Qualität sicherheitsrelevanter Software für speicherprogrammierbare Steuerungen

Prof. Dr. Dietmar Reinert, BGIA – Institut für Arbeitsschutz der deutschen gesetzlichen Unfallversicherung, Zentralbereich
Prof. Dr. Norbert Jung, Fachhochschule Bonn-Rhein-Sieg, Fachbereich Informatik
Thomas Breuer (BSc), BGIA – Institut für Arbeitsschutz der deutschen gesetzlichen Unfallversicherung, Zentralbereich

Quality of safety-related software for programmable logic controllers

A review of the present literature showed that there is no tool available which measures the quality of safety-related software of programmable logic controllers. We investigated therefore the specialities of PLC-sources programmed according to IEC 61131-3. Based on this analysis we propose 16 new metrics for PLC programs. Additional to seven of the new metrics five metrics derived from Halstead and McCabe are used to determine the quality-criteria: testability, self-description, legibility and simplicity. This new approach is applicable to all high-level languages used for safety applications. The results are documented in hyper-language files with links to the called modules and control-graphs for each function.

PLC, safety-related, static analysis, quality, metrics

1. Einführung

Die Steuerung sicherheitsrelevanter Funktionen von Maschinen wird heute vermehrt durch Software vorgenommen. Bei der Entwicklung und Prüfung sicherheitsrelevanter Software werden nach IEC 61508-3 zahlreiche Verfahren eingesetzt, um Fehler bei der Programmierung zu vermeiden. Die statische Analyse von Software ist dabei ein beliebtes Werkzeug, um Schwachstellen frühzeitig zu erkennen bzw. komplexe Algorithmen zu identifizieren, damit sie ausgiebigen Tests unterzogen werden können. Gemeinsam mit dem Fachbereich Informatik der Fachhochschule Bonn/Rhein-Sieg wurde ein Werkzeug entwickelt, mit dem die Qualität von SPS Software überprüft werden kann. Das Werkzeug steht kostenlos im Internet zum Download zur Verfügung. Der Beitrag beschreibt, wie die Qualität gemessen wird und worauf bei der Benutzung des Werkzeuges zu achten ist.

2. Sicherheitsfunktionen werden heute in Software realisiert

Seit Beginn der 90er Jahre dringen rechnergesteuerte Systeme vermehrt in die Sicherheitstechnik ein. Waren es zunächst maßgeschneiderte Systeme mit einem Hardwarekanal so hat sich das spätestens seit Einführung der sicheren

speicherprogrammierbaren Steuerung in so fern verändert, dass eine standardisierte Hardware eingesetzt wird und die Sicherheitsfunktionen im Wesentlichen in Software realisiert werden. Heute liegen für die Wesentlichen sicherheitsrelevanten Funktionen fertige Funktionsbausteine vor, die aber richtig parametriert und sinnvoll eingesetzt werden müssen. Qualität der Software spielt in diesem Zusammenhang nicht nur in Bezug auf die Sicherheitstechnik eine immer größere Rolle [1].

Die Disziplin des Softwareengineerings hat in den letzten 20 Jahren zahlreiche Maßnahmen und Methoden entwickelt, die Qualität von Software systematisch zu untersuchen und zu verbessern. Grundsätzlich unterscheidet man dabei analytische Verfahren und testende Verfahren [2]. Zu den analytischen Verfahren gehören neben der Fagan-Inspektion und dem Walk-Through auch die Animation und die statische Analyse. Die statische Analyse kann durch Zuhilfenahme von so genannten Softwaremetriken (Maßzahlen) für die Qualität der Software rechnerunterstützt durchgeführt werden. Sie hilft Schwachstellen in der Software frühzeitig zu erkennen bzw. komplexe Algorithmen zu identifizieren, die dann durch testende Verfahren (White-Box-Tests) in ihrem Echtzeitverhalten untersucht werden können. Eine schlecht kommentierte Software, die zudem ein umfangreiches Vokabular benutzt und komplexe Rechenoperationen einsetzt, ist wenn sie zusätzlich schlecht strukturiert ist, sehr fehleranfällig und kaum wartbar. Damit sind bereits einige Kriterien angesprochen, die im Qualitätscheck von Software berücksichtigt werden müssen.

