PROGRAMMIER-RICHTLINIE

Die Programmier-Richtlinie beschreibt die *organisatorische Vorgehensweise,* und die *einzuhaltenden Regeln* bei der Programmierung von Methoden im Rahmen von FVV-Forschungsvorhaben. Ziel dieser Richtlinie ist es, eine breite Nutzbarkeit und Qualität von erarbeiteten Methoden sicher zu stellen.

Inhalt

Präambel

Forschungsprojekte der FVV dienen dem wissenschaftlichen Erkenntnisgewinn, der häufig zu neuen Modellen, Gleichungen oder Vorgehensweisen führt. Es ist meist sinnvoll, diesen Erkenntnisgewinn (Modelle, Gleichungen, ...) nicht nur im Abschlussbericht, sondern auch in Form von eindeutigem Quellcode zu dokumentieren. Dieser Quellcode wird daher auch Methodenträger genannt. Bei diesem Quellcode handelt es sich nicht um Software im Sinne eines für den Endanwender einfach nutzbaren Produktes. Vielmehr geht es um eine eindeutige Dokumentation und Beschreibung der gewonnen Erkenntnisse in Form von einem programmierten Quelltext.

Einerseits werden die Forschungsstellen entlastet, wenn sie keine Software im Sinne eines Produktes erstellen müssen (keine Benutzeroberflächen-Entwicklung, Lauffähigkeit muss nur für eine Basisplattform gewährleistet sein, etc.). Andererseits müssen dadurch sehr hohe Qualitätsansprüche an den Quellcode gestellt werden, damit dieser problemlos von Mitgliedsfirmen in kommerzielle Produkte oder hauseigene Software integriert werden kann. Bei einer derartigen Integration des Quellcodes in eigene Software wird dieser praktisch immer angepasst werden müssen. Deshalb muss er in einer Art und Weise programmiert sein, die es nach Projektende einem an dem Forschungsvorhaben bis dahin unbeteiligten Software-Entwickler eines FVV-Mitglieds ermöglicht, problemlos den Code zu verstehen, ihn in eigene Software zu übernehmen, und die Ergebnisse gegen Referenz-Ergebnisse des FVV-Vorhabens zu validieren.

Die innerhalb der FVV laufenden Forschungsvorhaben sind zu unterschiedlich und vielfältig, als dass eine allgemeine und bindende Festlegung bezüglich der Basisplattform, d.h. insbesondere der Programmiersprache, sinnvoll wäre. Die Basisplattform incl. der Programmiersprache ist für jedes Projekt im Diskussionskreis festzulegen. Hierbei sind die konkreten Anforderungen des Projektes, vorangegangene FVV-Arbeiten, Synergieeffekte mit laufenden FVV-Arbeiten, eine langfristige Verwendbarkeit des Methodenträgers aber auch der technische Fortschritt in der Softwareentwicklung zu berücksichtigen.

0. Anwendungsbereich

Die Anwendung der Programmierrichtlinie ist verpflichtend, wenn die Entwicklung von Methodenträgern konkretes Ziel des Forschungsvorhabens ist.

Auch wenn die Entwicklung der Methodenträger nur ein notwendiger Schritt zur Zielerreichung des Projektziels ist oder es sich erst nach Projektbeginn zeigt, dass ein Methodenträger erarbeitet wird, ist die Anwendung der Programmierrichtlinie verpflichtend. Der projektbegleitende Ausschuss (PA) und die Forschungsstelle(n) können in Absprache selbstständig über die Randbedingungen der Methodenentwicklung im Rahmen der Programmier-Richtlinie entscheiden.

1. Ziele der Richtlinie

Die Ziele sind die Gewährleistung eines definierten Qualitätsstandards und die Erhöhung der Nutzbarkeit der Methodenträger.

Der Qualitätsstandard ist definiert durch

- Fehlerfreiheit des Codes
- Absicherung der Lauffähigkeit des Methodenträgers auf der/den mit dem PA vereinbarten Basisplattform/-en
- Portierbarkeit des Codes des Methodenträgers auf andere Basisplattformen
- Vorhandensein einer Eingabe- und Ausgabebeschreibung bzw. Benutzeranleitung
- Integrierte Überprüfung der Eingabedaten
- Ausreichende Kommentierung des Codes
- Einhaltung aller weiteren Anforderungen der Programmier-Richtlinie

2. Organisatorische Vorgehensweise

Die bei der Bearbeitung eines Forschungsthemas entstehenden Methodenträger sind Teil des Forschungsergebnisses. Sie werden von der Forschungsstelle erstellt, die verantwortlich ist für

- die zugrunde liegenden physikalisch- mathematischen Modelle und ihre Codierung
- den generellen Programmablauf
- die Vollständigkeit des Methodenträgers inkl. Validierungsdatensatz (Eingaben und Ausgaben) und Dokumentation

Obmann und Forschungsstelle werden bei der Auslegung der Richtlinie durch die FVV Geschäftsstelle unterstützt.

In Abstimmung mit dem PA ist es Aufgabe des Obmannes, die Koordination der Aktivitäten der Forschungsstelle durchzuführen. Die Forschungsstelle muss spätestens ein halbes Jahr vor dem Ende eines Vorhabens einen Vorab-Stand der dann existierenden Methodenträger an den zuständigen projektbegleitenden Ausschuss übergeben. Zweck dieser Vorab-Version des Methodenträgers ist eine grundsätzliche Abstimmung, ob sich die Forschungsstelle mit Programmierstil und Dokumentation auf dem gewünschten Weg befindet. Zu diesem Zeitpunkt soll der PA der Forschungsstelle Rückmeldung zur Einhaltung der Programmierrichtlinie und zum Umfang des Methodenträgers geben.

Der PA-Obmann wird das Vorhaben erst dann für erfolgreich abgeschlossen erklären, wenn der projektbegleitende Ausschuss den erreichten Stand des Methodenträgers inkl. Validierungsdatensatz und Dokumentation für ausreichend erklärt hat.

3. Umfang der abzuliefernden Ergebnisse zum Methodenträger

Die im Forschungsvorhaben entwickelten Methoden sind dem PA und der FVV in folgenden Umgang zu übergeben:

- Methodenträger
  - Vollständiger Quellcode
  - Lauffähige Version
- Validierungsdaten (Eingabe und zugehörige Ergebnisdaten in maschinenlesbarer Form)
- Übergabeworkshop
- Programmdokumentation inkl. Schnittstellen-Definition (siehe Kap 5)
- Beschreibung der Entwicklungsumgebung inklusive Compilereinstellungen (sofern zutreffend)
- Beschreibung der Basisplattform (Version und Konfiguration)
- Benutzerhandbuch

Die Überprüfung und Freigabe der Unterlagen/Ergebnisse erfolgt durch den PA.
Die Ergebnisse werden nach Abschluss des Projektes gemeinsam mit dem Abschlussbericht auf der FVV-Plattform zur Verfügung gestellt. Die Forschungsstelle sichert zu, dass der Methodenträger frei von Rechten Dritter ist.

4. Anforderungen an die Programmierung

Die Regeln, nach denen neue Methodenträger zu codieren sind, werden durch die zuständigen FVV- Gremien festgelegt. Sie werden in dieser Richtlinie beschrieben.

Die Regeln gliedern sich in Anforderungen an

- den Programmumfang
- die Programmstruktur
- die Laufeigenschaften des Programms
- die Implementierbarkeit des Programms auf anderen Basisplattformen
- die Lesbarkeit des Codes
- die Einzelheiten der Codierung
- die etwaige Verwendung von externen Bibliotheksunterprogrammen
- Freiheit von Rechten Dritter
- die Art und den Umfang der Dokumentation
- die etwaigen Besonderheiten der Aufgabenstellung

Für die Einhaltung der Regeln sind die Forschungsstelle und der zuständige „Projektbegleitende Ausschuss“ gemeinsam verantwortlich. Sollte im Einzelfall ein Problem entstehen, das nicht durch das Regelwerk abgedeckt ist, werden sich die Forschungsstelle und der „Projektbegleitende Ausschuss“ unter Leitung des Obmanns auf eine Lösung einigen. In diesen Fällen müssen die FVV Geschäftsstelle und die relevanten FVV Gremien über Probleme mit der Programmierrichtlinie und die vereinbarten Lösungen informiert werden.

Im Einzelnen sind hierbei die folgenden Punkte zu beachten:

- Die Übernahme der Programme (Methodenträger) in eine andere Basisplattform soll grundsätzlich ohne größere Eingriffe möglich sein. (Anforderungen an die Implementierbarkeit).
- Der Aufbau der Programme (Methodenträger) soll so sein, dass der Benutzer ganze Programmabschnitte ersetzen bzw. entnehmen kann. (Anforderungen an die Programmstruktur).
- Das Programm soll leicht verständlich geschrieben sein, so dass Einzelheiten ohne Nachfragen beim Ersteller geklärt bzw. geändert werden können. (Anforderungen an die Lesbarkeit des Codings).
- In diesem Zusammenhang wird die Verwendung englischer Kommentare und Bezeichner im Quelltext empfohlen, um das Forschungsergebnis den internationalen Mitgliedern der FVV zugänglich zu machen. Die zu verwendende Sprache wird zu

Beginn des Projekts im Einvernehmen von Obmann, PA und Forschungsstelle beschlossen.

- Jeder Methodenträger, der im Rahmen einer Forschungsaufgabe der FVV entwickelt wird, ist ausreichend zu dokumentieren.

  Dazu gehören:

  - Programm-Kurzbeschreibung (vgl. Abschnitt 5.1)
  - Benutzeranleitung
  - Quellprogramm
  - Programmunterlagen
  - Implementierungshinweise
  - Forschungsbericht mit der Darstellung der technisch-physikalischen Grundlagen sowie der mathematischen Methoden.

