      b = Abs(b)
      c = 0
      For i = 1 To b
            c = c + a
      Next i
      c = -1 * c
End If
```

Abbildung 13: Programmbeispiel 2

Einen anderen Aspekt der Komplexität bewertet die 1977 von Maurice Halstead entwickelte Metrik zur Messung der lexikalischen/textuellen Komplexität von Programmcode auf Basis des Vokabulars, d.h. der Menge verwendeter Operanden und Operatoren [Halstead 1977]. Ein weiteres Beispiel ist die Metrik „Response for a Class" (RfC) , welche die Komplexität einer Klasse durch das Zählen ihrer Methoden und der Kopplungen mit anderen Klassen errechnet [ITWissen 2014].

Das Problem dieser Metriken ist, dass sie die Komplexität einer konkreten Implementierung messen. Wie das Beispiel der Abbildungen 12 und 13 zeigt, kann es jedoch beliebig viele Implementierungen derselben funktionalen Anforderung geben, die sich durch sehr unterschiedliche Komplexitätswerte voneinander unterscheiden.

Metriken zur Bewertung der Komplexität einer konkreten Implementierung sind wichtig, wenn es um Aspekte wie Wartbarkeit oder Testbarkeit genau einer Implementierung geht. Ein anderes Entwicklungsteam hätte die gleichen funktionalen Anforderungen jedoch möglicherweise völlig anders umgesetzt – mit der Konsequenz einer besseren oder möglicherweise auch schlechteren Wartbarkeit. Für Produktivitätsmessungen ist diese Ahängigkeit von individuellen Programmierstilen oder technologischen Aspekten nicht wünschenswert, da sie die Validität der Messergebnisse einschränkt: Ein Projekt, das die gleichen funktionalen Anforderungen durch eine größere Menge umständlichen, unangemessen komplexen Programmcodes mit geringer Modularisierung und Wiederverwendung implementiert, würde sich durch eine höhere Produktivität auszeichnen. Es hätte mehr Output bei gleichem Aufwand produziert. Dies führt aber den Produktivitätsbegriff, wie er in Kapitel III definiert wurde, ad absurdum und bringt die Gefahr mit sich, dass Maßnahmen zur Verbesserung der Produktivität nur schlechte Codequalität fördern.

Interaktionskomplexität

Funktionsorientierte Metriken messen den Umfang von Interaktionen zwischen dem betrachteten System und den Akteuren seiner Anwendungsfälle (seinen funktionalen Benutzern). Sie sollten sich daher auch an der Komplexität dieser Interaktionen bzw. der funktionalen Prozesse, die durch diese Interaktionen angestoßen werden, und nicht an der Komplexität einer Implementierung orientieren.

Bei der Function Point-Analyse geht die Komplexität in Form unterschiedlicher Punktwerte für die gezählten Elementarprozesse und die Systemgrenzen überschreitende Datenstrukturen ein. Wie zuvor beschrieben, wird dabei der Punktwert eines Zählobjekts von der Anzahl involvierter Datenelemente bzw. Strukturen abgeleitet, was zweifellos eine Metrik für die Komplexität der Interaktionen ist. Die Bestimmung dieser Punktwerte durch dreistufige Intervallskalen kann jedoch zu Unschärfen und Begrenzungen führen, deren praktische Auswirkungen am Ende von Kapitel IV aufgezeigt wurden.