3. Statische Analyse umfangreicher Programme zeigt Schwachstellen auf

Obwohl der Trend heute in Richtung graphische Programmierung weist, gibt es immer noch zahlreiche, umfangreiche SPS Programme, die in Anweisungsliste geschrieben sind. Selbst ein umfangreiches Programm in Kontaktplandarstellung oder Funktionsplandarstellung ist häufig nicht einfach zu verstehen und zu warten. In einem Forschungsvorhaben an der Fachhochschule Bonn-Rhein-Sieg, Fachbereich Informatik, wird von den Autoren deshalb seit einigen Jahren das Thema Softwarequalitätssicherung durch Einsatz von Metriken und Qualitätskriterien für sehr hardwarenahe Sprachen wie Assembler [3] oder AWL [4], aber auch für Hochsprachen prozedualer Art wie C [5] oder objektorientierter Art wie C++ [6] und JAVA [7] bearbeitet. Es zeigte sich dabei früh, dass kommerziell erhältliche Tools auf die Besonderheiten der Sicherheitstechnik keine Rücksicht nehmen. Für die Analyse der SPS Sprachen gibt es zudem keine kommerziellen Tools zur Bestimmung der Qualität sicherheitsrelevanter Software. In verschiedenen Abschlussarbeiten an der Fachhochschule wurden deshalb nicht nur die altbekannten linguistischen Metriken von Halstead oder strukturellen Metriken von McCabe angepasst und erweitert, sondern insbesondere für speicherprogrammierbare Software neue Metriken entwickelt. Aus wichtigen zentralen Metriken wurden dann vier Qualitätskriterien erarbeitet. Wie kann man sich nun den Zusammenhang zwischen Qualität und Softwaremetrik vorstellen?

4. Metriken für sicherheitsbezogene Software

Bereits in den 70iger Jahren haben u.a. Halstead [8] und McCabe [9] Maßzahlen (Metriken) für die statische Beurteilung von Software entwickelt. Aus der Erfahrung mit umfangreichen Softwareprojekten konnten die Autoren Vorgaben machen,

innerhalb derer sich die verschiedenen Maßzahlen bewegen müssen, um von einer entsprechend robusten Software, die gut wartbar ist, sprechen zu können. Diese Metriken gingen damals weder ausreichend auf die strukturierten Programmiersprachen ein noch berücksichtigten sie die Besonderheiten der objektorientierten Sprachen. Spätere Arbeiten [10], [11], [12] gingen auf diese neueren Entwicklungen ein, sodass heute zahlreiche Metriken zur Verfügung stehen. Die Bewertung sicherheitsrelevanter Software wird allerdings auch von den neueren Entwicklungen nicht berücksichtigt. Insbesondere auf dem Gebiet der Programmierung von Speicherprogrammierbaren Steuerungen stehen unseres Wissens zurzeit keine Metriken zur Bewertung zur Verfügung.

In Auseinandersetzung mit dem Werkzeug Logiscope der Fa. Verilog [13] haben die Autoren die dort verwendeten Metriken und Qualitätskriterien auch sicherheitsbezogene Software angepasst. Dazu wurden zahlreiche Programme zum Maschinenschutz aus der Industrie untersucht, neue Metriken entwickelt sowie bewährte Metriken und deren Grenzwerte angepasst [14]. Einen Überblick darüber gibt Tabelle 1.

Bezeichnung der Metrik	**Kurzbe-zeichnung**	**Berechnung**	**obere Grenze**	**untere Grenze**
Durchschnittliche Verwendung der Operanden	N2_F	$\frac{N2}{n2}$	1	3
Anzahl unterschiedlicher Aussprünge	NP_NODES	Ermittelt	0	1
Anzahl indirekter Aussprünge	IP_NODES	Ermittelt	0	5
Summe aller maximalen Grade	S_MAX_DEG	Ermittelt	0	10
Auflistung aller maximalen Grade	A-MAX_DEG	Ermittelt	n.a.	n.a.
VG ohne CASE-Strukturen	VG_S_MAX_DEG	$VG - S\_MAX\_DEG$	1	10
VG ohne Fehlerverwaltung	VG_NEU	$VG - S\_MAX\_DEG - (IP\_NODES - NP\_NODES)$	1	8
Anzahl "tote" Sprungmarken	N_DEAD_LABELS	Ermittelt	0	0
Anzahl der überflüssigen Sprungmarken	N_WASTE_JUMPS	Ermittelt	0	0
Anzahl der einzeiligen Schleifen	N_LOOP_LINE	Ermittelt	0	0
Anzahl Schleifen	N_LOOP	Ermittelt	0	0
VG mal Anzahl Programmzeilen	VG_STMTS	$VG * N\_STMTS$	1	500
Erweiterte Frequenz Kommentare	NEU_COM_R	$\frac{N\_COM * VOC\_F}{N\_STMTS + (VG\_S\_MAX\_DEG - 1) * 2}$	0,5	8