4.1 Basisplattform

Definition:

- Die Basisplattform ist notwendig und hinreichend für Ausführung des Methodenträgers, d.h. sie ermöglicht seine Ausführung und stellt alle erforderlichen Voraussetzungen für die Arbeiten am Methodenträger zur Verfügung
- Die Basisplattform existiert bereits zum Projektstart. Dies schließt einen späteren Wechsel auf eine neuere Version der Basisplattform im Einvernehmen zwischen PA und Forschungsstelle nicht aus.
- Verfügbarkeit vorzugsweise für die Forschungsstellen und für alle FVV Mitglieder, wenigstens aber für die Mitglieder des Projektbegleitenden Ausschuss soll gegeben sein. Steht im Einzelfall die Basisplattform nicht allen Mitgliedern des PA zur Verfügung, soll die Möglichkeit geprüft werden, eine weitere Basisplattform parallel zu unterstützen. Damit wird zugleich die Portierbarkeit des Methodenträgers abgesichert.
- Alles was nicht Basisplattform ist, ist Methodenträger und muss als Sourcecode ausgeliefert werden

Beispiele für Basisplattformen:

- Compiler, Interpreter, Linker, Bibliotheken
- Softwarepakete mit Hochsprachen-Programmierschnittstelle (z.B. zur Erstellung von Submodellen)
- Softwarepakete mit visueller Programmierschnittstelle (z.B. zur Funktionsentwicklung für Motorsteuergeräte oder zur Abbildung eines physikalischen Zusammenhangs in dieser grafischen Darstellung)

Die Basisplattform wird vor Projektbeginn gemeinsam mit dem DK oder PA festgelegt. Bei der Auswahl soll berücksichtigt werden, dass:

- die Erreichung der Ziele des Vorhabens durch die Basisplattform unterstützt wird. Dies umfasst auch, dass keine undefinierten Wechselwirkungen zwischen Basisplattform und Methodenträger zu erwarten sind.
- Synergien zu laufenden bzw. abgeschlossenen Vorhaben genutzt werden. FVV Programme wurden in der Vergangenheit in Fortran und ANSI C erstellt. Erweiterungen in diesen Programmen sind vorzugsweise in diesen Sprachen zu erstellen.
- eine langfristige Verwendbarkeit des Methodenträgers sichergestellt ist. (Robustheit gegenüber Versionswechsel)
- dem technischen Fortschritt der Programmierung Rechnung getragen wird.

Sofern zu Projektbeginn die Auswahl auf die Programmiersprachen Fortran oder C fällt, sollte in den Programmiersprachen Fortran 95 oder ANSI C nach ISO/IEC 9899 programmiert werden. Grundsätzlich sollen Programme, gleich in welcher Hochsprache sie verfasst werden, nach gültigen ANSI Standards codiert werden.

Eine Trennung von Programmfunktion und Benutzeroberfläche ist in jedem Fall durchzuführen.

Bibliotheksmodule, welche die Forschungsstelle benötigt, um die Lauffähigkeit des Methodenträgers zu gewährleisten, zählen entweder zur Basisplattform und unterliegen denselben Qualitätsansprüchen, oder sind als Teil des Methodenträgers im Quellcode beizustellen. Sofern diese Bibliotheksmodule als Teil der Basisplattform gelten sollen, ist dies vor Vergabe des Vorhabens im DK festzulegen und im Beiblatt zu dokumentieren.

4.2 Anforderungen an die Laufeigenschaften des Methodenträgers.

- Vor der eigentlichen Verarbeitung werden die Eingaben auf Vollständigkeit, Plausibilität und Überschreitung der gegebenenfalls fest dimensionierten Speicherbereiche geprüft.
- Das Programm ist gegen Abbrüche durch nicht definierte Operationen (z.B. Division durch Null, Wurzel aus negativer Zahl) abzusichern
- Iterationsschleifen werden nicht nur durch eine Genauigkeitsschranke, sondern zusätzlich durch einen Zähler abgeschlossen. Schranke und Zähler können programmseitig definiert sein und/oder mittels Eingaben variiert werden. Bei Erreichen der Iterationsgrenze ohne Konvergenz ist eine Information für den Anwender vorzusehen.
- Bei einem normalen Programmablauf sind keine Interaktionen (Meldungen, Schalterabfragen) und Pausen zugelassen.
- Der Datentransfer zwischen unterschiedlichen Programmen eines Programmpakets ist formatiert oder in binärer Form zulässig. Bei Verwendung von Binärdateien ist ein Konverter vom Binär- zum 7-Bit-ASCII- Format mit in das Programmpaket einzufügen.

- Zusätzlich für die Weiterverarbeitung (z. B. Grafik) erforderliche Ergebnisdaten sind wahlweise als Binär- oder 7Bit- ASCII- Datei auszugeben.
- Durch Eliminierung von Grafik- Programmen darf die Funktion des Berechnungsprogramms nicht beeinträchtigt werden.

4.3 Anforderungen an die Implementierung der Programme

- Jeder arithmetische Ausdruck darf nur in einem 'Mode' (also nur 'Real' oder 'Integer') codiert werden. Für Typumwandlungen ('Real' _ 'Integer' usw.) sind die genormten Konvertierungs- Funktionen zu verwenden.
- Alle Variablen sollten möglichst in SI-Einheiten vorliegen. Abweichungen sind möglichst im Variablennamen (z.B. p_Zyl_bar) oder als Kommentar kenntlich zu machen.
- Bei Fortran sind grundsätzlich alle Variablen über IMPLICITE NONE zu deklarieren
- Alle Variablen sind im darstellbaren Wertebereich und in der Auflösung zu spezifizieren.
- Die Parameter der Unterprogramm-Aufrufe sind, streng nach 'Eingabe' und 'Ausgabe' zu trennen.
- Die Datenfeldgrößen müssen an zentraler Stelle im Programm leicht zu ändern sein, dynamisches Memory-Management ist anzustreben.

4.4 Anforderungen an die Programmstruktur

- Das Hauptprogramm enthält im Wesentlichen nur Unterprogramm-Aufrufe. Globale Datenstrukturen sind möglichst zu vermeiden.
- Falls in Ausnahmesituationen compilerspezifische Funktionen verwendet werden müssen, sind sie in einem Modul des Programmpaketes zusammenzufassen und zu dokumentieren. Gleiches gilt für Funktionsaufrufe, wie z.B. Differentialgleichungslöser o.ä. aus kommerziellen Softwarepaketen, die als Basisplattform genutzt werden.
- Ein Unterprogramm sollte möglichst nicht mehr als 200 ausführbare Anweisungen enthalten.
- Die Möglichkeiten einer übersichtlichen Programmgestaltung und Datenübergabe zwischen Programmteilen mittels strukturierter Datentypen sind weitgehend zu nutzen.
- Eingabe, Ausgabe sowie Grafikaufrufe sind jeweils in getrennten Unterprogrammen durchzuführen.
- Fehlermeldungen enthalten eine kurze Beschreibung des Fehlers und den Namen des ihn verursachenden Unterprogramms
- Beim Entwurf der Programmstruktur ist die Möglichkeit zur Parallelisierung zu berücksichtigen.
- Abhängig vom Anwendungsgebiet (.z.B. FEM, CFD) sollte von Anfang an die Implementierung mit 64 bit Adressierung angestrebt werden.

- Für Fortran gilt grundsätzlich, dass
    - die ab der Version Fortran 95 überholten Sprachmittel nicht mehr zulässig sind. Analog zu der vorstehenden Aussage hinsichtlich der Verwendung globaler Datenstrukturen sind in Fortran COMMON-Blöcke nur in besonders begründeten Ausnahmefällen und in Absprache mit dem PA zulässig. Die Belegung der COMMON-Blöcke ist per INCLUDE Anweisung in alle betroffenen Unterprogrammen einheitlich vorzunehmen.
    - Es ist ein linearer Programmablauf anzustreben. Bevorzugt sind BLOCK IF-Anweisungen oder SELECT CASE-Konstrukte zu verwenden. Assigned GOTO-, computed GOTO- und arithmetische IF-Anweisungen sind nicht zugelassen.
    - In Fortran sind benannte Schleifen von der Struktur „DO … END DO“ zu verwenden

4.5 Anforderungen an Quellcode und Lesbarkeit

Die Programme sollen selbstdokumentierend codiert werden, d. h. sie sollen so strukturiert und mit so viele Kommentaren versehen sein, dass sich ein fremder, mit Bericht und Literatur vertrauter Leser ohne weitere Hilfsmittel und ohne Rückfragen beim Ersteller zurechtfinden kann.

4.5.1 Namensgebung

Die richtige Namensgebung ist das entscheidendste Element beim Schreiben von Quellcode. Prägnante und selbstsprechende Namen sind entscheidend für ein intuitives Verständnis des Quellcodes. Eine gute Namenswahl ist ein kreativer Prozess, der sich in diesem Dokument nicht normieren lässt, da er immer dem Kontext angepasst sein muss.

Es soll hier keine Namensgebungs-Konvention festgeschrieben werden, wichtig ist jedoch, dass innerhalb des Projektes eine einheitliche, allgemein verständliche und sinnvolle Namensgebung erfolgt. Die folgende Konvention ist als Vorschlag für die Forschungsstelle gedacht. Wenn die Forschungsstelle eine andere Namensgebungskonvention nutzen möchte, muss sie dies in einer der ersten Sitzungen gegenüber dem projektbegleitenden Ausschuss erklären und ihre eigene Konvention schriftlich dokumentieren.

Eine ungarische Notation (Typ-Präfix) wird nicht angewandt. Eine Ausnahme von dieser Regel bilden Namen von Pointern.

Zusammengesetzte Bezeichner werden in Kamelbuckelnotation geschrieben, d.h. die einzelnen Wörter innerhalb des Gesamtwortes beginnen jeweils mit einem Grossbuchstaben. (Bsp: getnextsample wird zu getNextSample). Für die Gross-/Kleinschreibung des ersten Buchstabens der Bezeichner gelten separate Vorschriften (siehe unten).

Alternativ zur Kamelbuckelnotation können in Anlehnung an TEX-Notation Unterstriche zur Darstellung von Subscripts verwendet werden.

Benennung von Methoden

Namen von Methoden und Funktionen beginnen immer mit einem Verb. Der erste Buchstabe ist daher (in Anlehnung an die Deutsche Sprache) ein Kleinbuchstabe. Es wird kein Präfix vorangestellt (Bsp: getNextSample()).

Methodennamen sollen genau aussagen, was die Methode tut. Deshalb soll im Namen nach dem einleitenden Verb ein zweiter Teil stehen, der beschreibt, was von dem „Machen“ betroffen ist. Dies lässt sich nur erreichen, wenn jede Methode genau eine Aufgabe wahrnimmt und nicht zwei, drei oder noch mehr.

Benennung von Konstanten

Namen von Konstanten bestehen nur aus Grossbuchstaben. Da hier eine optische Worttrennung durch Kamelbuckelnotation nicht möglich ist, sollen die Wortteile durch ein „_“ (Underline) Zeichen getrennt werden (Bsp: MAX_CYLINDER_COUNT).

