Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

AW-1 Ausarbeitung

Daniel Wojtucki

Softwaremessung und -metriken

Inhaltsverzeichnis

1 Einleitung

Lord Kelvin stellte 1883 fest:

> *„When you can measure what you are speaking about and express it in numbers you know something about it, but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind.“* [vgl. 8, S. 33]

Diese Aussage trifft ebenfalls auf die Softwareentwicklungsprojekte zu. Damit gewünschte Projektziele erreicht werden können, wird ein Maß benötigt welches es zu erfüllen gilt. In den ingenieurs- und betriebswirtschaftlichen Bereichen hat sich die Erhebung von Kennzahlen bereits fest etabliert. Die theoretischen Grundlagen für Softwaremessungen wurden bereits vor über einem halben Jahrhundert gelegt. Teilweise werden und wurden die sogenannten *Metriken* (Softwaremessungen) bereits erfolgreich in IT-Projekten eingesetzt. Allerdings setzten viele Unternehmen dieses Hilfsmittel für die Qualitätsbewertung nicht ein, was meist auf einen mangelhaften Softwareprozess oder fehlende Werkzeuge zurückzuführen ist.

Diese Arbeit soll einen ersten Einblick in die Verwendung von Metriken geben. Dazu werden in Kapitel 2 einleitend verschiedene Definitionen für *Softwarequalität* vorgestellt und erläutert was unter dem Begriff *Qualitätsmanagement* zu verstehen ist. Dabei werden *Reviews* als ein Werkzeug der Qualitätskontrollen eingeführt und erläutert wie Metriken diese Maßnahmen unterstützen können. In Kapitel 3 wird definiert was Metriken sind, wofür und wie sie in einem Softwareprozess eingesetzt werden können. Dafür wird neben den sogenannten *Gütekriterien* (vgl. Abschnitt 3.2) ein möglicher Messprozess skizziert. Anhand dessen können die einzelnen Tätigkeiten nachvollzogen werden, die für eine erfolgreiche Messung durchzuführen sind. Um einige konkrete Metriken kennenzulernen, werden in Kapitel 4 drei der bekanntesten klassischen Metriken vorgestellt und einige Probleme geschildert, die bei der Verwendung dieser Metriken auftreten können. Schließlich werden in Kapitel 5 die wichtigsten Faktoren beschrieben, die es bei der Verwendung von Metriken zu beachten gilt.

Es ist nicht Ziel dieser Arbeit einen konkreten Umgang mit Metriken in einem realen Projektumfeld zu vermitteln. Vielmehr sollen die Rahmenbedingungen bei einem Umgang mit Metriken aufgezeigt und die Vor- und Nachteile beleuchtet werden.

2 Softwarequalität und Qualitätsmanagement

Bevor genauer auf die Metrik, als eine der qualitätssichernden Maßnahmen, eingegangen werden kann, ist es sinnvoll, sich mit dem Begriff der *Softwarequalität* auseinanderzusetzen.

2.1 Definitionen Softwarequalität

Dieser Abschnitt stellt eine Auswahl von Definitionen vor, welche beschreiben, wie der Begriff Qualität bei einem Softwareprodukt zu verstehen ist.

E. Wallmüller fasst dazu aus einer empirischen Studie von D. A. Garvin (1984) folgende Ansätze zusammen, wie Softwarequalität unter Betrachtung verschiedener Ansätze zu verstehen ist [vgl. 8, S.8-12]:

- „Der *transzendente* Ansatz“

  Qualität wird durch die direkte Verwendung der Software *erlebt*, ähnlich wie der Begriff der Schönheit.

- „Der *produktbezogene* Ansatz“

  Hierbei werden quantitativ gleichwertige Softwareprodukte miteinander verglichen und in eine Rangfolge gebracht. Dabei richtet sich die höchste Qualität nach dem höchsten Rang.

- „Der *anwendungsbezogene* Ansatz“

  Das Bedürfniss des Softwareanwenders gibt nach diesem Ansatz das Maß der Qualität vor. Passt die Software sehr gut zu den Bedürfnissen des Anwenders, ist sie als qualitativ hochwertig einzustufen.