Tabelle 1: Auf die Sicherheitstechnik angepasste Metriken [14]

Neben dieser Adaption auf die Sicherheitstechnik wurden neue Metriken speziell für die SPS-Programmierung entwickelt. Dabei wurde vor allem die Verwendung von Ein- und Ausgangsvariablen, die Bezeichnung von Variablen und Programmorganisationseinheiten und der Einsatz von Timern berücksichtigt. Tabelle 2 gibt einen Überblick über die neuen Metriken, die die klassischen Metriken

ergänzen sollen. Die vorgeschlagenen Grenzwerte entstammen den Erfahrungen der Autoren mit sicherheitsbezogenen SPS-Programmen, sind aber noch durch die industrielle Praxis weiter zu verifizieren.

Bezeichnung der Metrik	**Kurzbezeichnung**	**Berechnung**	**obere Grenze**	**untere Grenze**
Anzahl unterschiedlicher direkt aufgerufener Komponenten	N_DRCT_CALLS	Ermittelt	0	9
Anzahl globaler Variablen	N_GLOB_VAR	Ermittelt	0	3
Anzahl lokaler Variablen	N_LOC_VAR	Ermittelt	0	10
Anzahl Eingangsvariablen	N_IN_VAR	Ermittelt	1	5
Anzahl Ausgangsvariablen	N_OUT_VAR	Ermittelt	1	5
Gesamtzahl aller Variablen	N_ALL_VAR	*N_GLOB_VAR + N_LOC_VAR+N_IN_VAR + N_OUT_VAR*	2	15
Gesamtlänge aller Variablen	L_ALL_VAR	Ermittelt	n.a.	n.a.
Durchschnittliche Länge aller Variablennamen	L_VAR	$\frac{L\_ALL\_VAR}{N\_ALL\_VAR}$	6	50
Länge des Namens der Programm-organisationseinheit	L_POE	Ermittelt	6	50
Anzahl der Variablen nicht primitiver Datentypen	N_NP_VAR	Ermittelt	0	50
Anzahl beschriebener Eingangsvariablen	N_WR_IN	Ermittelt	0	0
Anzahl gelesener Ausgangsvariablen	N_RD_OUT	Ermittelt	0	0
Verhältnis IN/OUT-Variablen zu anderen Variablen	N_IO_TO_OTHER	$\frac{N\_GLOB\_VAR + N\_LOC\_VAR}{N\_IN\_VAR + N\_OUT\_VAR}$	0	2
Anzahl gesetzter Ausgänge	N_SET_OUT	Ermittelt	n.a.	n.a.
Anzahl Zuweisungen pro Ausgang	N_ASS_OUT	$\frac{N\_SET\_OUT}{N\_OUT\_VAR}$	1	1
Häufigkeit von direkter Speicheradressierung	N_DRCT_ADD	Ermittelt	0	0
Anzahl Zuweisungen	N_ASS	Ermittelt	n.a.	n.a.
Durchschnittliche Häufigkeit der Veränderung der Variablen	N_C_VAR	$\frac{N\_ASS}{N\_ALL\_VAR}$	0	1
Anzahl Sprungmarken innerhalb einer Klammer	N_P_JMPS	Ermittelt	0	0
Anzahl Timer	N_TIMER	Ermittelt	0	4
Häufigkeit der Abfrage von Timern	N_QUERY_T	Ermittelt	0	12
Anzahl übersprungener Timerbefehle	N_T_JMPS	Ermittelt	0	0

Tabelle 2: Spezifische Metriken für SPS-Software [4]

Die in den Tabellen genannten Metriken bilden die wesentliche Grundlage für die Bestimmung der Qualität sicherheitsrelevanter Software.