Benennung von Variablen

Namen von Variablen/Instanzen/Objekten beginnen im Normalfall mit einem Nomen. Der erste Buchstabe ist daher (in Anlehnung an die Deutsche Sprache) ein Grossbuchstabe. Es soll darauf geachtet werden, dass der Variablenname präzise über Inhalt oder Sinn und Zweck der Variable Auskunft gibt. Variablennamen wie „i“, „j“ oder „k“ sind als Schleifenindizes zulässig.

Boolesche Variablen sollen so bezeichnet werden, dass beim Lesen einer Struktur automatisch richtig interpretiert wird.

Variablen, die physikalische Größen wiedergeben und für die ein gängiges Symbol existiert (z.B. Druck p, Temperatur T) oder für die ein Symbol im Abschlussbericht des Vorhabens eingeführt wurde, sollten mit diesem Symbol bezeichnet werden. Hierdurch ergibt sich eine bessere Lesbarkeit von Formeln und ein einfacheres paralleles Lesen von Abschlussbericht und Quellcode. Groß- und Kleinschreibung bei solchen Symbolen ist zwingend zu beachten (z.B. Druck p und Leistung P; Zeit t und Temperatur T). Derartige gängige Symbole sollten nicht zweckfremd verwendet werden. Für eine bessere Lesbarkeit sollten Indizes von den Symbolen, die meist aus einem Buchstaben bestehen, mit einem Unterstrich abgetrennt werden (z.B. T_unburned, p_CylMotored). Nicht-SI-Einheiten müssen im Variablennamen gekennzeichnet werden, z.B. p_bar.

Beispiel:

```
Deklaration: bool   IsValid;
             bool   HasNewValuesReceived;
             real*8    p_CylinderMaxFired; /* max. cylinder pressure during combustion in
Pa*/
             real*8    p_CylinderMaxMotored; /* max. cylinder pressure w/o combustion in
Pa*/
             real*8    FiringPressureRatio; /* pressure ratio fired/motored */

im Code:     if( IsValid ) {....}
             if( HasNewValuesReceived ) {....}
             FiringPressureRatio= p_CylinderMaxFired/ p_CylinderMaxMotored;
```

4.5.2 Kommentare

- Die Programme sollen selbstdokumentierend codiert werden, d. h. es sollen so viele Kommentare eingefügt werden, dass sich ein fremder, mit Bericht und Literatur vertrauter Leser ohne weitere Hilfsmittel zurechtfinden kann.
- Jedes Programm-Modul enthält den standardisierten Programmkopf gemäß Formblatt „Standard- Programmkopf" (siehe Anlage 3, Beispiele für FORTRAN und C in Anlage 4 und 5) sowie eine „History“-Liste, die Auskunft über Code-Änderungen gibt.
- Von allen Variablen ist bei der Deklaration als Kommentar ihre Bedeutung / Verwendung und die physikalische Einheit anzugeben
- Generell wird die Verwendung von Dokumentations-Systemen wie Doxygen u.ä. empfohlen. In diesem Fall kann von der Form der Dokumentation der Variablendeklaration, wie sie der Vorlage im Formblatt „Standard- Programmkopf" dargestellt ist, abgewichen werden. Es ist dann bei der Deklaration der Variablen in Kommentaren für die entsprechenden Informationen für die automatische Übernahme in die Dokumentation zu achten.

4.5.3 Lesbarkeit

- Im Quelltext ist eine strukturierte Schreibweise („Einrücken“) anzustreben. Freiräume in der Struktur sind mit Leerzeichen und nicht mit TAB’s zu erzeugen.
- Grundsätzlich sollen Codezeilen nicht breiter als 80 Zeichen werden. Zeilen lassen sich dadurch, ohne abgeschnitten zu werden, auf Terminals oder auf Papier ausgeben. Zeilen dürfen länger werden, wenn dadurch die Lesbarkeit verbessert wird.
- Der Quellcode der Programme muss in 7 Bit ASCII vorliegen, wobei Umlaute nicht zulässig sind (z.B. Ä = Ae; Ö= Oe; Ü = Ue; ß = ss usw.). Dies gilt auch für Kommentare im Quelltext und in den Datensätzen, sofern dies Textdateien sind.

4.6 Sicherung der Qualität

Alle Methoden sollten fehlerfrei im Quellcode implementiert sein, insbesondere sollten Abschlussbericht und Quellcode keine Widersprüche enthalten. Die Forschungsstelle ist frei in der Wahl der Methoden, die sie zur Qualitätssicherung einsetzt. Zu Beginn des Projektes muss die Forschungsstelle gegenüber dem Arbeitskreis erklären, wann sie welche Maßnahmen zur Qualitätssicherung einsetzen möchte. Empfohlen werden Code-Reviews und Versionskontrollsysteme. Beim Einsatz von Code-Reviews ist in jedem Quellcode zu dokumentieren, wann und durch wen das Review stattgefunden hat.

5. Dokumentation von Methodenträgern

Alle nachstehenden Dokumentationen sind als separate Dateien gemeinsam mit der Abschlussdokumentation auf dem Prometa-Server abzulegen.

5.1 Programm-Kurzbeschreibung

Durch die Kurzbeschreibung soll sich der Anwender darüber informieren können, ob das Programm grundsätzlich zur Lösung seines fachlichen Problems geeignet ist. Die Beschreibung ist auf einem Formblatt gemäß anliegendem Beispiel abzufassen (siehe Formblatt Programm-Kurzbeschreibung, Anlage 8).

5.2 Anwender-Dokumentation (Benutzerhandbuch)

Für die Dokumentation des Methodenträgers sind die entsprechenden Vorlagen der FVV zu verwenden (z.B. Kurzbeschreibung, s.o.). Die Anwender-Dokumentation enthält alle wichtigen Informationen, die zur Anwendung eines Programms bzw. Methodenträgers notwendig sind. Insbesondere gehören zur Anwender-Dokumentation:

a) Hinweise zur Installation und zur Basisplattform

b) Eingabe

- Eingabeformular, sofern aus älteren FVV Programmen formatierte Text-Eingabedatensätze übernommen werden
- Erklärung der Eingabegrößen
- Physikalische Größen mit Einheitenbezeichnung und Wertebereich
- Steuergrößen mit Angabe ihrer Funktion
- Angaben über zulässige Bereiche der Eingabedaten
- Definitionen und Konventionen über Messgrößen und deren Aufbereitung.

Ggf. ist auf eine entsprechende Erklärung in Forschungsbericht oder Programmhandbuch zu verweisen.

c) Ausgabe

Erläuterung der Ausgabe z. B. für Bildschirm, Speichermedien und ggf. Weiterverarbeitung in der Basisplattform (siehe Kap. 5.4) sowie von Aufbau und Inhalt einer möglichen Ergebnisdatei. Dabei sind die ausgegebenen Größen hinsichtlich ihrer Bedeutung und technischen Einheit zu dokumentieren.

d) Fehlermeldungen, Restart

- Erläuterungen der Fehlermeldungen
- Beschreibung von Abhilfen
- Angaben zum Restart nach vorzeitigem Programmabbruch

e) Dateien

Art, Größe und Aufbau der verwendeten Dateien sind anzugeben.

f) Testbeispiele

An einem oder mehreren praktischen Anwendungsfällen sollen Ein- und Ausgabe veranschaulicht werden.
Dazu gehören:

- Rechnerspezifische Angaben (Hardware, Betriebssystem, Basisplattform, ggf. Compiler plus Optionen)
- Eingabedatensätze und Ergebnisdateien
- Bildschirm- bzw. Druckerausgabe und/oder grafische Ausgaben des Programms
- Rechenzeiten unter Angabe der verwendeten Hardware
- Speicherplatzbedarf (intern/extern)

5.3 Quellprogramm

Das vollständige Quellprogramm des Methodenträgers inklusive ggf. von der Forschungsstelle beizustellende Bibliotheksmodule sind unmittelbar nach Abschluss des Forschungsvorhabens auf dem FVV-Prometa-Server jederzeit abrufbereit zu deponieren. Die Verwaltung des gesamten Methodenträgers obliegt der FVV.

5.4 Programm-Dokumentation

Die Programm-Dokumentation muss so gehalten sein, dass Fehlerbeseitigung, Programmänderungen und -erweiterungen ohne besondere Schwierigkeiten möglich sind. Sie ist als Ergänzung zur Anwender- Dokumentation und zum Abschlußbericht zu verstehen.

Dazu gehören:

- Grobblockdiagramm (Anlage 1)
- Strukturbild (vgl. Anlage 6)
- Kurzbeschreibung der einzelnen Unterprogramme analog dem Programmkopf (vgl. Anlage 3, bzw. Anlagen 4 oder 5)
- Erklärung aller vereinbarten Variablen
- Datenflusspläne bei Programmpaketen

Alle Schnittstellen zu Basisplattformen (d.h. Fremdprogramme, Bibliotheken) sind so zu dokumentieren, dass eine Portierung auf beliebige andere Basisplattformen ermöglicht wird.

Dazu gehören

- Angaben zu den genutzten Funktionen der Basisplattform
- Argumentenliste
    - Einheiten und zulässige Wertebereiche von Eingangs- und Ausgangsgrößen
    - Interpretation der über die Schnittstelle übergebenen Größen
- Zeitpunkt und Art der Interaktionen zwischen Basisplattform und Methodenträger
- Besonderheiten der verwendeten Basisplattform (z.B. Workarounds bei Compilerfehlern)
- Ggf. Kriterien, die zur Auswahl der Basisplattform geführt haben.
- Ggf. bei mathematischen Verfahren die Beschreibung des mathematischen Verfahrens innerhalb der Basisplattform, sofern bekannt bzw. veröffentlicht.
- Ggf. Literaturhinweise zur Basisplattform

Bibliotheken, die nicht Teil der Basisplattform sind, müssen als Teil des Methodenträgers im Quellcode ausgehändigt werden. Sie sind in gleicher Weise zu dokumentieren wie der Methodenträger.

Für jedes physikalische Modell bzw. Unterprogramm ist ebenfalls eine Dokumentation zu erstellen, dies kann im Rahmen des Abschlussberichtes des Vorhabens geschehen.