- „Der *prozessbezogene* Ansatz“

  Dieser Ansatz spiegelt den heute dominierenden wirtschaftlichen Ansatz wider. Demnach richtet sich die Qualität eines Produkts nach seinem Produktionsprozess. Dieser wird ständig kontrolliert, verbessert und nach Möglichkeit automatisiert, was zu einer gleichbleibenden bzw. steigenden Qualität führt.

  Dabei wird idealerweise davon ausgegangen, dass der Produktionsprozess beim ersten Durchlauf richtig ausgeführt wird.

Die inzwischen obsolete DIN EN ISO 8402:1995-08 fasste den Qualitätsbegriff wiefolgt zusammen [vgl. 11]:

> *„Qualität ist die Gesamtheit von Merkmalen einer Einheit bezüglich ihrer Eignung, festgelegte und vorausgesetzte Erfordernisse zu erfüllen.“*

Abgelöst wurde diese Norm durch die DIN EN ISO 9000:2008, welche die gültige Norm des Qualitätsmanagements ist und damit auch den Begriff der Softwarequalität definiert. Neben der IEEE-Norm für Softwarequalität (IEEE Std 729-1983) gibt es eine Vielzahl weiterer Definitionen.

Eine einheitliche Definition von Qualität, insbesondere von Softwarequalität, existiert derzeit also nicht. Zollondz beschreibt Qualität sogar als *komplexes Phänomen* [vgl. 13, S.159], was andeutet, dass Qualität nicht als fest definierte Größe verstanden werden kann. Vielmehr muss Qualität relativ zu den gegebenen Erfordernissen an das Produkt gesehen werden. Eine Qualitätsbewertung muss aufgrund der den Erfordernissen angepassten Qualitätsvorgaben („Soll-Wert“) und den tatsächlich umgesetzten Qualitätsmerkmalen („Ist-Wert“) durchgeführt werden [vgl. 8].

2.2 Qualitätsmanagement

Ähnlich der Definition der Qualität verhält es sich mit der Definition der Qualitätssicherung. In der Literatur gibt es eine Vielzahl von Ausführungen, was unter Qualitätssicherung zu verstehen ist. Sommerville sieht Qualitätssicherung als „Prozess des Definierens, wie sich Softwarequalität erreichen lässt“ [vgl. 6, S. 693]. Das Prozessergebnis sind Standards und Verfahren, die eine gleichbleibende Qualität des Softwareprodukts unterstützen sollen. In der ISO 900 Norm, speziell der ISO 9003, werden bewährte Standards für die Softwareentwicklung vorgestellt.

Qualitätskontrollen

Damit sichergestellt werden kann, dass die Verfahren und Standards der Qualitätssicherung eingehalten werden, wird die Qualitätskontrolle durchgeführt. Dabei werden die Ergebnisse des Softwareentwicklungsprozesses im Qualitätskontrollprozess auf die festgelegten Standards überprüft. Dabei stellen Qualitäts-Reviews das Hauptwerkzeug für derartige Qualitätskontrollen dar.

Werkzeug: Reviews

Reviews sind eine statische, manuelle Überprüfung von Arbeitsergebnissen (bspw. Code, Anfoderungsdokumente, Testpläne oder Architekturentwürfe) während der Softwareentwicklung. Ein sogenanntes Review-Team führt zu einem definierten Zeitpunkt und mit einer festgelegten Dauer die Überprüfung auf ein bestimmtes Dokument durch. Dieses Team kann diese Aufgabe aber auch an andere Projektteilnehmer (bspw. andere Entwickler) delegieren. Je nach Review-Art werden die gefundenen Fehler und Widersprüche mit dem Autor des Dokuments in einem länger angelegten Gespräch diskutiert oder schriftlich kommentiert [vgl. 6, S. 701f]. Sofern eine sinnvolle Review-Strategie existiert, erhöhen Reviews signifikant die Qualität der inspizierten Dokumente. Frühauf, Ludewig und Sandmayr [vgl. 2, S. 100f] schätzen, dass während eines Reviews 60% bis 70% der Fehler in einem Dokument gefunden werden. Dies kann für ein Softwareprojekt eine Einsparung zwischen 14% bis 25% bedeuten.