5. Qualität, die man messen kann

Die vier gewählten Qualitätskriterien lauten Selbstbeschreibung, Testbarkeit, Einfachheit und Lesbarkeit:

Die gute Selbstbeschreibung ist eine besondere Anforderung an Software mit Sicherheitsaufgaben, da diese während der Verifizierung und der Zertifizierung von mehreren Personen gelesen und nachvollzogen werden muss. Maßnahmen, die das Nachvollziehen unterstützen, sind: viele Kommentare, selbsterklärende Variablennamen, ein klar strukturierter und überschaubarer Kontrollflussgraph und eine überschaubare Modullänge. Konkret fließen in dieses Kriterium folgende Metriken ein: die Kommentarrate in Abhängigkeit von der Komplexität und dem Umfang eines Moduls (NEW_COM_R), die durchschnittliche Länge aller Variablennamen (L_VAR) und die Länge des Namens einer Programmorganisationseinheit (L_POE). Liegen alle drei Metriken im Bereich ihrer Grenzen, ist das Kriterium Selbstbeschreibung akzeptiert, liegen die Länge der Namen außerhalb der Grenzen, erscheint der Hinweis TO IMPROVE NAMING und liegt die Kommentierung außerhalb der Grenzen erscheint der Hinweis TO DOCUMENT.

Auch nach einer Codeinspektion lassen sich durch die Echtzeitanforderungen nur begrenzt Aussagen über das korrekte Verhalten des Gesamtsystems im Betrieb machen. Aus diesen Gründen ist es notwendig, Testfälle während allen Entwicklungsphasen des Systems durchzuführen. Diese Testfälle gestalten sich meist sehr umfangreich, da möglichst alle wichtigen Pfade der Software durchlaufen werden sollen. Eine starke Modularisierung und wenig verschachtelte Programme vereinfachen das Testen. Bei der Testbarkeit werden deshalb folgende Metriken betrachtet: die Komplexität des Moduls (zyklomatische Zahl - VG), die Anzahl der Rücksprünge (N_DRCT_CALLS), die Komplexität des Moduls in Abhängigkeit von seinem Umfang (VG_STMTS) und die Häufigkeit der Abfrage von Timern (N_TIMER). Das Modul ist im Bezug auf dieses Kriterium TO INSPECT, wenn zu häufig Timer eingesetzt wurden, TO STRUCTURE, wenn es zu viele Rücksprünge oder eine zu hohe Komplexität gibt und TO CUT, wenn das Modul zu lang ist.

Eines der Hauptargumente für den Einsatz von Software ist die Flexibilität, Anpassungen an ihr vornehmen zu können. Für eine leichte und sichere Einarbeitung ist es wichtig, dass das System so einfach wie möglich aufgebaut ist. Die Schnittstellen der einzelnen Module sollten klar definiert und abgegrenzt sein. Der Programmcode der einzelnen Module sollte überschaubar und leicht verständlich sein. Dieses Kriterium der Einfachheit wird durch folgende Metriken bewertet: die Verschachtelung des Moduls ohne Fehleraussprünge auf ein einziges Modul (VG_NEW), Komplexität des Moduls in Abhängigkeit von seinem Umfang (VG_STMTS), das Verhältnis der Ein/Ausgangsvariablen zu anderen Variablen (N_IO_TO_OTHER), die Anzahl der Timer (N_TIMER) und das Verhältnis der Anzahl der Timer zu deren Abfragehäufigkeit (N_QUERY_T). Ist keine dieser Metriken erfüllt, so muss das Modul neu geschrieben werden (TO REWRITE), ist die Komplexität in Abhängigkeit vom Umfang zu groß, lautet das Ergebnis TO CUT. Liegt

außer der Komplexität in Abhängigkeit vom Umfang eine der anderen Metriken außerhalb der Grenzen, lautet das Ergebnis TO INSPECT.

