42 Jahre Komplexitätsmetriken - Was stoppt uns?

Software-Komplexitätsmetriken effektiv einsetzen

Thomas Grundler, Hendrik Post, Jochen Quante, Sadi Yigit;
Robert Bosch GmbH

Die wohl bekannteste Software-Komplexitätsmetrik wurde bereits 1976 von Thomas J. McCabe eingeführt und führt seit Generationen von Software-Entwicklern zu Diskussionen über die Aussagekraft von Metriken. Im folgenden Beitrag wird der Umgang mit Software-Komplexitätsmetriken im „Automotive Bereich“ der Robert Bosch GmbH geschildert.

Ausgangslage

Unser Ziel ist es, Software-Produkt-Metriken zu finden, die wiedergeben, bei welchen Software-Komponenten die Entwickler Schwierigkeiten haben, diese zu verstehen und diese dann weiter zu entwickeln. Die generelle Frage also lautet: wie kann die Wartbarkeit von Software gemessen werden und wie kann diese Messung zur Steuerung des Software-Entwicklungsprozesses eingesetzt werden?
Es gibt viele Metriken: „Lines of Code“ kam schon in den 1960er Jahren auf, 1976 wurde die zyklomatische Komplexität von McCabe eingeführt und in 1980igern und 1990igern sind viele weitere Metriken entstanden, die den einen oder anderen Aspekt von Software betrachten.

Im Automotive-Bereich kamen dann 2005 die HIS-Metriken dazu. HIS steht für „Herstellerinitiative Software“ und die HIS bestand aus den Automobilherstellern Audi, BMW Group, DaimlerChrysler, Porsche und Volkswagen. Das Ziel der HIS war, eine Grundmenge an Metriken zur Bewertung der Testbarkeit und Software-Qualität allgemein festzulegen. Die HIS-Metriken bestehen aus 15 Software-Metriken mit entsprechenden Grenzwerten, die für jede „kompilierbare Einheit“ erhoben werden sollen.

Unsere Erfahrung heute ist, dass sehr viel gemessen wird. Es gibt eine unüberschaubare Menge an Metriken und Tools auf dem Markt, die alle für sich beanspruchen, die Qualität des Software-Produkts messen und beurteilen zu können. Nach unserem Kenntnisstand werden aber aktuell nur wenige Konsequenzen aus diesen Messungen gezogen und oft ist nicht klar, was mit der Messung von bestimmten Metriken überhaupt erreicht werden soll. Und auch die Messung einer großen Anzahl an Metriken (evtl. noch mit ihren individuellen Verstößen gegen eine vorgegebene Grenze) ist oft nicht hilfreich, um daraus die notwendigen Schlüsse zu ziehen.

Vorarbeit

Um dem Ziel näherzukommen, die Wartbarkeit von Software mit Hilfe von Produkt-Metriken messen zu können, wurde das „MI-Verfahren“ („Maintainability Index“) nach Oman und Hagemeister angewandt (*P. Oman, J. Hagemeister: "*Construction and Testing of Polynomials Predicting Software Maintainability*", in Journal of Systems and Software, Vol. 24, No. 3, 1994.*). Zunächst wurde eine repräsentative Auswahl von sog. Programmständen der Bosch Motorsteuerung vermessen. Gemessen wurde auf dem C-Code, unabhängig davon, ob dieser handgeschrieben oder generiert wurde. Die gemessenen Metriken bestanden aus den HIS-Metriken

und vielen weiteren aus der Literatur bekannten Komplexitätsmetriken wie z.B. Halstead, cognitive complexity, Ausreißer oder variable lifespan.

Der nächste Schritt bestand in der Berechnung von Korrelationen zwischen den Metriken, um Metriken mit möglichst orthogonaler Aussagekraft zu erhalten. Dazu wurde eine sog. Hauptkomponentenanalyse eingesetzt. Die Hauptkomponenten-analyse liefert 2 Aussagen: zum einen, welche Metriken ein sog. Cluster bilden (also stark korreliert sind) und zum anderen welchen Anteil an der Gesamtinformation dieses Cluster liefert. Anschließend wurde jeweils ein Repräsentant dieses Clusters ausgewählt und mit Hilfe der Aussage über den Informationsanteil des Clusters wurde dann ermittelt, wie viele Cluster in die weiteren Schritte mitgenommen werden, ohne zu viel an Information zu verlieren. Tabelle 1 zeigt ein typisches Ergebnis der Hauptkomponentenanalyse.