Probleme mit Reviews

Trotz der genannten Vorteile benötigen Reviews Aufwand und Zeit. Die Nutzung dieser beiden Projektressourcen kann letztendlich zu Verzögerungen im Softwareprozess führen und damit zu einer verzögerten Auslieferung der Software selbst. In der Praxis können Reviews also nur schwer in ganzer Breite eingesetzt werden [vgl. 6, S. 702]. Es ist nötig, mit Hilfe eines geeigneten Analysewerkzeugs, schon vor der Durchführung eines Reviews mögliche Schwachstellen in den Projektergebnissen zu finden. Solche Analysewerkzeuge stellen vor allem *Softwaremetriken* dar, die in den folgenden Kapiteln eingehend erläutert werden. Allerdings dürfen Softwaremetriken keinesfalls die Durchführung von Reviews ersetzten. Metriken können diese nur unterstützen, beziehungsweise den erforderlichen Aufwand verringern.

3 Grundlagen der Softwaremetrik

Im vorherigen Abschnitt 2.2 wurde die Softwaremetrik als unterstützendes Analysewerkzeug der Reviews vorgestellt. Tatsächlich lassen sich Metriken allerdings vielseitiger einsetzen, zum Beispiel:

- Messen und Vergleichen der Qualität eines Objekts
- Überprüfung eines Prozesses bzw. einer Prozessverbesserung
- Abschätzen von Aufwänden (bspw. der benötigten Personentage für eine bestimmte Aufgabe)
- Bestimmung des derzeitigen Teststandes während einer Testphase

3.1 Definition

Man versteht unter einer Softwaremetrik also „ ...jede Art von Messung, die sich auf ein Softwaresystem, einen Prozess oder die dazugehörige Dokumentation bezieht“ [vgl. 6, S. 703]. Das Wort *Messung*, ein aus den Ingenieurwissenschaften geprägter Begriff, bedeutet auch im Zusammenhang mit Softwaremetriken das Ausführen einer geplanten Tätigkeit, um quantitativ und systematisch bestimmte Merkmale und Eigenschaften von Softwaresystemen zu erfassen [vgl. 10]. Der IEEE Standard 1061-1992 fasst dies wiefolgt zusammen:

> *„Eine Softwaremetrik ist eine Funktion, die eine Softwareeinheit in einem Zahlenwert abbildet. Dieser berechnete Wert ist interpretierbar als der Erfüllungsgrad einer Qualitätseigenschaft der Software-Einheit.“* [vgl. 12]

Für die Softwareeinheit führen Simon, Seng und Mohaupt den Begriff der *Entität* ein. Die Autoren definieren eine Entität als Sammelbegriff für verschiedene Objekte, die während der Softwareerstellung identifizierbar und messbar sind. Dies schließt sowohl *materielle* Objekte (z.B. Personen), *idelle* Objekte (z.B. Programme) als auch *konzeptionelle* Objekte (z.B. Prozesse) mit ein. Daraus ergibt sich eine mögliche Kategorisierung von Metriken in drei Gruppen [vgl. 5, S. 113f]:

Prozessmetriken – für diejenigen Aktivitäten, die sich auf die Bearbeitung von Software beziehen

Ressourcenmetriken – für diejenigen Mittel, die für einzelne Prozesse benötigt werden

Produktmetriken – für diejenigen Artefakte, die das Ergebnis eines Prozesses sind

3.2 Voraussetzung für eine korrekte Messung

Neben der Methodik zur Auswahl einer Metrik, um eine Antwort auf eine bestimmte Frage zu erhalten (vgl. Abschnitt 3.3), gibt es sogenannte *Gütekriterien* für Metriken. Die Gütekriterien beschreiben notwendige Eigenschaften einer Metrik, damit diese auch

nutzbringend eingesetzt werden kann. Hoffmann definiert die folgenden sechs Kriterien [vgl. 4, S. 248f]:

Objektivität – Die Messung muss weitgehend frei von subjektiven Einflüssen sein. Da eine Metrik in der Regel über eine mathematische Funktion definiert wird, ist dies meist gewährleistet.