Software mit sicherheitsrelevanten Aufgaben sollte nicht nur in der Lage sein, auf externe Einflüsse (Sensoren) zu reagieren, sondern auch interne Fehler erkennen. Für derartige Fehler muss eine einfache und einheitliche Fehlerroutine existieren. Die Nutzung von CASE-Strukturen und Fehlerausprüngen ist notwendig. Schleifen erschweren die Lesbarkeit und sollten sparsam verwendet werden. Langer linearer Programmtext ist ähnlich unkritisch wie kurzer, verschachtelter Code. Operatoren sollten nicht zu oft verändert werden. Dies führt zum Kriterium der Lesbarkeit. Folgende Metriken bestimmen die Lesbarkeit: Verschachtelung eines Moduls ohne Auswahlstrukturen und Fehlerausprünge (VG_NEW), die Anzahl der Rücksprünge (N_LOOP), die Komplexität in Abhängigkeit vom Umfang des Moduls (VG_STMTS) und die durchschnittliche Häufigkeit der Veränderung von Variablen (N_C_VAR). Die Lesbarkeit ist ACCEPTED, wenn alle Metriken erfüllt sind, TO INSPECT, wenn Verschachtelung und Komplexität zu groß sind und TO STRUCTURE, wenn die Anzahl der Rücksprünge und die durchschnittliche Häufigkeit der Veränderung der Variablen außerhalb der Grenzen liegen.

Für jede Einzelmetrik können die Grenzen für ihre Gültigkeit im Tool zum Qualitätscheck angepasst werden. Die vorgegebenen Standardwerte sind zum großen Teil der Literatur entnommen oder haben sich aus der inzwischen zwanzigjährigen Praxis der BGIA mit sicherheitsrelevanter Software ergeben. Aus den vier oben genannten Qualitätskriterien wird die globale Qualität ermittelt, zu der die oben genannten Aussagen aus den Einzelkriterien gebildet werden.

Zusätzlich zur Bestimmung der Softwarequalität erlaubt das im Internet unter: www.dguv.de Webcode 2386566 zur Verfügung gestellte Werkzeug (siehe Abbildung 1) eine graphische Aufbereitung der untersuchten Software. In einem HTML-Dokument wird die Aufrufstruktur der Software über Hyperlinks sichtbar gemacht.

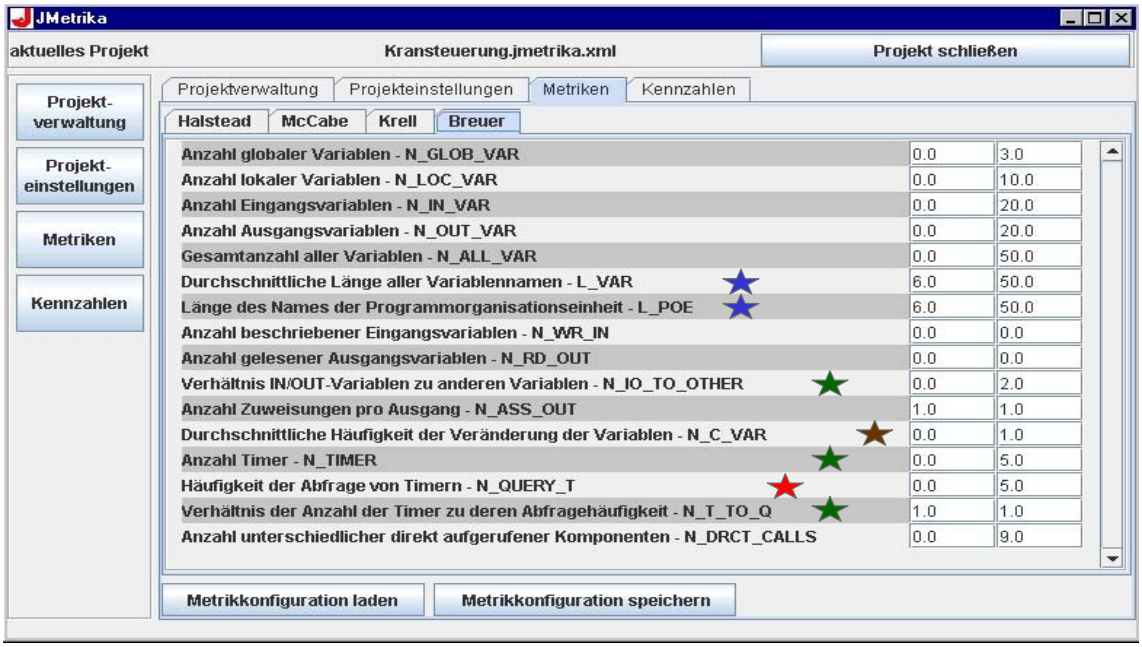

Abbildung 1: Oberfläche des Softwarewerkzeugs zur Qualitätssicherung [4]