- Schriftliche Aufbereitung
- Mathematische Ableitung und deren numerische Umsetzung
- Gültigkeitsbereiche und Randbedingungen
- Musteranwendungsbeispiel (Verifikationsmuster)
- Modellstrukturplan, d.h. Übersicht über weitere Untermodelle

6. Echtzeitprogramme

Bei echtzeitfähigen Programmen wird die Verwendung von ANSI C (ISO/IEC 9899) zur Programmierung empfohlen. Aufgrund der anders gearteten Struktur derartiger Softwarelösungen sind hardwarespezifische Erweiterungen notwendig und daher im Einzelfall mit dem jeweiligen projektbegleitenden Ausschuss genau abzustimmen.

Jeder manuell neu zu erstellende Programmcode ist nach dem „MISRA-C:2004 Guidelines for the Use of the C Language in Critical Systems“ zu implementieren. Dieser Standard basiert auf der Norm ISO/IEC 9899: Programming languages - C..

Für Autocodierung sollen die nachfolgenden Standards berücksichtigt werden:

- MISRA AC AGC - Guidelines for the application of MISRA-C:2004 in the context of automatic code generation
- MISRA AC GMG - Generic modelling design and style guidelines
- MISRA AC SLSF - Modelling design and style guidelines for the application of Simulink and Stateflow
- Sowie ggf. MISRA AC TL - Modelling style guidelines for the application of TargetLink in the context of automatic code generation

Die Einhaltung der MISRA-Standards soll nach Möglichkeit automatisch geprüft werden. Die Auswahl der geeigneten Tools hat projektbezogen im PA zu erfolgen. Da nicht alle MISRA-Regeln durch automatisch ablaufende Algorithmen überprüft werden können, ist es notwendig, einige dieser Regeln „von Hand“ zu überprüfen.

Um eine klare Zuordnung zu schaffen, ist zu Beginn eines neuen Projektes die *Compliance Matrix* zu erstellen, in der festgelegt wird, welche Regel mit welcher Methode geprüft wird. Falls nicht alle Regeln geprüft werden, ist dies in der Matrix eindeutig zu vermerken.

Das Programm muss nach Regeln des im Anhang befindlichen Style Guide (siehe Anlage 9) erstellt werden. Die Programmstruktur muss modular sein. Die Modularisierung hat nach den Regeln des Style Guide zu erfolgen. Die Kommentierung soll ebenfalls modular erfolgen.

Die vorstehenden Dokumentationsregeln bleiben auch für echtzeitfähige Programme erhalten, zusätzlich sind alle Funktionen in Zeitablaufplänen (siehe Anlage 7) darzustellen.

Sollte der C-Code mittels automatischer Codegenerierung erzeugt worden sein, können auch andere Diagrammarten zur Dokumentation des C-Code eingesetzt werden, wie zum Beispiel Signalflussdiagramme, Zustandsübergangsdiagramme und Flussdiagramme.

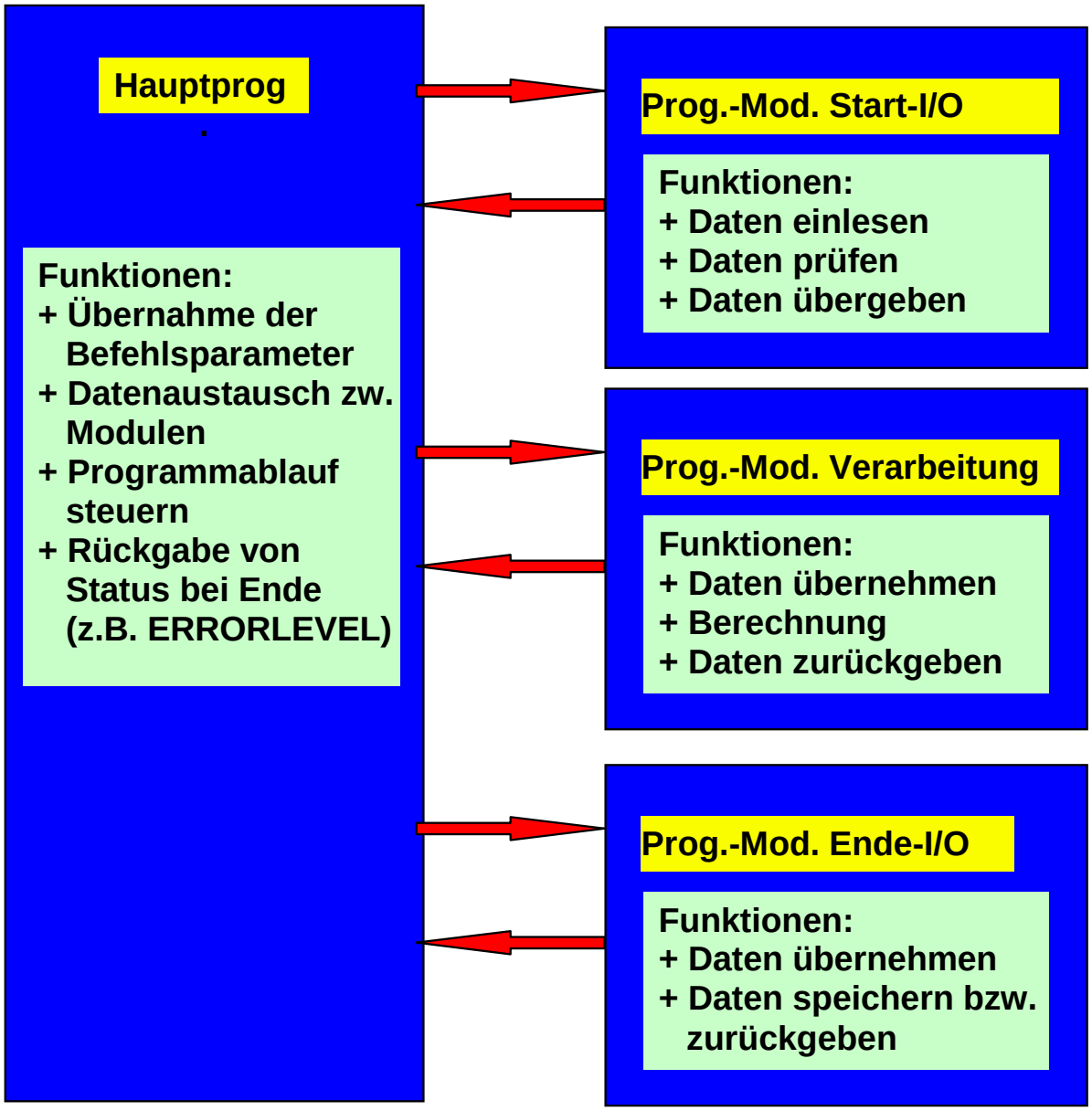
7. Abkürzungen

DK	Diskussionskreis
PA	Projektbegleitender Ausschuss

8. Glossar

Begriff	Definition
Methode	• Abstrakte Vorgehensweise / Algorithmus • Auch neuartige Kombination bewährter Techniken • Parameterableitung und/oder –identifikation aus Versuchsergebnissen.
Methodenträger	• Beispielanwendung • Dokumentierte Kern-Implementierung der Methode (keine GUI etc.) auf eine definierte Basisplattform
Basisplattform	• Die Basisplattform ist notwendig und hinreichend für Ausführung des Methodenträgers • Die Basisplattform existiert zum Projektstart • Verfügbar für Forschungsstellen und alle FVV Mitglieder, wenigstens PA Mitglieder • Bespiele für Basisplattformen sind: Excel, MATLAB, höhere Programmiersprachen, CAE-Software,… • Alles was nicht Basisplattform ist, ist Methodenträger und muss als Sourcecode ausgeliefert werden.
Softwareprodukt	• Allgemeingültige, parametrisierte Gesamtanwendung, incl.: – Methode – Datenmodellstruktur – Benutzerinterface (z.B. pre- postprocessor) – Produktdokumentation – Versionskontrolle – Fehlerbehandlungsmethoden – Portierungen – Regressionstests – Installationsprogramm – Ggf. Integration m. anderen Softwareprodukten – Zertifizierung – …
Softwarepflege	– Weiterentwicklung und Pflege bzgl.: – Methodischer Stand der Technik – Softwareperformance – Stabilität – Allg. Qualität (Bug-fixing) – Anpassung auf veränderliche Hardware und OS-Anforderungen
Software-Kommerzialisierung	kommerzieller Vertrieb und Service weltweit ausnahmslos im Rahmen der gesetzlichen Randbedingungen

Beispiel für einen modularen Programmaufbau



Beispiel für die Anordnung der DO-Schleifenlabel in Fortran:

```
DO L=1,LMAX
  DO J=1,JMAX
    A=Y(J,L)
    DO I=1,IMAX
      X(I,J,L)=Z(L)*A
    END DO
    DO I=1,IMAX
      T(I,J,L)=X(I,J,L)**2-A
    END DO
  END DO
  DO J=1,JMAX
    B=Y(J,L)**2
    C(J,L)=PI*B
    D(J,L)=28.5*(B-5.)
  END DO
END DO
```

Vorlage für die Kommentierung von Programm- / Unterprogrammköpfen

Erklärung der Programmfunktion in einem Satz	
Autor	Dipl.-Ing. P. Mustermann
Institut	Inst. für technische Rechentechnik
Universität	Techn. Uni. Klein Kleckersdorf
Datum	11.11.2011
Sprache	Fortran 95
Vorhaben	Kurbelgehäusebeleuchtung
FVV-Nr	4711

Code Review	Reviewer	Datum
Review I	Dipl-Ing. X. Fehlersucher,	16.02.1999
Review II	Prof. Dr.-Ing. Chef	28.02.1999

Variablenname	**Typ**	**Ein/Aus**	**Einheit**	**Erklärung der Größe**
A	Real*4	Ein	m**2	Wärmeübertragende Fläche des Kolbenbodens
R	Real*4	Ein	KJ/kg K	Gaskonstante
X	Real*4	Aus	m	Kolbenweg

History:			
Version	Datum	Autor	Kommentar
V 1.0	2004-07-07	P.Mustermann	Neu erstellt
V2.0	2007-11-30	F. Musterfrau, ITRT, Uni Klein Kleckersdorf	Formale Überarbeitung in FVV M815
…			

Implementierung nach:
- MERKER, Günter; SCHWARZ, Christian; STIESCH, Gunnar; OTTO, Frank: Verbrennungsmotoren. Simulation der Verbrennung u. Schadstoffbildung. 2. Aufl. Stuttgart: Teubner, 2004
- PISCHINGER, R. ; KLELL, M. ; SAMS, T.: Thermodynamik der Verbrennungskraftmaschine. 2. Aufl. Wien: Springer, 2002.