Robustheit – Eine Metrik gilt als Robust, wenn eine wiederholte Messung auf dieselbe Kenngröße einer Entität das gleiche Ergebnis liefert. In Kapitel 4 wird erklärt, warum bspw. die *LOC-Metrik* dieses Kriterium nur bedingt erfüllt.

Vergleichbarkeit – Verschiedene Messergebnisse derselben Kenngröße müssen miteinander vergleichbar sein, d.h. die gleiche Einheit aufweisen. Nur so ist es möglich, verschiedene Entitäten untereinander zu vergleichen und Metriken auch als Steuermetriken einzusetzten.

Ökonomie – Der Einsatz einer Metrik muss im Vergleich mit dessen Nutzen kostenökonomisch sinnvoll sein. In der Regel lassen sich Metriken automatisiert einsetzen. Ist für die Anwendung einer Metrik ein hoher Personaleinsatz notwendig, muss geprüft werden, ob der Einsatz dieser Metrik angemessen ist.

Korrelation – Die Korrelation einer Metrik beschreibt die Aussagekraft zwischen dem Messergebnis und der überwachten Kenngröße. Für einige Eigenschaften von Software, bspw. der Transparents, ist eine korrelierende Metrik schwer zu formulieren.

Verwertbarkeit – Dieses Kriterium gehört zu den wichtigsten Vorraussetzungen bei dem Einsatz von Metriken. Eine Metrik ist nur dann zu nutzen, wenn das Messergebnis überhaupt verwertet werden kann. Metriken dürfen nicht zum Selbstzweck eingesetzt werden, sondern sollen immer Antworten auf bestimmte Fragestellungen geben und damit das Handeln in einem Softwareprozess beeinflussen.

3.3 Der Messprozess

Damit die Anwendung von Metriken tatsächlich zu einer Qualitätssteigerung des Softwareprodukts beitragen kann oder den Softwareprozess gewinnbringend unterstützt, skizziert Sommerville einen möglichen Prozess für die Verwendung von Metriken [vgl. 6, S. 705f]. In Abbildung 1 ist der Prozess mit seinen Schritten schematisch dargestellt. Die einzelnen Prozessschritte werden folgend näher erläutert:

1. *Durchzuführende Messung wählen:* Dieser Schritt hat wie das Gütekriterium *Verwertbarkeit* (vgl. Abschnitt 3.2) besondere Bedeutung für den Umgang mit Metriken. In dieser Phase muss bestimmt werden, welche Metriken benötigt werden, um eine bestimmte Frage zu beantworten. Basili entwickelte für diese Aufgabe ein

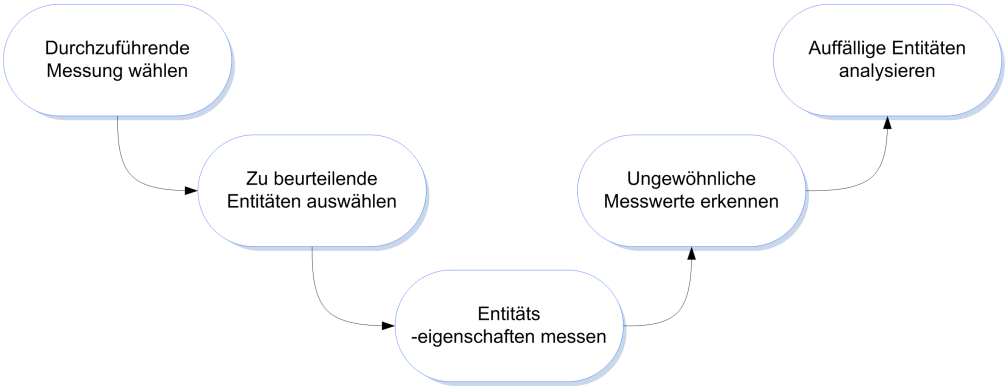

Abbildung 1: Prozess für die Anwendung einer Metrik [nach 6].

spezielles Vorgehen, das GQM-Muster (*Goal-Question-Metric*) [vgl. 1]. Dabei wird ein Ziel definiert, welches durch die Messung erreicht werden soll. *„Effektivität der Coderichtlinien bestimmen“*, könnte bspw. ein formuliertes Ziel sein. Für die Beantwortung eines Ziels müssen verschiedene Fragen formuliert werden. Jede Frage wird dann mit einer oder mehreren Metriken beantwortet.