Cluster	**Informationsanteil**	**Metriken (Auszug)**
1	70%	Lines Of Code, Number of Statements in Function, Cyclomatic Complexity, Halstead Difficulty, …
2	9%	Estimated Static Program Paths, Number of Local Variables Declared, …
3	7%	Deepest Level of Control Flow Nesting, …
4	6%	Number of function parameters, …
5 .. 12	< 2%	

Tabelle 1: Ergebnis der Hauptkomponentenanalyse

Speziell in den größeren Softwarefunktionen korrelieren viele Metriken mit „Lines Of Code“ und dieses Cluster beinhaltet auch den größten Anteil an der Gesamtinformation.

Designguidelines (HIS-Metriken neu bewertet)
Mit dem Wissen der Korrelationen unter den Metriken wurden verschiedene Metriken ausgewählt, die einerseits alle Cluster umfassen und andererseits eine gewisse Steuerwirkung haben. Steuerwirkung in dem Sinne, dass eine Verbesserung der Metrik auch eine Verbesserung der Software Qualität nach sich zieht und mit modernen Entwicklungsparadigmen in Einklang steht. Außerdem wurde Wert darauf gelegt, dass die Metriken relativ einfach zu ermitteln und nachzuvollziehen sind. Damit wird dem Entwickler die Möglichkeit gegeben, im Vorfeld abzuschätzen, welche Auswirkung eine bestimmte Codeänderung auf den neuen Wert dieser Metrik hat. Um Metriken als Designguideline einzusetzen, müssen sie zudem ein gewisses Maß an Selektivität aufweisen, d.h. der gewählte Grenzwert der Metrik darf nicht dazu führen, dass praktisch alle Funktionen einer Gesamtsoftware detektiert werden.

Mit diesem Hintergrund und der Erfahrung aus zahlreichen Entwickler-Reviews und -Interviews wurden 6 HIS-Metriken mitsamt Grenzwerten ausgewählt, die als Designguideline für neue Software-Module dienen, die in C programmiert werden. Tabelle 2 zeigt diese Metriken.

Metrik	Bezeichnung QA-C®	Empfohlener Bereich
Zyklomatische Komplexität	STCYC	1 .. 20
Anzahl „Statements“ pro Funktion	STST3	1 .. 100
Statische Schätzung der möglichen Pfade	STPTH	1 .. 1000
Max. Verschachtelungstiefe der Kontrollstrukturen	STMIF	0 .. 6
Anzahl Sprunganweisungen	STGTO	0
Anzahl Funktionsparameter	STPAR	0 .. 12

Tabelle 2: Designvorgabe bei Software-Modulen geschrieben in C

BMI (Bosch Maintainability Index)
Zur Steuerung und Priorisierung des Software-Entwicklungsprozesses war eine weitere Vereinfachung und Reduktion der Kenngrößen sinnvoll und notwendig. Deswegen wurde der 2. Teil des „MI“-Verfahrens nach Oman und Hagemeister angewandt.

Von den Funktionsexperten wurde ihre Meinung zur Wartbarkeit der entsprechenden Funktion eingeholt. Dabei wurde darauf geachtet, die Komplexität / Wartbarkeit im Verhältnis zum zu lösenden Problem zu setzen, um eine Vergleichbarkeit herzustellen. Über eine lineare Regression unter Verwendung der für Wartbarkeit signifikanten Metriken / Cluster wurde eine Formel entwickelt, die einen guten Indikator für die Wartbarkeit des Software-Moduls darstellt.

Dieser sog. BMI (Bosch Maintainability Index) setzt sich also aus mehreren Metriken zusammen und ist dabei normiert auf +100 (sehr gute Wartbarkeit) bis -100 (sehr kritische Wartbarkeit). Alle verwendeten Metriken wirken dabei in die gleiche Richtung, d.h. je größer die Metriken werden, desto schwerer sind die Funktionen zu warten. Der BMI wird bei handgeschriebenem C-Code und C-Code, der mit dem Tool „ETAS ASCET“ erzeugt wird, angewandt. Bei C-Code, der aus Simulink®-Modellen heraus generiert wird, lassen sich mit diesen Codemetriken wenig Rückschlüsse auf das Modell ziehen. Deswegen wurde ein „ModelBMI“ entwickelt, der Metriken direkt auf dem Simulink®-Modell misst und der sich aktuell in der Pilotierungsphase befindet.