Ein Programmbeispiel in Fortran 95

```
!
! Unterprogramm zur Berechnung der Waermeuebertragung nach Woschni
!
! Autor:          Dipl.-Ing. P. Mustermann
! Institut:       Inst. fuer technische Rechentechnik
! Universitaet:   Techn. Universitaet Klein Kleckersdorf
! Datum:          2004-07-07
! Sprache:        Fortran 95
! Vorhaben:       Kurbelgehaeusebeleuchtung
! FVV-Nr.:        4711
! ------------------------------------------------------------------------
! Review I:       Dipl-Ing. X. Fehlersucher, 2004-07-09
! Review II:      Prof. Dr.-Ing. Chef, 2004-12-10
! ------------------------------------------------------------------------
! geaendert:      Dipl.-Ing. F. Musterfrau
! Datum:          2009-11-18
! Institut:       Inst. fuer technische Rechentechnik
! Universitaet:   Techn. Universitaet Klein Kleckersdorf
! Vorhaben:       Kurbelgehaeusebeleuchtung
! FVV-Nr.:        4711
!

! ------------------------------------------------------------------------
! Variable      Typ     Ein/Aus  Einheit  Bezeichnung
! ------------------------------------------------------------------------
! p             real*8    E     (Pa)      Druck im Zylinder
! T             real*8    E     (K)       mittl. Temp. im Zylinder
! p_ES          real*8    E     (Pa)      Druck im Zylinder bei
!                                         Einlass-Schliesst (ES)
! T_ES          real*8    E     (K)       Temp. im Zylinder bei ES
! V_ES          real*8    E     (m^3)     Vol. im Zyl. bei ES
! V_Hub         real*8    E     (m^3)     Hubvolumen des Zyl.
! d_Zyl         real*8    E     (m)       Durchmesser des Zylinders
! c_m           real*8    E     (m/sec)   mittlere Kolbengeschw.
! c_u_Drall     real*8    E     (m/sec)   Drallgeschwindigkeit
! p_Schlepp     real*8    E     (Pa)      Druck des Arbeitsgases
!                                         ohne Verbrennung
! isGasExchange logical   E     (-)       gibt an, ob Ladungswechsel
!                                         oder nicht
!
! alpha         real*8    A     (W/m^2/K) Waermeuebergangszahl
! isValid       logical   A               Fehlerflag  0: Fehler 1: i.O.
!

! History:
! --------
! Version   Datum      Name        Beschreibung
```

```
!  V 1.0    20040707    Mustermann  neu erstellt
!  V 1.1    20040709    Mustermann  Huber nur noch für ganze ASPs aktiviert
!  V 1.2    20041210    Mustermann  alpha_uv wird nicht mehr für leere Zonen
!                                    berechnet
!  V 1.45   20070615    Musterfrau  kleines internes Redesign (DataMain%geo)
!  V 1.52   20081216    Musterfrau  verbesserter Schwingungsschutz gegen An-
!                                    /Abschalten der Huber-Erweiterung
!  V 1.54   20091001    Musterfrau  saubere Fehlermeldung wenn alpha = NaN
!  V 1.55   20091118    Musterfrau  keine Abstuerze mehr durch extremen
!                                    Verbrennungsterm


! Implementierung nach:
!   - MERKER, Günter; SCHWARZ, Christian; STIESCH, Gunnar; OTTO, Frank:
!     Verbrennungsmotoren. Simulation der Verbrennung u. Schadstoffbildung.
!     2. Aufl. Stuttgart: Teubner, 2004
!   - PISCHINGER, R. ; KLELL, M. ; SAMS, T.: Thermodynamik der
!     Verbrennungskraftmaschine. 2. Aufl. Wien: Springer, 2002.


subroutine calcAlphaWoschni( p, T, p_ES, T_ES, V_ES, V_Hub, d_Zyl, c_m,
                             c_u_Drall, p_Schlepp, isGasExchange, alpha,
                             isValid )

implicit none

! Argumente
real*8, intent(in)    :: p           ! (Pa) Druck im Zylinder
real*8, intent (in)   :: T           ! (K) mittl. Temp. im Zylinder
real*8, intent (in)   :: p_ES        ! (Pa) Druck im Zylinder bei ES
real*8, intent (in)   :: T_ES        ! (K) Temp. im Zylinder bei ES
real*8, intent (in)   :: V_ES        ! (m^3) Vol. im Zyl. bei ES
real*8, intent (in)   :: V_Hub       ! (m^3) Hubvolumen des Zyl.
real*8, intent (in)   :: d_Zyl       ! (m) Durchmesser des Zylinders
real*8, intent (in)   :: c_m         ! (m/s) mittlere Kolbengeschw.
real*8, intent (in)   :: c_u_Drall   ! (m/s) Drallgeschwindigkeit
real*8, intent (in)   :: p_Schlepp   ! (Pa) geschleppter Druckverlauf
logical, intent (in)  :: isGasExchange ! gibt an, ob im Ladungswechsel
real*8, intent (out)  :: alpha       ! (W/m^2/K) Waermeuebergangszahl
logical, intent (out):: isValid      ! Fehlerflag  0: Fehler; 1: i.O.

! interne Variablen:
real*8 :: C1, C2                          ! Konstanten nach Woschni
real*8 :: p_bar, p_ES_bar, p_Schlepp_bar  ! (bar) umgerechnete Druecke
real*8 :: delta_p_bar                     ! (bar) Druckdifferenz gefeuert
real*8 :: zh1, zh2                        ! Hilfsvariablen


if (p > 0.0 .and. p_ES > 0.0) then  ! gueltiger Druck ?
  C2=3.24*0.001               ! Konst. C2
  if (isGasExchange) then
    C1=6.18+0.417*(c_u_Drall / c_m)   ! Konst. C1 fuer den Ladungswechsel
  else
```

```
    C1=2.28+0.308*(c_u_Drall / c_m)   ! Konst. C1 fuer Hochdruckphase
  end if

  ! Variablen mit nicht SI-konformen Einheiten sind deutlich
  ! kennzuzeichnen
  p_bar = p*1e-5
  p_ES_bar = p_ES*1e-5
  p_Schlepp_bar = p_Schlepp*1e-5

  zh1 = 130.0 * (d**(-0.2))*(p_bar**0.8)*(T**(-0.53))

  delta_p_bar = p_bar - p_Schlepp_bar
  if (delta_p_bar > 0.0) then      ! darf nicht negativ werden
    ! Verbrennungs-Term nach Woschni berechnen:
    zh2 = ( C1 * c_m + C2 * ((V_Hub * T_ES) / (p_ES_bar * V_ES)) *
                                                    delta_p_bar)**0.8
  else
    zh2 = ( C1 * c_m )**0.8
  endif

  alpha=zh1*zh2         ! (W/m^2/K) Waermeuebergangszahl

  isValid = .true.

else
  alpha = 0.0
  isValid = .false.           ! Fehler ungueltiger Druck
end if

end subroutine calcAlphaWoschni
```

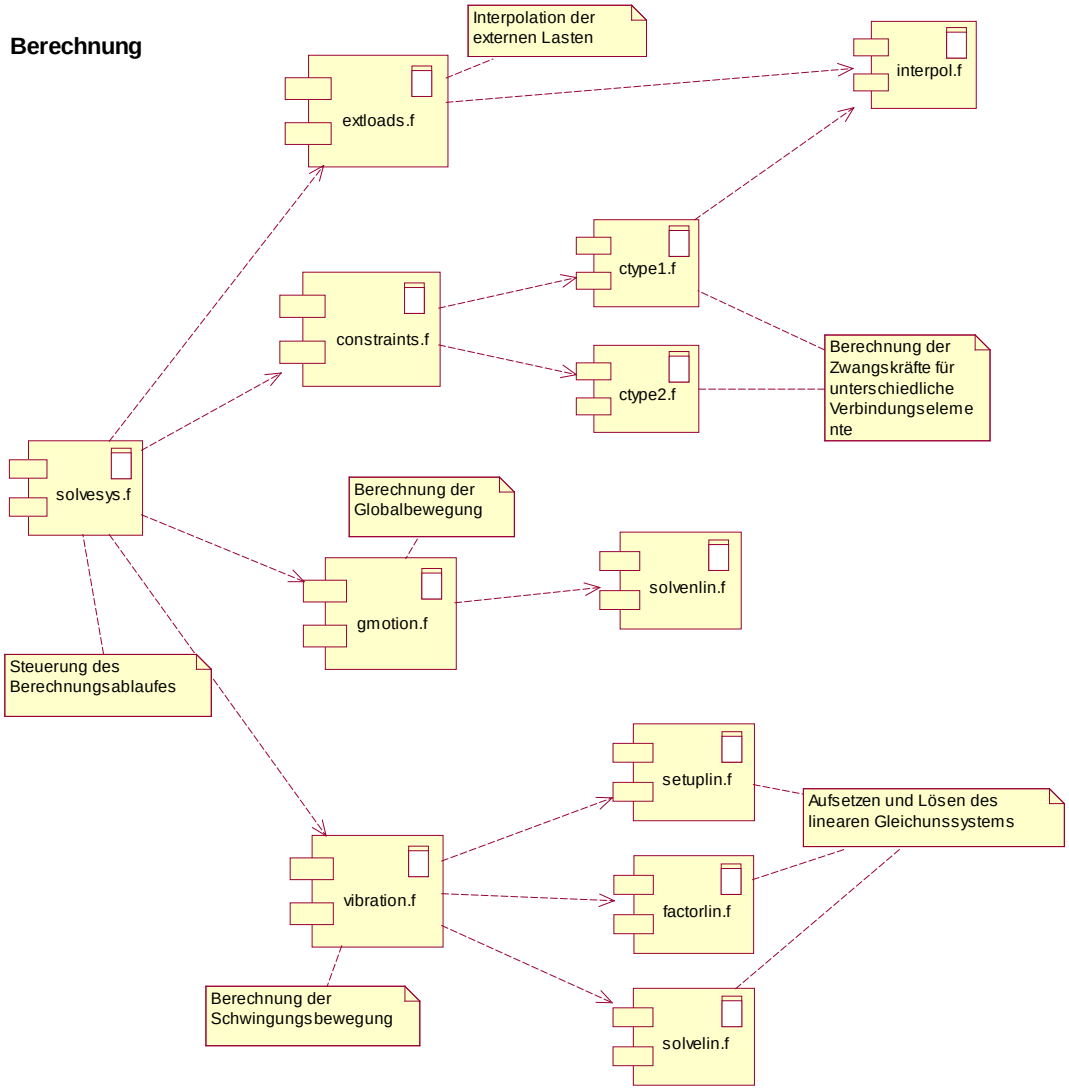
Ein Programmbeispiel in C

```
/*
 ------------------------------------------------------------------------

 Unterprogramm zur Berechnung der Waermeuebertragung nach Woschni

 ------------------------------------------------------------------------

 Autor:           Dipl.-Ing. P. Mustermann
 Institut:        Inst. fuer technische Rechentechnik
 Universitaet:    Techn. Universitaet Klein Kleckersdorf
 Datum:           2007-07-07
 Sprache:         ANSI C
 Vorhaben:        Kurbelgehaeusebeleuchtung
 FVV-Nr.:         4711
 ------------------------------------------------------------------------
 Review I:        Dipl-Ing. X. Fehlersucher, 2004-07-09
 Review II:       Prof. Dr.-Ing. Chef, 2004-12-10
 ------------------------------------------------------------------------
 geaendert:       Dipl.-Ing. F. Musterfrau
 Datum:           2009-11-18
 Institut:        Inst. fuer technische Rechentechnik
 Universitaet:    Techn. Universitaet Klein Kleckersdorf
 Vorhaben:        Kurbelgehaeusebeleuchtung
 FVV-Nr.:         4711


 ------------------------------------------------------------------------
 Variable      Typ      Ein/Aus  Einheit  Bezeichnung
 ------------------------------------------------------------------------
 p               real*8    E     (Pa)     Druck im Zylinder
 T               real*8    E     (K)      mittl. Temp. im Zylinder
 p_ES            real*8    E     (Pa)     Druck im Zylinder bei
                                          Einlass-Schliesst (ES)
 T_ES            real*8    E     (K)      Temp. im Zylinder bei ES
 V_ES            real*8    E     (m^3)    Vol. im Zyl. bei ES
 V_Hub           real*8    E     (m^3)    Hubvolumen des Zyl.
 d_Zyl           real*8    E     (m)      Durchmesser des Zylinders
 c_m             real*8    E     (m/sec)  mittlere Kolbengeschw.
 c_u_Drall       real*8    E     (m/sec)  Drallgeschwindigkeit
 p_Schlepp       real*8    E     (Pa)     Druck des Arbeitsgases
                                          ohne Verbrennung
 isGasExchange   logical   E     (-)      gibt an, ob Ladungswechsel
                                          oder nicht

 isValid         logical   A              Fehlerflag  0: Fehler 1: i.O.
```