Es ist wichtig, dass auch nur die Messungen durchgeführt werden, die für die Erfüllung eines Ziels benötigt werden. Werden zuviele Metriken erhoben und ist die Verwertbarkeit fragwürdig, so schadet dies zurecht der Akzeptanz der Projektteilnehmer gegenüber den Metriken.

2. *Zu beurteilende Entitäten auswählen:* Nicht immer ist es nötig, alle Entitäten einer Gruppe der Messung zu unterziehen. Oftmals können einige Entitäten repräsentative Aussagen über alle anderen Gruppenentitäten treffen. Handelt es sich bei den Entitäten zum Beispiel um alle Komponenten eines Softwaresystems, ist es zum Beispiel sinnvoll, nur die Kernkomponenten zu messen, weil diese am häufigsten genutzt werden.

3. *Entitätseigenschaften messen:* In diesem Schritt werden alle benötigten Metriken auf die ausgewählten Entitäten angewandt. Die meisten Metriken lassen sich mit Hilfe von Werkzeugen automatisiert durchführen. Solche Werkzeuge können speziell erstellt werden oder sind bereits in dem verwendeten CASE-Werkzeug enthalten. Die ermittelten Messwerte können bswp. in einer Datenbank gespeichert und somit projekt- und organisationsübergreifend genutzt werden.

4. *Ungewöhnliche Messwerte erkennen:* Die ermittelten Messergebnisse müssen untereinander oder aufgrund von Schwellwerten nach auffällig hohen oder niedrigen Ergebnissen untersucht werden. Es können sowohl die aktuellen Ergebnisse als auch historische miteinander verglichen werden. Werden solche Werte gefunden, ist es möglich, dass es Probleme mit der betreffenden Entität gibt.

5. *Auffällige Entitäten analysieren:* Nur weil es in einer Entität zu einer Anomalie in den Messergebnissen gekommen ist, bedeutet das nicht automatisch, dass die Qualität der Entität vermindert ist. Es muss also der Grund für das auffällige Messergebnis ermittelt und möglicherweise ein manuelles Review durchgeführt werden.

Sommerville sieht den Einsatz von Metriken vor allem als Analysewerkzeug, um Qualitätsprobleme aufzudecken, wie dies bereits in Abschnitt 2.2 erläutert wurde. Das spiegelt sich besonders in den letzten Prozessschritten wider. Dort, wo der Messprozess aufhört, schließen die Qualitätsreviews an. Somit können die aufwändigen Reviews gezielt mit den Entitäten durchgeführt werden, die möglicherweise ein Qualitätsdefizit aufweisen.

4 Beispiele konkreter Metriken

In diesem Kapitel wird eine Auswahl an konkreten Metriken vorgestellt. Bei den folgenden drei Metriken handelt es sich um die wichtigsten Metriken der Vergangenheit, welche aus den 50er bis 70er Jahren stammen - also den Anfängen der Softwareentwicklung. Sie sollen einen ersten Einblick in konkrete Metriken geben und mögliche Probleme bei ihrer Verwendung erläutern.

4.1 LOC und NCSS

Die *LOC-Metrik* (*Lines Of Code*) ist eine der simpelsten Metriken, die eingesetzt werden. Dabei werden die einzelnen Codezeilen einer Entität (bswp. einer Code-Datei) addiert. Somit lässt sich der Umfang einer Software messen, was allerdings keine Aussage über die Komplexität bzw. die Qualität der Software zulässt. Bei der eigentlichen LOC-Metrik werden keinerlei Unterscheidungen der einzelnen Zeilen gemacht. Dadurch gibt es eine Vielzahl an Variationen der LOC-Metrik, wovon einige genannt werden sollen [vgl. 4, S.249ff]:

- Zählen aller Zeilen ohne Kommentarzeilen (*NCSS-Metrik, Non Commented Source Statements*)
- Zählen aller Zeilen, die nur ausführbaren Code enthalten
- Zählen aller Zeilen, die nur ausführbaren Code und Variablendeklaration enthalten
- Zählen aller Programmsymbole (*Token*)
- Zählen aller Anweisungsparamter (bswp. das Semikolon in Java)