Für jedes Modul gibt es eine graphische Darstellung des Kontrollflusses. Damit lässt sich die Komplexität eines Moduls sofort graphisch darstellen. Ein so genannter Kiviatgraph zeigt auf einen Blick, welche Metriken außerhalb der Normwerte liegen. Auf diese Weise generiert das Werkzeug neben der Analyse der Qualität auch eine Unterstützung zur Dokumentation des Quelltextes. Jede Analyse dokumentiert das Werkzeug durch eine HTML-Datei, die neben den Standardinformationen über die untersuchte Software alle Module graphisch beschreibt sowie die ermittelte Qualität insgesamt als auch jedes einzelnen Moduls dokumentiert.

6. Die Software zum Qualitätscheck

Das Programm JMetrikaAWL kann Quelltexte, die der Norm IEC 61131 Teil 3 entsprechen oder in der Siemens Sprache STEP 7 implementiert sind, analysieren. Alle anderen Quelltexte müssen geringfügig angepasst werden, um analysiert werden zu können. Dazu sind die Programmorganisationseinheiten und die Variablendeklarationen in der entsprechenden normkonformen Weise darzustellen. Möglicherweise sind die Sprung-, Aussprung- und Callbefehle ebenfalls auf die normkonformen Befehle hin zu ändern. Eine letzte Anpassung betrifft die Variablendeklarationen für Timer, die ebenfalls normkonform sein müssen.

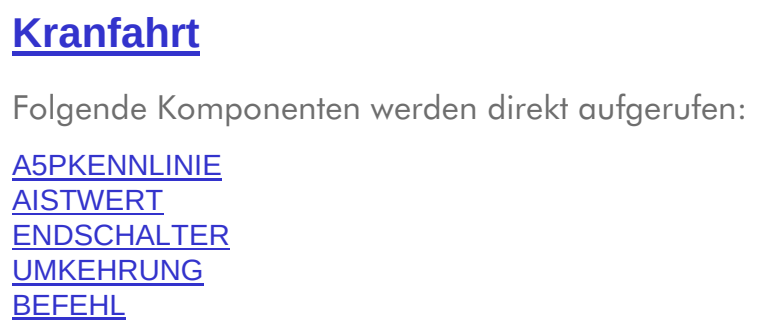

Abbildung 2: Verdeutlichung der Aufrufe durch Hyperlinks

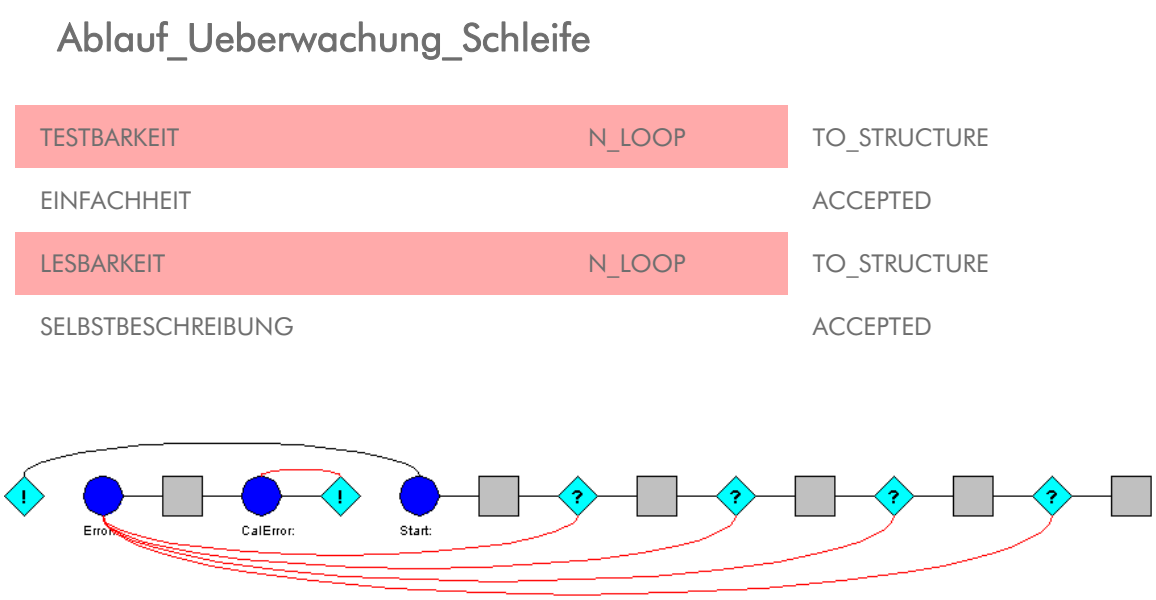

Abbildung 3: Beschreibung der Qualität eines Moduls und seines Kontrollflusses (rot markiert sind die verletzten Qualitätskriterien)