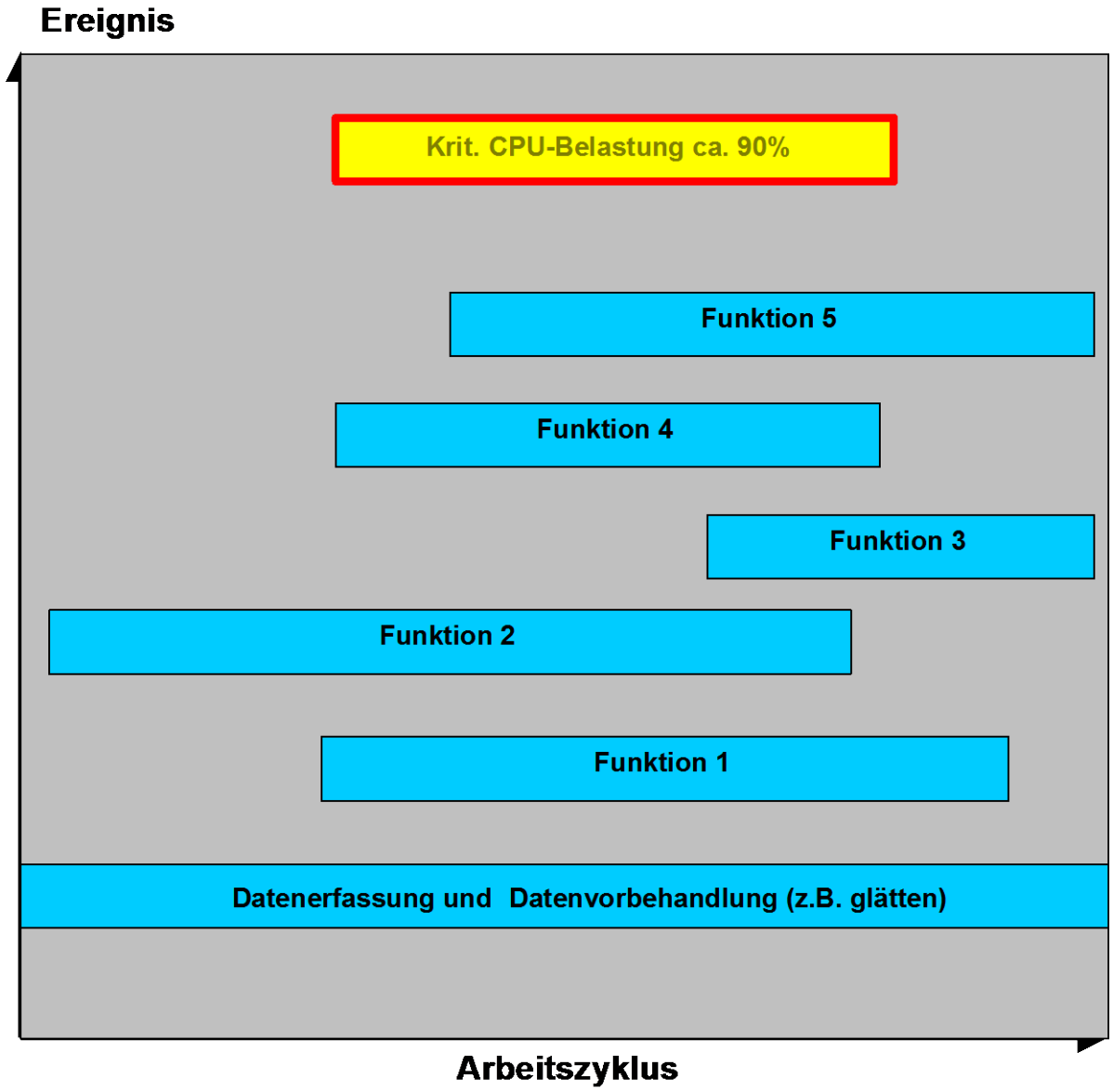
```
 History:
 --------
 Version    Datum       Name        Beschreibung
  V 1.0     20040707    Mustermann  neu erstellt
  V 1.1     20040709    Mustermann  Huber nur noch für ganze ASPs aktiviert
  V 1.2     20041210    Mustermann  alpha_uv wird nicht mehr für leere Zonen
                                     berechnet
  V 1.45    20070615    Musterfrau  kleines internes Redesign (DataMain%geo)
  V 1.52    20081216    Musterfrau  verbesserter Schwingungsschutz gegen An-
                                     /Abschalten der Huber-Erweiterung
  V 1.54    20091001    Musterfrau  saubere Fehlermeldung wenn alpha = NaN
  V 1.55    20091118    Musterfrau  keine Abstuerze mehr durch extremen
                                     Verbrennungsterm


 Implementierung nach:
   - MERKER, Günter; SCHWARZ, Christian; STIESCH, Gunnar; OTTO, Frank:
     Verbrennungsmotoren. Simulation der Verbrennung u. Schadstoffbildung.
     2. Aufl. Stuttgart: Teubner, 2004
   - PISCHINGER, R. ; KLELL, M. ; SAMS, T.: Thermodynamik der
     Verbrennungskraftmaschine. 2. Aufl. Wien: Springer, 2002.


*/

double calcAlphaWoschni( p, T, p_ES, T_ES, V_ES, V_Hub, d_Zyl, c_m,
                         c_u_Drall, p_Schlepp, isGasExchange,
                         isValid )

{

//interne Variablen:
double C1, C2;                              // Konstanten nach Woschni
double p_bar, p_ES_bar, p_Schlepp_bar;      // (bar) umgerechnete Druecke
double delta_p_bar;                         // (bar) Druckdifferenz gefeuert
double zh1, zh2;                            // Hilfsvariablen
double alpha;                               // (W/m^2/K Waermeuebergangszahl
```

```
if (p > 0.0 && p_ES > 0.0)    // gueltiger Druck ?
{
  C2=3.24*0.001;              // Konst. C2
  if (isGasExchange)
    /* Konst. C1 fuer den Ladungswechsel */
    C1=6.18+0.417*(c_u_Drall / c_m);
  else
    /* Konst. C1 fuer Hochdruckphase */
    C1=2.28+0.308*(c_u_Drall / c_m);

  /* Variablen mit nicht SI-konformen Einheiten sind deutlich
     zu kennzeichnen */
  p_bar = p*1e-5;
  p_ES_bar = p_ES*1e-5;
  p_Schlepp_bar = p_Schlepp*1e-5;

  zh1 = 130.0 * pow(d,-0.2) * pow(p_bar,0.8) * pow(T, -0.53);

  delta_p_bar = p_bar - p_Schlepp_bar;
  if (delta_p_bar > 0.0)       /* darf nicht negativ werden */
  {
    /* Verbrennungs-Term nach Woschni berechnen: */
    zh2 = pow( C1 * c_m + C2 * ((V_Hub * T_ES) / (p_ES_bar * V_ES)) *
                                                    delta_p_bar, 0.8 );
  }
  else
  {
    zh2 = pow( C1 * c_m, 0.8);
  }

  alpha=zh1*zh2;        /* (W/m^2/K) Waermeuebergangszahl */

  isValid = true;

}
else
{
  alpha = 0.0;
  isValid = false;            /* Fehler ungueltiger Druck */
}

return alpha ;

}
```

Beispiel für einen Programmstrukturplan



Beispiel für einen Zeitablaufplan



Programm-Kurzbeschreibung

Name des Programms	
Vorhaben-Kurztitel	
Vorhaben-Nr.	
Forschungsstelle	
Leiter der Forschungsstelle	
Autoren	
erstellt am	
Version	
überarbeitet von	
Zweck der Methode	
Schlagworte	
Programmiersprache	
Anforderungen an	
Systemumgebung	
Vorhandene Schnittstellen	
Wichtige Anmerkungen	

Echtzeitprogramme

Begriffserklärung – Echtzeitprogramme

1.1 Gruppe

SW-Entwurfseinheit mit abgegrenzter Funktionalität und definierten Schnittstellen, kann aus mehreren Komponenten bestehen (z. B. die Gruppe COM – beinhaltet alle Kommunikationskomponenten).