Die Aussagekraft der Messergebnisse mit der LOC-Metrik ist gerade in Hinsicht auf *Robustheit* eher gering. Bereits das Umformatieren des Sourcecodes hat erhebliche Auswirkung auf die Messergebnisse bei wiederholter Messung derselben Entität, obwohl sich die Funktionalität der Entität nicht verändert hat. Ein ähnliches Problem kann auch bei einem Vergleich zweier unterschiedlicher Entitäten auftreten. Zum Beispiel können

zwei verschiedene Entwickler ganz unterschiedliche Gewohnheiten haben, Sourcecode zu formatieren.

Ein weiteres Problem besteht in der *Vergleichbarkeit* von Kenngrößen, wenn der zu messende Sourcecode in verschiedenen Programmiersprachen geschrieben ist. Je nach Sprache kann die Länge des Codes variieren. Dazu wurden sogenannte *Sprachfaktoren* empirisch ermittelt, womit die ermittelte Kenngröße multipliziert werden kann. Somit sollen dann auch Entitäten in verschiedenen Programmiersprachen vergleichbar sein.

Trotz der genannten Nachteile werden laut Hoffmann [vgl. 4, S. 251] die LOC- und NCSS-Metriken verbreitet in Firmen eingesetzt, um den Wartungsaufwand für ein bestimmtes Volumen an Sourcecode abzuschätzen. Zudem soll die LOC-Metrik häufig als Basisparamter von anderen Metriken verwendet werden.

4.2 Halstead-Metriken

Mit den Halstead-Metriken lässt sich unter anderem auch das Volumen einer Code-Entität bestimmen. Die 1977 von Maurice Howard Halstead begründeten Metriken berechnen dabei allerdings das Volumen bzw. die Länge einer Code-Entität (folgend *Programm* genannt) unmittelbar aus dessen lexikalischer Struktur. Dazu werden die meisten Bestandteile einer Programmiersprache in zwei Klassen aufgeteilt, die *Operatoren* und die *Operanden*. Zu den Operatoren zählen alle Schlüsselwörter, vordefinierte Operatoren und Präpozessoranweisungen. Unter die Operanden fallen die Bezeichner von Funktionen und Variablen, sowie numerische und textuelle Konstanten. Daraus ergeben sich vier Basisparamter, welche die Grundlage für die meisten Halstead-Metriken bilden [vgl. 4, S. 251-259]:

$$\eta_1 = \text{Anzahl unterschiedlicher Operatoren} \tag{1}$$

$$\eta_2 = \text{Anzahl unterschiedlicher Operanden} \tag{2}$$

$$N_1 = \text{Gesamtzahl der Vorkommen aller Operatoren} \tag{3}$$

$$N_2 = \text{Gesamtzahl der Vorkommen aller Operanden} \tag{4}$$

Weitere Basisgrößen lassen sich direkt aus den vier Basisparamtern bestimmen. Zum einen stellt die Summe aller unterschiedlichen Operatoren und Operanden die Vokabulargröße des Programms dar (vgl. Funktion 5).

$$\eta = \eta_1 + \eta_2 \tag{5}$$

Des Weiteren kann die Länge N eines Programms aus der Summe der Vorkommen aller Operanden und Operatoren gebildet werden:

$$N = N_1 + N_2 \tag{6}$$

In der nachfolgenden Auflistung sind alle weiteren Halstead-Metriken zusammengefasst, allerdings wird an dieser Stelle auf eine detaillierte Beschreibung der einzelnen Metriken verzichtet:

Minimalvokabular $\eta^* = \eta_2^* + 2$

Volumen	$V = N \cdot log_2 \eta$
Minimalvolumen	$V^* = \eta^* \cdot log_2 \eta^*$
Level	$L = \frac{V^*}{V}$
Schwierigkeit	$D = \frac{1}{L}$
Aufwand	$E = V \cdot D$