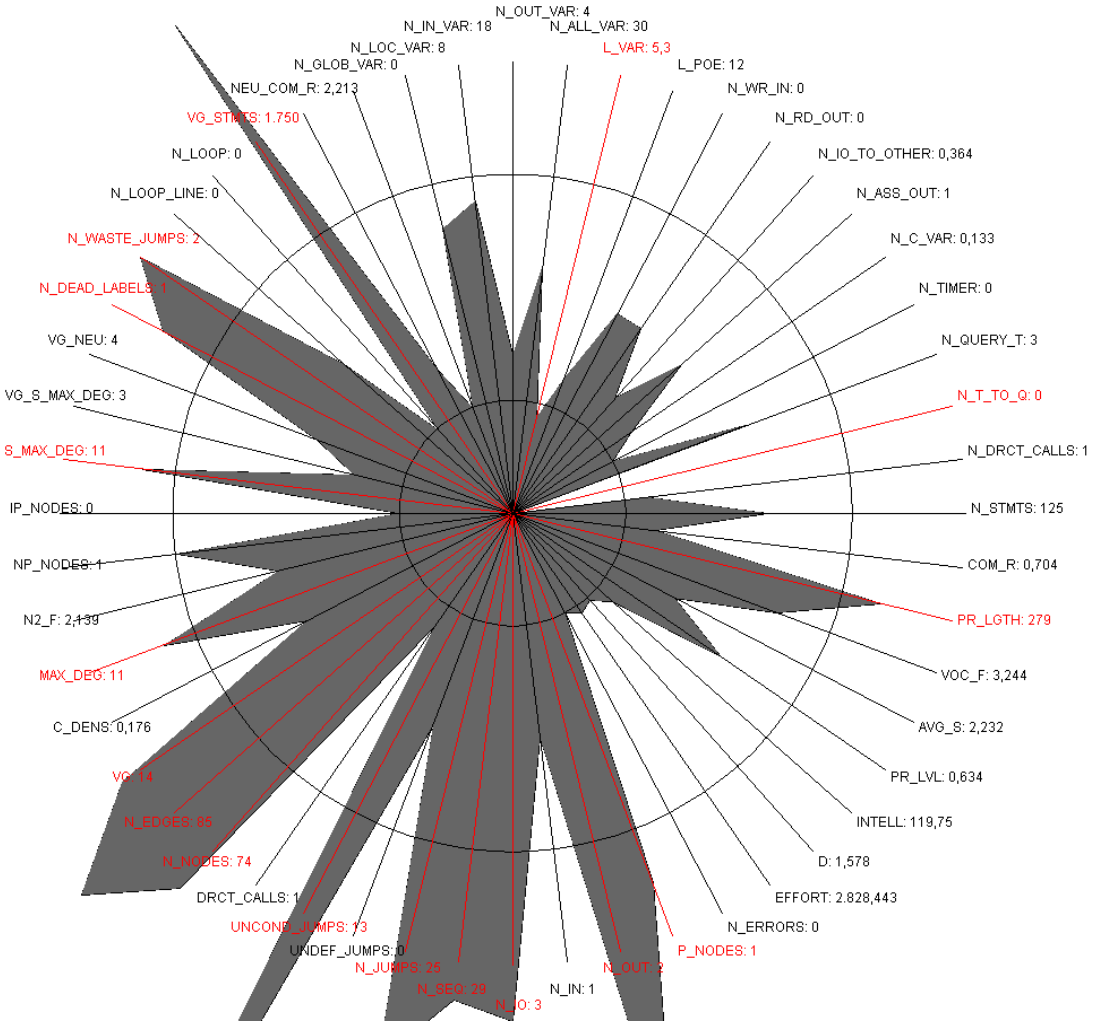

Abbildung 4: Kiviatgraph aller ermittelten Metriken (Verletzungen werden rot dargestellt)

