Die Komplexität der Traceability

Andrea Herrmann
Freie Software Engineering Trainerin und Forscherin
D-70372 Stuttgart, AndreaHerrmann3@gmx.de

Motivation

IT-Systeme und ihre Dokumentation werden immer komplexer. Gerade die Verwaltung von Traceability-Informationen ist hiervon betroffen, da die Anzahl möglicher Traceability-Beziehungen zwischen den Software- und Dokumentations-Elementen (z.B. zwischen Klassen, Testfällen und Anforderungen) mehr als linear mit der Größe des IT-Systems steigt.

Komplexität bewirkt bei der Software-Entwicklung, dass die Ergebnisse für den Menschen umso fehleranfälliger zu entwickeln und aufwändiger zu warten sind, je komplexer sie sind. Komplexität verursacht also Kosten. Dem Menschen bereitet Komplexität Schwierigkeiten bei der Wahrnehmung (Visuelle Belastung), beim Verstehen (Kognitive Belastung) und beim Schlussfolgern (Konzept der beschränkten Rationalität). Die begrenzte Speicherkapazität des Arbeitsgedächtnisses beschränkt die Anzahl an Modell-Elementen, die gleichzeitig wahrgenommen und verstanden werden können. Immerhin können Experten mehr Komplexität verarbeiten als Laien. Beim Laien geht man davon aus, dass er 7 ± 2, also maximal 9 Elemente gleichzeitig verstehen kann [8]. Für Experten darf ein Datenmodell auch bis zu 95 Elemente enthalten [6].

Definition: Komplexität und nützliche Metriken

Frederick Brooks [1] beschäftigte sich bereits in den 80er Jahren wegweisend mit der Komplexität von IT-Systemen. Er unterschied zwischen der „essential complexity“ und der „accidential complexity“, die sich zur Gesamtkomplexität aufsummieren. Die „essential complexity“ lässt sich nicht vermeiden, denn sie wohnt dem zu lösenden Problem inne. Sie entsteht durch die Anzahl der Elemente des Systems und deren Abhängigkeiten, von außen vorgegebene Konformitäts-Anforderungen an die Schnittstellen, die Veränderlichkeit und Unsichtbarkeit von Software, die es nötig macht, mehrere Modelle zu verwenden, um ein System vollständig zu beschreiben.

Die „accidential complexity“ entsteht zusätzlich und unnötig durch die gewählte Lösung und deren Darstellung; beispielsweise durch eine schlecht modularisierte Software-Architektur mit unnötig stark gekoppelten Komponenten oder durch eine ungünstige Darstellung von Informationen.

Das Ziel der Komplexitätsforschung im Software Engineering ist es üblicherweise, die Lösung passend zu gegebenem Problem so zu wählen, dass die accidential complexity minimal wird. Allerdings werden Problem und Lösung durch mehrere Variablen beschrieben, und wie sie die Komplexität beeinflussen, ist bisher nicht quantitativ bekannt. Mehr noch: Es gibt verschiedenste Komplexitätsmetriken.

Eine Komplexitätsmetrik ist dann nützlich, wenn sie geeignet ist, um eine praxisrelevante Größe wie Fehleranfälligkeit oder Wartungsaufwand des Systems vorherzusagen. Auf der Suche nach der besten Metrik wurde eine umfangreiche Studie durchgeführt, mit dem zweischneidigen Ergebnis, dass zwar Komplexitätsmetriken gut die Fehleranfälligkeit über die Zeit und Komponenten desselben Projektes voraussagen, aber für jedes Projekt sind dies jeweils andere Metriken [9].

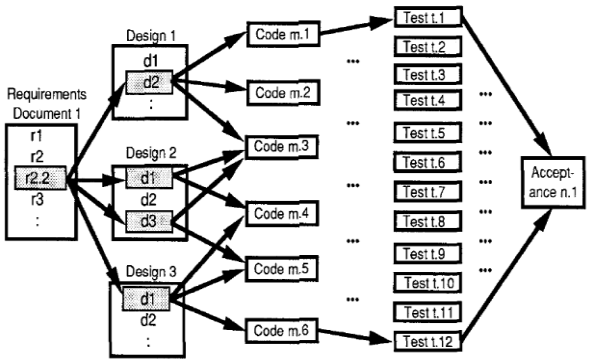

Abbildung 1: Traceability Graph [10]

Traceability und deren Komplexität

Traceability (deutsch: Nachverfolgbarkeit) bedeutet die Dokumentation von Beziehungen zwischen Projektergebnissen wie Anforderungen, Entwurfselementen, Code-Komponenten und Testfällen. Die Traceability lässt sich darstellen als Graph (wie in Abbildung 1), in einer Matrix, als Hyperlinks oder in einer Datenbank.

Die sogenannte horizontale Traceability beschreibt die Beziehungen zwischen Ergebnissen desselben Typs, also z.B. zwischen Anforderungen, die vertikale zwischen verschiedenen Typen, beispielsweise welche Code-Komponenten welche Anforderungen umsetzen und durch welche Testfälle getestet werden.

Traceability und deren Dokumentation wird dadurch komplex, dass sie vielfältige Beziehungen zwischen einer großen Anzahl von Elementen darstellt. Unnötig erhöht wird sie jedoch durch eine ungünstige

Modularisierung des Systems sowie durch eine ungünstige Darstellung (siehe [5]).