4.3 McCabe-Metrik

Mit der McCabe-Metrik lässt sich die Komplexität einer Software bzw. eines Softwaremoduls bestimmen. 1976 führte Thomas J. McCabe die *zyklomatische Komplexität* ein [vgl. 9], welche auf die *zyklomatische Zahl* aus der Graphentheorie zurückzuführen ist. Mit Hilfe der zyklomatischen Zahl $V(Z)$ lässt sich ein Rückschluss auf die Elementarpfade eines stark zusammenhängenden Graphen Z ziehen [vgl. 4, S. 259]:

$$V(Z) = |E| - |N| + 1 \tag{7}$$

Dabei sind $|E|$ und $|N|$ die Anzahl der Kanten und Knoten von Z. Da sich die Gleichung 7 allerdings nur auf stark zusammenhängende Graphen bezieht, kann diese Gleichung nicht auf einen Kontrollflussgraphen angewandt werden. Zu diesem Zweck werden der Start- und Endpunkt des Kontrollflussgraphen künstlich verbunden. Mit dieser zusätzlichen Kante sieht die zyklomatische Zahl für einen Kontrollflussgraphen G folgendermaßen aus:

$$V(G) = |E| - |N| + 2 \tag{8}$$

In der Regel wird die Komplexitätsmessung auf einzelnen Programmfunktionen durchgeführt, wie in Abbildung 2 beispielhaft zu sehen ist. Somit können während des gesamten Softwareentwicklungsprozesses kritische Programmkomponenten entdeckt und gegebenfalls angepasst werden. McCabe hält eine obere Komplexitätsgrenze von 10 für sinnvoll, sagt aber auch, dass dieser Wert nicht absolut betrachtet werden kann und je nach Umständen angepasst werden muss [vgl. 4, S. 261].

Mehrere empirische Studien, darunter eine von Robert B. Grady von Hewlett-Packard [vgl. 3], bestätigen eine gute Korrelation zwischen der zyklomatischen Komplexität und der Wartbarkeit bzw. Fehleranfälligkeit von Programmkomponenten. Kritiker halten das Komplexitätsmaß für Menschen nicht intuitiv nachvollziehbar. Es gibt Beispiele, in denen Programmstrukturen mit einer hohen Komplexität für Menschen trotzdem leicht nachvollziehbar und damit auch wartbar sind (bspw. `switch`-Konstrukte) [vgl. 9].

Neben der Bestimmung der Code-Komplexität wird die McCabe-Überdeckung auch für die Testfallgenerierung eingesetzt. Somit lassen sich alle nötigen Testfälle ermitteln, um eine komplette Testüberdeckung sicherzustellen.

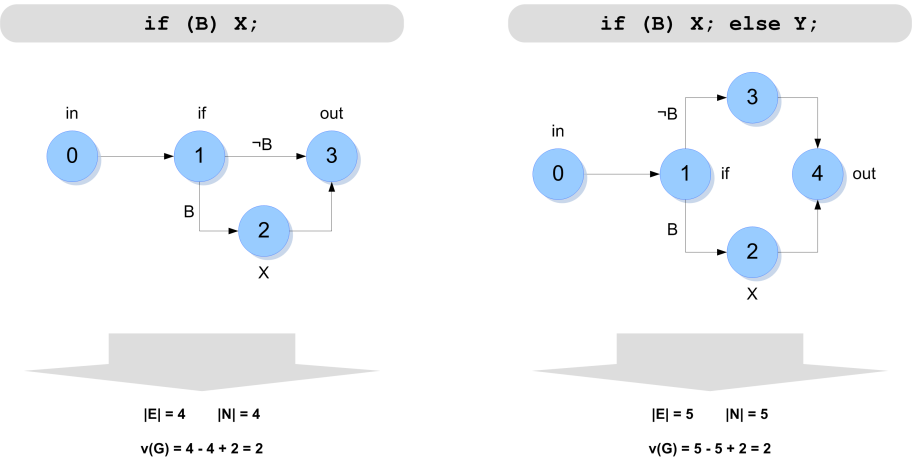

Abbildung 2: Beispiel zweier Kontrollflussgraphen mit gleicher Komplexität [nach 4].

5 Fazit

Metriken können als Werkzeug bei der Qualitätskontrolle eingesetzt werden um eventuelle Qualitätsdefizite aufzudecken. Dabei müssen verschiedenen Faktoren *vor* dem Einsatz von Metriken geklärt werden.