1.2 Komponente

SW-Entwurfseinheit mit abgegrenzter Funktionalität und definierten Schnittstellen innerhalb einer Gruppe (z. B. CCP, KWP).

1.3 Modul

Im programmiertechnischen Sinne eine Datei (auch Source-File genannt), die aus einer Reihe von Operationen (Funktionen), Typdeklarationen, Variablen und Konstanten bestehen kann. Jedes Modul wird physikalisch unter einem spezifischen Namen, der den Inhalt des Moduls am nächsten beschreibt, abgelegt. Somit stellt jedes Modul die kleinste kompilierbare Programmeinheit dar (siehe Anhang A zum Aufbau eines Moduls).
Um eine höhere Softwarequalität im Sinne von Wartbarkeit und Lesbarkeit zu erreichen, soll der Gesamtquellcode eines Programms modular aufgebaut werden, d. h. möglichst mehrere Module (Programmdateien), die eigenständig kompiliert werden können und jeweils einen bestimmten Sachverhalt beschreiben. Modulnamen sollen die Gesamtaufgabe aller Operationen wiedergeben, die in dem Modul enthalten sind. Einige Beispiele sind nachfolgend aufgeführt (Module sind C-Dateien).

Beispiel:

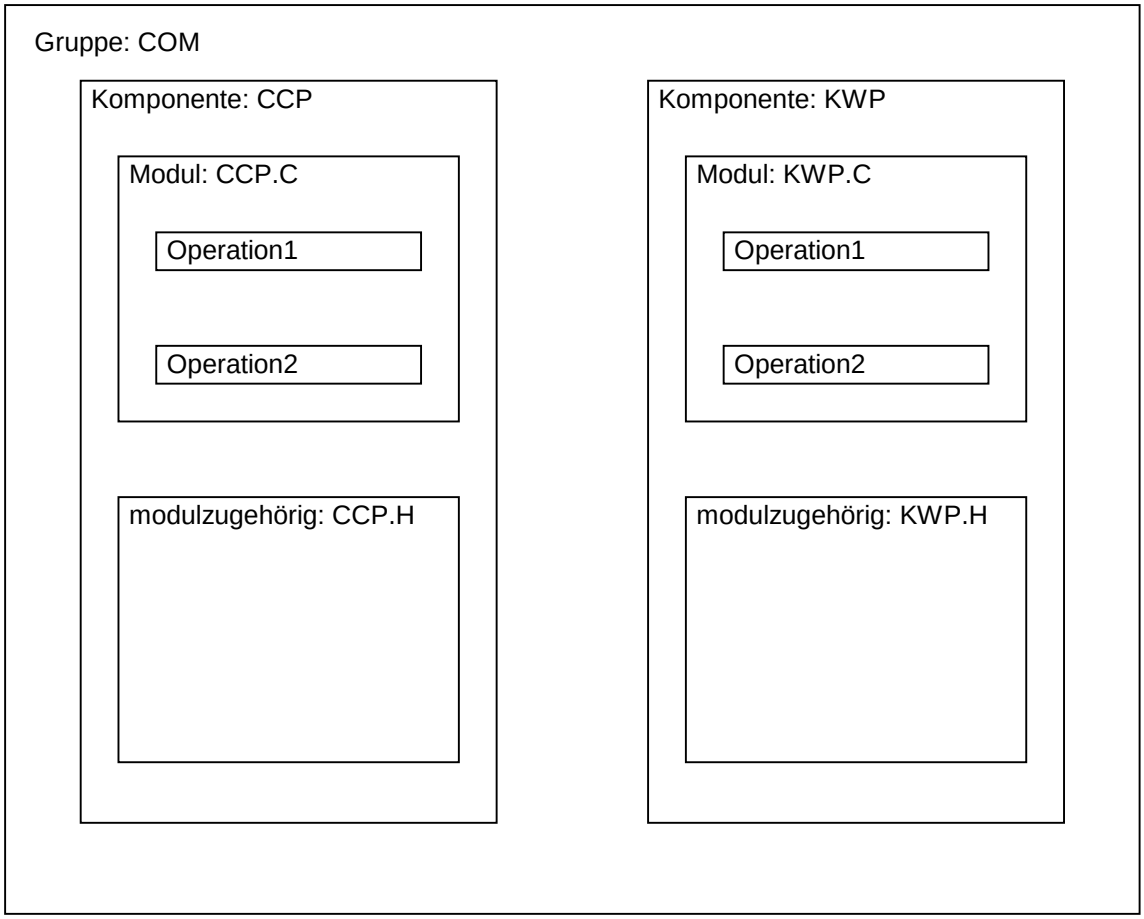
CCP.C
KWP.C

Anmerkungen:

- Komponentenabkürzung in allen Namen bei nicht-lokalen Objekten
- Speicherklasse in Namen wiedergeben (`_var` vs. `var` vs. `var_` z.B. für modul-global, lokal, persistent-lokal)
- Zwingende und umfassende Verwendung von `const`
- Zwingende Verwendung modulglobaler, nicht extern sichtbarer Operationen wo immer möglich
    - `static` benutzen
    - Modulkürzel weglassen
    - stehen im Quelltext vor globalen, extern sichtbaren Operationen
- Analog für Datenobjekte

1.4 Operation (Funktion)

Ist ein Teil eines Moduls, bestehend aus dem Operationsnamen und einer Folge von Anweisungen. Eine Operation hat einen definierten Ein- und Ausgang.



Diese Anordnung kann nach Bedarf vereinfacht (wenn eine Gruppe nur eine Komponente beinhaltet) oder verfeinert werden.

Style Guide – Echtzeitprogramme (zu Kapitel 4)

1.5 Style Guide

Der MISRA Standard „Guidelines For The Use Of The C Language In Vehicle Based Software", stellt eine Reihe von Regeln zur Verfügung, die die Erstellung von korrekten C-Programmen unterstützen sollen. Diese Regeln beziehen sich hauptsächlich auf die Gegebenheiten der Programmiersprache C und machen keine Aussage über den Programmierstil (Style guide). Aus diesem Grund soll dieses Kapitel als eine Ergänzung des MISRA-Standards bzgl. Definition des Style guide betrachtet werden.

Hinweis: (MISRA: R111) nennt als Referenz die MISRA Regel 111.

1.5.1 Datentypen

1.5.1.1 Basisdatentypen

Die zu verwendenden Datentypen müssen grundsätzlich über „Compilerschalter“ für unterschiedliche Prozessoren bzw. Compiler definiert werden. Es sind die in der folgenden Tabelle aufgeführten Basisdatentypen zu verwenden, um unterschiedliche Interpretationen von Datentypen auf unterschiedlichen Systemen auszuschließen. Die vom Compiler angebotenen Datentypen müssen auf die folgenden Basisdatentypen konvertiert werden.

ECU-Bezeichnung	**Datenbreite**	**ANSI Datentyp (z.B. MPC 565)**
bool	**8 bit**	
s8	8 Bit	signed char
u8	8 Bit	unsigned char
s16	16 Bit	signed int
u16	16 Bit	
s32	32 Bit	
u32	32 Bit	
s64	64 Bit	
u64	64 Bit	
f32	32 Bit	
f64	64 Bit	
f128	128 Bit	

1.5.1.2 Abgeleitete Datentypen

Aus den Basisdatentypen können nun weitere Datentypen abgeleitet werden. Sie werden mit einem nachgestellten „**_t**“ gekennzeichnet.

1.5.1.3 Strukturen / Bitvariablen

Die Datentypen der Bitvariablen sind nach ANSI-C zu deklarieren. Abgeleitete „typedefs“ sind compilerabhängig und führen zu Fehlern.

```
typedef struct
{
 unsigned int    bit1_b:1
 unsigned int    bit2_b:1
 unsigned int    bit3_b:1
.
.
.
}bitword_t
```

(MISRA: R111-R113)

1.5.1.4 Typumwandlung

Typumwandlung darf nur explizit vorgenommen werden. Hierdurch zeigt der Anwender, dass die Umwandlung gewollt ist. Implizite Typumwandlungen sind compilerabhängig. Damit ist das Programm nicht portierbar und die Datentypen nicht vorhersagbar.

(MISRA: R43-R45)

1.5.1.5 Präprozessorarithmetik

Präprozessorkonstanten, die für Präprozessorarithmetik verwendet werden, sind mit Angabe des Datentyps „double“ oder „float“ zu deklarieren. Anschließend ist der gewünschte Datentyp explizit anzugeben (als „cast“).

(MISRA: R87-R100)

Beispiele:

```
#define ADC_LOOP_TIME_C     ( (f64) 1000 )
#define ADC_FACTOR_C        ( (f64)  0.01 )
#define ADC_VALUE_C         ( (u16) ADC_LOOP_TIME_C * ADC_FACTOR_C  )
```

1.5.1.6 Deklaration von Datenmasken (Masken)

Die Deklaration von Datenmasken muss über „Compilerschalter“ erfolgen, da die Bit-Struktur und Byte-Ordnung prozessorabhängig sind. Der „else Zweig“ ist nicht optional. Der Kontrollfluss der Anweisungen muss über Einrückungen verdeutlicht werdern.

Beispiele:

```
#ifdef INTEL
  #define MASK 0x0001
#elif MOTOROLA
  #define MASK 0x0100
#else
  #error message_no_processor
#endif
```

1.5.2 Namenskonventionen

Übersichtlichkeit, Wartbarkeit und Kompatibilität (innerhalb eines Programmierungsteams) ist dann gewährleistet, wenn sinnvolle Namen sowie einheitliches Schreibformat für Variablen, Konstanten, Strukturen etc. gewählt werden. Aufgrund der Bestrebung zur Wiederverwendbarkeit der existierenden Software sollen die Namen für die oben aufgeführten Sprachelemente grundsätzlich projektunabhängig sein und so verbleiben. Die Namen sollen aussagekräftig und nicht zu kurz gestaltet werden. In Extremfällen sollen sie sinngemäß abgekürzt werden. Man beachte dabei, dass diese Bezeichnung nicht mit den reservierten Namen der Programmierumgebung (Compiler, Linker, Programmiersprache etc.) in Konflikt gerät!
Alle, zu einem funktionalen Modul gehörenden Bezeichner sollen mit einer definierten Abkürzung beginnen. Die Abkürzung soll sich nach Möglichkeit auf drei Buchstaben begrenzen.