Die Trefferquote des BMI beträgt ca. 80%, d.h. 80% der vom BMI-Verfahren als komplex / schwer wartbar eingestuften SW-Module werden auch von den Experten so eingestuft.

Nicht alle Aspekte von „Wartbarkeit“ können in einen einzigen Index gepackt werden (auch nicht durch die Verwendung von deutlich mehr Metriken) – wir lösen das Problem durch die entsprechende Einbindung in den Software-Entwicklungsprozess.

Prozess
Kundenprojekte vermessen zu Projektstart ihre Software und starten eine Diskussion mit den Experten über die Befunde. Es werden nur solche Module restrukturiert, die auch nach Expertenmeinung schwer wartbar sind und bei denen häufige Änderungen in Zukunft erwartet werden. Dazu werden sog. Assessments genutzt, in denen eine Expertenrunde eine wohl begründete und dokumentierte Entscheidung trifft. Software, die seit vielen Jahren problemlos im Feld ist und die nicht mehr geändert wird, wird nicht angefasst.

Mindset
Unser Ziel ist es, Software-Produkt-Metriken zu finden, die wiedergeben, bei welchen Software-Komponenten die Entwickler Schwierigkeiten haben, diese zu verstehen und diese dann weiter zu entwickeln. Der darauffolgende Restrukturierungsschritt soll dann einer Verbesserung der Software – und nicht ausschließlich der Metriken dienen. Wir sehen Metriken als keine absoluten Schranken an, auf die unbedingt Aktionen folgen müssen. Wir vertrauen unseren Entwicklern mehr als den Metriken und lassen begründete Ausnahmen zu.

Bei Unterschreitung eines gewissen BMI-Wertes gibt es aber ein „Muss“ für eine Diskussion mit Dokumentation der Ergebnisse im SCM-System. Dadurch werden die Metriken eher als Unterstützung, denn als Gängelung angesehen. Auch die einfache Benutzbarkeit (Metriken inkl. BMI werden standardmäßig bei den statischen Codechecks den Entwicklern zur Verfügung gestellt) und die Konzentration auf wenige Größen tragen zur Akzeptanz des Messsystems bei. Auch die Einrichtung eines Unterstützungs-Teams für die Software-Entwickler wurde sehr positiv aufgenommen: dieses Team unterstützt sowohl in der Bewertung der Befunde wie auch bei der konkreten Restrukturierung der Software.

Fazit
Mit dem richtigen Mindset (den Entwicklern mehr als den Metriken vertrauen) und der Konzentration auf wenige Kenngrößen sind Metriken eine sinnvolle und wichtige Ergänzung des täglichen Software-Entwicklungs-Geschäfts. Entwickler müssen in Metriken eine Unterstützung erfahren, die eine offene und objektive Diskussion über die Wartbarkeit von „ihren“ Modulen ermöglicht. Durch diese Offenheit ergab und ergibt sich die gewünschte Steuerungswirkung der Komplexitätsmetriken, die zu wartbareren und weniger komplexen Modulen führt.

Auch beim Management können Metriken eine deutlich erhöhte Transparenz von „Internas“ der Software erzeugen. Mit ihrer Hilfe konnte z.B. gezeigt werden, dass eine gewisse Alterung bei langlebiger Software normal ist, Software daher von Zeit zu Zeit auf neue Randbedingungen angepasst werden muss und dass anschließend die restrukturierte Version eine gute Basis für die Weiterentwicklung darstellt. Aus unserer Sicht haben Komplexitätsmetriken auch im fortgeschrittenen Alter von 42 Jahren nichts von ihrem Nutzen – aber auch nichts von ihrem Diskussionspotential – verloren.

Autor
Thomas Grundler hat 20 Jahre Erfahrung in Software-Entwicklung in Embedded Systemen. Seine Schwerpunkte dabei sind Software Analyse und Software Wartung. Seit 2004 ist Thomas Grundler bei der Robert Bosch GmbH tätig und verantwortet das Thema „Komplexitätsmessung und Komplexitätsreduktion“ der klassischen Motorsteuerungs-Software des Geschäftsbereichs „Powertrain Solutions“.

Kontakt
Thomas Grundler
Robert-Bosch-Strasse 2
71701 Schwieberdingen
E-Mail: thomas.grundler@de.bosch.com