Ein anderer Ansatz ist es, die Komplexität von Anwendungsfällen aus der Richtung der Datenbewegungen abzuleiten, wie im Fall der Data Interaction Point-Methode. Die Eingabe eines Datenelements in einen Dialog hat aufgrund der notwendigen Validierungen und der Persistenz eine höhere Komplexität als die reine Anzeige bzw. Ausgabe und wird daher mit einem höheren Punktwert gezählt. Ebenso ist das Schreiben eines Datenelements in die Datenbank aufgrund der Validierungen, Konsistenz- und Integritätsprüfungen usw. mit einer höheren Komplexität verbunden als ein reiner Lesezugriff. Der Vorteil solcher Unterscheidungen liegt in ihrer Eindeutigkeit, d.h. Eingabe- und Ausgabeelemente lassen sich selbst bei einer automatisierten Zählung meist leicht voneinander unterscheiden und präzise ermitteln. Hinzu kommt, dass die Anzahl der Eingabe- und die Anzahl der Ausgabeelemente den Umfangswert direkt und nicht wie im Fall der

Function Point-Analyse über Intervallskalen beeinflussen. Schwieriger wird diese Unterscheidung bei Schnittstellen, d.h. bei Datenbewegungen zwischen dem zu messenden System und anderen Systemen. Die Frage, ob die Aufbereitung von Daten zur Ausgabe an ein anderes System komplexer ist oder die Validierung und Verarbeitung eingehender Daten, kann nicht allgemein beantwortet werden. Im günstigsten Fall lassen sich Komplexitätsklassen in Abhängigkeit von Schnittstellentypen festlegen, deren Punktwerte mit der Komplexität der vor- bzw. nachgelagerten Funktionalität korrelieren. Falsch wäre es, die Komplexität zu schätzen, da verschiedene Schätzer meist zu unterschiedlichen Schätzwerten kommen und dies – nach der Summierung vieler geschätzter Punktwerte – die Objektivität einer Messmethode einschränken würde.

Algorithmische Komplexität

Überall dort, wo Interaktionen eines Systems mit Menschen oder anderen Systemen im Vordergrund stehen, ist eine Umfangsmessung auf Basis der Anwendungsfälle sinnvoll, wie sie im Standard ISO/IEC 14143 beschrieben wird. Dabei orientiert man sich ausschließlich an den Datenbewegungen zwischen dem zu messenden System und den Akteuren der Anwendungsfälle sowie der Repräsentanz dieser Daten in der Datenbank.

Auf Systeme, bei denen diese Interaktionen nicht im Vordergrund stehen, die beispielsweise in der Hauptsache komplexe Algorithmen ausführen, lassen sich funktionsorientierte Umfangsmetriken, wie sie in diesem Buch beschrieben wurden, nur sehr eingeschränkt oder gar nicht anwenden. Dies mag am Beispiel eines Routenplanungsprogramms deutlich werden, das als Input eine Start- und mehrere Zieladressen erhält und als Output eine Liste von Streckenabschnitten liefert. Bei Verwendung einer funktionsorientierten Umfangsmetrik nach ISO/IEC 14143 würde man einen eher geringen Umfang und bei der Nachbetrachtung einer ab-

geschlossenen Implementierung wahrscheinlich eine sehr geringe Produktivität feststellen. Die hochkomplexen Planungsalgorithmen, deren Implementierung ohne jeden Zweifel einen hohen Aufwand verursacht, wären dabei völlig unberücksichtigt geblieben.

Das Beispiel eines Routenplanungsprogramms zeigt deutlich, dass eine Messung des funktionalen Umfangs, wie sie im Standard ISO/IEC 14143 beschrieben wird, ein – hinsichtlich Aufwands- bzw. Produktivitätsbetrachtungen – unvollständiges Bild des Output aus einem Softwareentwicklungsprozess ergibt, falls der Schwerpunkt des Systems in der Ausführung komplexer Algorithmen liegt. Die Antwort, wie Umfang und Produktivität solcher Systeme praxistauglich und im Sinne der Anforderungen aus Kapitel IV objektiv, zuverlässig und valide gemessen werden können, muss zunächst noch offen gelassen werden, da dieses Thema noch weiterer Untersuchungen bedarf. In der Praxis hat sich eine Aufteilung des zu messenden Systems hinsichtlich Komponenten mit hoher und niedriger algorithmischer Komplexität bewährt. Während so der Umfang des einen Teilsystems mit einer funktionsorientierten Metrik gemessen und auf dieser Basis der Aufwand ermittelt werden kann, sind Systemteile mit hoher algorithmischer Komplexität einer Expertenschätzung zu unterziehen.