Einerseits muss allen Beteiligten am Softwareprozess klar sein, was unter einer hohen bzw. niedrigen Qualität der Prozessergebnisse zu verstehen ist. Nur so kann überhaupt ein gewinnbringendes Qualitätsmanagment eingesetzt werden und mit dem Einsatz von Standards, Reviews und Metriken ein brauchbares Resultat erzielt werden (vgl. auch Abschnitt 2). Vor allem muss aber vor dem Einsatz von Metriken klar sein, welches Ziel mit einer Reihe von Messungen erreicht werden soll und welche Entitäten dafür gemessen werden müssen. Erst dann sollte entschieden werden, welche konkreten Metriken dafür eigesetzt werden. Außerdem ist es wichtig, dass die gewonnen Messergebnisse, besonders auffällige Werte, kritisch analysiert werden. Die Qualität einer Software sollte keinesfalls ausschließlich auf Grundlage von Metriken beurteilt werden. Metriken können ein gutes Werkzeug für die Unterstützung des Qualitätsmanagements sein, es aber nicht ersetzen. Werden zu viele überflüssige Metriken erhoben oder sind die zu erreichenden Ziele nicht klar, kann die Verwendung von Metriken sogar vorhandene Probleme verschleiern und wird auf Dauer keine Akzeptanz bei den Projektbeteiligten erreichen [vgl. 7, S. 317f].

Die in Abschnitt 4 vorgestellten Metriken sollen keinesfalls eine uneingeschränkte Empfehlung für zu verwendende Metriken sein. Historisch bedingt handelt es sich bei diesen dreien um die bekanntesten und sollen einen ersten Einblick in die Welt der Metriken bieten. Für einen Einsatz in realen Projekten müssen gezielt spezielle Metriken ausgewählt werden die eine größtmögliche Aussage über die zu ermittelnden Kenngrößen treffen können. Dafür existieren spezielle Metriken die bspw. objektorientierte Eigenschaften in besonderem Maß berücksichtigen [vgl. 4] oder Risikofaktoren in Softwarearchitekturen messen können [vgl. 7].

Literatur

[1] Basili, V.R. ; Rombach, H.D.: The TAME project. Towards improvement-oriented software environments. In: *IEEE Transactions on Software Engineering* 14 (1988), Nr. 6

[2] Frühauf, Karol ; Ludewig, Jochen ; Sandmayr, Helmut: *Software-Prüfung.* 6. Auflage. Vdf Hochschulverlag, 2006

[3] Grady, Robert B.: Practical results from measuring software quality. In: *Communication of the ACM* 36 (1993), Nr. 11

[4] Hoffmann, Dirk W.: *Software-Qualität.* Springer, 2008

[5] Simon, Frank ; Seng, Olaf ; Mohaupt, Thomas: *Code-Quality-Management.* 1. Auflage. dpunkt.verlag, 2006

[6] Sommerville, Ian: *Software Engineering.* 8. Auflage. Person Studium, 2007

[7] Starke, Gernot: *Effektive Software Architekturen.* Carl Hanser Verlag, 2009

[8] Wallmüller, Ernest: *Software-Qualitätsmanagement in der Praxis.* 2. Auflage. Carl Hanser Verlag, 2001

[9] Wikipedia: *McCabe-Metrik.* – URL `http://de.wikipedia.org/wiki/McCabe-Metrik`. – Zugriffsdatum: 20. Februar 2010

[10] Wikipedia: *Messung.* – URL `http://de.wikipedia.org/wiki/Messung`. – Zugriffsdatum: 24. Februar 2010

[11] Wikipedia: *Qualität.* – URL `http://de.wikipedia.org/wiki/Qualität`. – Zugriffsdatum: 18. Februar 2010

[12] Wikipedia: *Softwaremetrik.* – URL `http://de.wikipedia.org/wiki/Softwaremetrik`. – Zugriffsdatum: 24. Februar 2010

[13] Zollondz, Hans-Dieter: *Grundlagen Qualitätsmanagement.* 2. Auflage. Oldenbourg, 2002