Die im Internet herunterladbare Benutzerinformation zum Werkzeug beschreibt die notwendigen Anpassungen anhand zahlreicher Beispiele. Neben dem Analysewerkzeug für Anweisungsliste finden sich auf der oben genannten Internetseite weitere Analysetools. So ist es möglich, die Qualität sicherheitsrelevanter Software in C, in C++ und in Zukunft auch in JAVA zu analysieren. In Planung befindet sich zurzeit ein Analysewerkzeug für Assemblerprogramme. Da jedes der Werkzeuge auf einem als Freeware verfügbaren Parser aufsetzt, ist es im Rahmen einer Bachelor-Abschlussarbeit auf jede andere Hochsprache anpassbar. Alle Werkzeuge werden der Industrie über die oben genannte Seite kostenfrei zur Verfügung gestellt. Die angefertigten Abschlussarbeiten finden sich auch auf folgender Seite der Fachhochschule Bonn/Rhein-Sieg (www.inf.fh-bonn-rhein-sieg.de/reinert.html). Hinweise, die zur Verbesserung der Werkzeuge führen, werden von den Autoren gerne entgegen genommen und im Rahmen weiterer Praxisprojekte umgesetzt.

7. Literaturverweise:

[1] Bell, R.; Reinert, D.: Risk and system integrity concepts for safety-related control systems. Microprocessors and Microsystems 17 (1993) Nr. 1, S. 3-15.

[2] Balzert, H.: Lehrbuch der Software-Technik. Spektrum Akademischer Verlag. Heidelberg 1998.

[3] Staron, Ch.: Entwicklung eines Analysewerkzeugs zur Ermittlung von Metriken und Qualitätskriterien sicherheitsrelevanter Software im Maschinenschutz. Bachelor-Thesis. Fachbereich Informatik. Fachhochschule Bonn-Rhein-Sieg. Sankt Augustin 2004.

[4] Breuer, T.: Konzeption und Entwicklung eines PC-gestützten Werkzeugs zur Bestimmung von Codemetriken und Qualitätskriterien für sicherheitsrelevante Software für speicherprogrammierbare Steuerungen. Bachelor-Thesis. Fachbereich Informatik. Fachhochschule Bonn-Rhein-Sieg. Sankt Augustin 2006.

[5] Maurer, V.: Entwicklung eines Werkzeuges zur Bestimmung von Qualitätskriterien und Metriken für sicherheitsrelevante Industriesoftware in der Hochsprache C. Bachelor-Thesis. Fachbereich Informatik. Fachhochschule Bonn-Rhein-Sieg. Sankt Augustin 2005.

[6] Paurat, D.: Weiterentwicklung eines PC-gestützten Werkzeugs zur Bestimmung von Qualitätskriterien sicherheitsrelevanter C++ Software. Bachelor-Thesis. Fachbereich Informatik. Fachhochschule Bonn-Rhein-Sieg. Sankt Augustin 2007.

[7] Ley, M.: Entwicklung eines PC-gestützten Werkzeuges zur Bestimmung von Qualitätskriterien sicherheitsrelevanter Java Programme. Diplomarbeit. Fachbereich Informatik. Fachhochschule Bonn-Rhein-Sieg. Sankt Augustin 2005.

[8] Halstead, M. H.: Elements of Software Science. Prentice-Hall. New York. 1977.

[9] McCabe, T. J.: A Complexity Measure. IEEE Transactions of Software Engineering 2 (1976)4. S. 308-320.

[10] Henry, S.; Kafura, D.: The evaluation of software systems´ structure using quantitative software metrics. Software – Practice and Experience, 14 (1984)6 S. 561-573.

[11] Lorenz, M.; Kidd, J.: Object-Oriented Software-Metrics. Prentice Hall. 1994.

[12] NASA Software Assurance Technology Center. http://satc.gsfc.nasa.gov/metrics/index.html 15.8.2007.

[13] Verilog: Software development and testing using Logiscope. Toulouse 1992.

[14] Krell, M.: Bestimmung von Qualitätskiterien für sicherheitrelevante Software im Maschinenschutz auf Basis zertifizierter Industrieanwendungen, Diplomarbeit, Fachbereich Informatik, Fachhochschule Bonn-Rhein-Sieg. St. Augustin, 2003.