Eine umfangreiche Literaturrecherche zum Thema Komplexität brachte keine Metrik zutage, welche speziell die Komplexität von Traceability misst, wohl aber gängige Metriken, die für die Messung der Komplexität von Traceability-Dokumentationen nützlich sein könnten. Nützlich ist eine solche Metrik dann, wenn sie die Fehleranfälligkeit bei Erstellung / Inspektion oder den Aufwand bei der Wartung eines solchen Modells vorhersagen kann. Dies könnte für verschiedene Tätigkeiten eventuell eine andere Metrik sein [5]. Mehrere bewährte Metriken bieten sich als Kandidaten an:

1. Die zyklomatische Komplexität [7] ist eine graphentheoretische Größe, die u.a. für die Messung von Code-Komplexität eingesetzt wird. Sie berechnet sich als Anzahl Kanten – Anzahl Knoten + 2. Bei Code gilt ein Wert von 10 als sinnvolle Obergrenze.
2. Die Informationsflusskopplung von Henry und Kafura [4] misst den Datenfluss zwischen Komponenten mittels der gegenseitigen Aufrufe.
3. Die Formel 1 – Anzahl der Einheiten / Anzahl der Beziehungen von Sneed [11].
4. Kohärenz und Kopplung der Komponenten, ähnlich der objektorientierten Metriken von Chidamber und Kemerer [2], [3].

Komplexität verringern

Die Komplexität von Traceability-Dokumentation kann man wie bei Code durch Umstrukturierung (d.h. andere Aufteilung der Elemente auf Komponenten) verringern oder durch eine geeignete Darstellung verbergen. Z.B. lässt sich Komplexität verstecken durch eine fischaugenartige Fokussierung der Anzeige oder das Einziehen einer zusätzlichen Hierarchieebene in einem hierarchischen Modell.

Zusammenfassung und Ausblick

Die Komplexität von Traceability und Traceability-Dokumentationen zu messen ist bisher noch niemandem eingefallen, obwohl hieraus dieselbe Nützlichkeit entstehen kann wie bei der Komplexitätsmessung von Code: Eine gute Komplexitätsmetrik kann die Fehleranfälligkeit und Wartungsaufwände für Traceability-Dokumentationen vorhersagen und somit eine Kosten-Nutzen-Schätzung vor der Einführung oder beim Vergleich verschiedener Traceability-Ansätze unterstützen. Sind Grenzwerte bekannt, die man nicht überschreiten sollte (wie dies bei Code der Fall ist), existiert auch ein Indikator dafür, ab wann ein System zu komplex wird und neu strukturiert werden sollte.

Dieser Artikel schlägt mehrere Metriken vor, die bereits im Zusammenhang mit Code zur Messung von Komplexität verwendet werden. Ein nächster Schritt wäre es, die Nützlichkeit dieser Metriken empirisch zu untersuchen, beispielsweise anhand eines kontrollierten Experiments.

Referenzen

[1] Frederick P. Brooks, Jr.: "No Silver Bullet: Essence and Accidents of Software Engineering". Computer, Vol. 20, No. 4, 1987, pp. 10-19, www.cs. unibo.it/~cianca/wwwpages/ids/letture/Brooks.pdf

[2] S.R. Chidamber, C.F. Kemerer: „Towards a Metrics Suite for Object Oriented design". Proceedings of the Conference on Object-Oriented Programming: Systems, Languages and Applications OOPSLA'91, October 1991, Vol. 26, No. 11, 1991, pp. 197-211

[3] S.R. Chidamber, C.F. Kemerer: „ A Metrics Suite for Object Oriented Design". IEEE Transactions on Software Engineering, Vol. 20, No. 6, 1994, pp. 476-493

[4] S. Henry, D. Kafura: "Software structure metrics based on information flow". IEEE Transactions on Software Engineering 7(5) 1981, pp. 510-518

[5] Y. Li, W. Maalej: Which Traceability Visualization Is Suitable in This Context? A Comparative Study. Proceedings of the 18th International Working Conference, REFSQ 2012, Essen, Germany, March 2012, Springer, Lecture Notes of Computer Science LNCS 7195, pp. 194-210

[6] R. Maier: "Benefits And Quality Of Data Modelling: Results Of An Empirical Analysis". Proceedings of the fifteenth International Conference on the Entity Relationship Approach, 1996

[7] T.J. McCabe: "A complexity measure". IEEE Transactions on Software Engineering 2(4) 1976, pp. 308-320

[8] G.A. Miller: "The Magical Number Seven, Plus Or Minus Two: Some Limits On Our Capacity For Processing Information". The Psychological Review 63, 1956, pp. 81-97

[9] N. Nagappan, T. Ball, A. Zeller: "Mining Metrics to Predict Component Failures". Proc. 28th Conference on Software Engineering (ICSE), Shanghai, China, 2006, ftp://ftp.research.microsoft. com/pub/tr/TR-2005-149.pdf

[10] S.L. Pfleeger, S.A. Bohner: "A Framework for Software Maintenance Metrics". Conference on Software Maintenance, IEEE Computer Society Press, Los Alamitos CA, 1990, pp. 320–327

[11] H.M. Sneed: "Design Metrics for UML Models". Proc. Of the joined International Conference on Software Measurement IWSM/ MetriKon/ Mensura 2010, Stuttgart, Germany, pp. 115-139