1.5.2.1 Variablennamen

Variablennamen werden nach dem folgenden Schema gebildet:

[abkürzung]_[BezeichnungDerVariablen]_[variablentyp]

abkürzung	-	Name der zugehörigen Komponenten, klein geschrieben
BezeichnungDerVariablen	-	logische Bezeichnung der Variablen, keine Trennung der einzelne Wörter durch Unterstrich, sondern Kennzeichnung durch Groß-, Kleinschreibung
variablentyp	-	s8, u8, s16, u16, s32, u32, s64, u64, f32, f64, f128

Beispiele:

```
adc_BatteryVoltage_u16      /* Analogwert Batteriespannung */

can_ReceiveBuffer_u8[8]     /* CAN Empfangspuffer */
```

Bei lokalen Variablen (innerhalb der Operationen) ist die Verwendung der Komponentenbezeichnung nicht notwendig.

1.5.2.1.1 Zeiger (Pointer)

Die Pointernamen werden entsprechen den Vorgaben für Variablennamen gebildet, mit dem Unterschied, dass Variablentyp durch die Zeichenfolge „**_p**“ (Pointer) ersetzt wird.

[abkürzung]_[*BezeichnungDerVariablen*]_p

Beispiele:

```
can_TransmitBuffer_p        /* CAN Sendepuffer */
```

1.5.2.1.2 Struktur-Variablen

Die Struktur-Variablen werden entsprechen den Vorgaben für Variablennamen gebildet, mit dem Unterschied, dass Variablentyp durch die Zeichenfolge „**_st**“ (Struktur) ersetzt wird.

[abkürzung]_[BezeichnungDerVariablen]_st

Beispiele:

```
can_Bitfeld_t  can_ErrorStatus_st    /* CAN Status */
```

1.5.2.2 Applizierbare Größen

Applizierbare Größen (Kennwerte, Kennlinien, Kennfelder) haben folgende Bezeichnung:

[abkürzung]_[Bezeichnung]_[kennung]

abkürzung	-	Name der zugehörigen Komponenten, klein geschrieben
Bezeichnung	-	logische Bezeichnung der applizierbaren Größe, keine Trennung der einzelnen Wörter durch Unterstrich, sondern Kennzeichnung durch Großbuchstaben
kennung		kw, kl, kf, kr (keine Angabe vom Datentyp) kw – Kennwert kl – Kennlinie kf – Kennfeld (kr – Kennraum)

Beispiele:

```
adc_TempSensor_kl                 /* Sensorkennlinie */

adc_TempSensorThreshold_kw        /* Sensor-Schwellenwert */
```

1.5.2.3 Operationsnamen

Operationsnamen werden analog zu den Variablennamen gebildet, mit folgendem Unterschied:

- Der Variablentyp beschreibt den Rückgabewert der Operation, liefert die Operation keinen Wert zurück (void), entfällt diese Bezeichnung.

[abkürzung]_[BezeichnungDerOperation]_[variablentyp]( )

abkürzung	-	Name der zugehörigen Komponenten, klein geschrieben
BezeichnungDerOperation	-	logische Bezeichnung der Operation, keine Trennung der einzelne Wörter durch Unterstrich, sondern Kennzeichnung durch Großbuchstaben
variablentyp	-	s8, u8, s16, u16, s32, u32, s64, u64, f32, f64, f128

Beispiele:

```
u8 can_TransmitData_u8 ()
void can_BusOff ()
```

1.5.2.4 Konstanten per Defines

Konstanten werden mit Großbuchstaben geschrieben, mit der angehängten Zeichenkette „**_C“** gekennzeichnet und entsprechen dem folgenden Schema:

[ABKÜRZUNG]_[BEZEICHNUNG_DER_KONSTANTEN]_C

ABKÜRZUNG	-	Name der zugehörigen Komponenten, groß geschrieben
BEZEICHNUNG_DER_KONSTANTEN	-	logische Bezeichnung der Konstanten, Trennung der einzelnen Wörter durch Unterstrich

- Konstanten müssen eingeklammert werden.
- Konstanten werden explizit mit Datentyp definiert.

Beispiele:

```
#define CAN_RECEIVE_BUFFER_LENGTH_C        ( (u8)8 )
```

1.5.2.5 Makros

Makros werden mit Großbuchstaben geschrieben. Unmittelbar nach dem Makronamen folgt ein Klammernpaar „()“.

[ABKÜRZUNG]_[BEZEICHNUNG_DES_MAKROS]( )

ABKÜRZUNG	-	Name der zugehörigen Komponenten, groß geschrieben
BEZEICHNUNG_DES_MAKROS	-	logische Bezeichnung des Makros, Trennung der einzelne Wörter durch Unterstrich

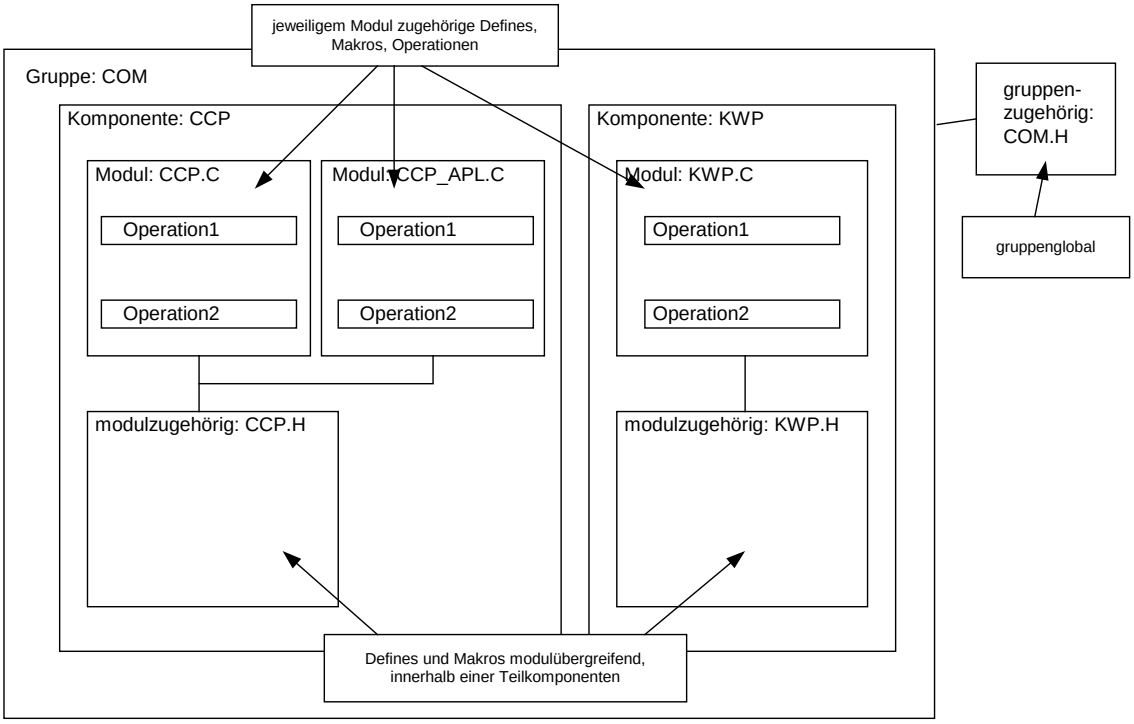
Beispiele:

```
CAN_NEWDAT_SET()
```

(MISRA: R90-R96)

1.5.2.6 Headerdateien

Die Namen der Headerdateien sollen mit den Abkürzungen, die die Zuordnung zu einer Komponenten signalisieren, bezeichnet werden. Dies fördert die Anschaulichkeit, Zugehörigkeit und Wartbarkeit der Software. In der folgenden Abbildung ist das Prinzip für die Erstellung von Headerdateien dargestellt.



Module dürfen eigene Header Files haben.

1.5.3 Projektdateien

1.5.3.1 Grundlegende Eigenschaften

- Es dürfen keine Tabulatoren vorhanden sein.
- Einrücktiefe: 3 Spaces
- Die Zeilenlänge darf 120 Zeichen nicht überschreiten.
- Die zusammengehörenden geschweiften Klammern sind übereinander zu positionieren.

```
if( )
{

}
else
{

}
```

- Sonderzeichen wie ä, ö, ü, ß sind verboten.
- Leerzeichen zwischen Operatoren sind zu verwenden: x + y

1.5.3.2 Modulstruktur

Um die Einheitlichkeit und die Lesbarkeit des Codes zu bewahren, ist in jeder Programmdatei ein Modulkopf (s. Anhang A) anzulegen. Für jede Operation ist ein Operationskopf (s. Anhang B) anzulegen.

1.5.4 Vordefinierte Compiler Makros und Datentypen

Vordefinierte Datentypen und Makros dürfen nicht benutzt werden, da diese ausschließlich von verwendeten Compilern abhängig sind.

1.5.5 „Return“-Anweisung

Innerhalb einer Operation soll maximal eine „Return-Anweisung“ als letzte Anweisung ausgeführt werden.

(MISRA: R79-R86)

1.5.6 Kommentare

Kommentare sollen erklären, was warum gemacht wird und nicht, wie es gemacht wird. Kommentare dürfen nicht verschachtelt sein. Das Wiederholen des Programmcodes in Worten ist verboten.

1.5.7 Programmcode

1.5.7.1 Bedingte Compilierung

Grundsätzlich ist eine „Bedingte Compilierung“ für unterschiedliche Prozessoren oder Varianten der Hardware etc. vorzusehen, auch wenn nur ein Prozessor verwendet wird. Dadurch ist eine Erweiterbarkeit bzw. Portierbarkeit möglich. Der „else-Zweig“ dient zur Fehlerbehandlung und muss vorhanden sein.
Der Kontrollfuß muss durch Einrückungen verdeutlicht werden.

Beispiele:

```
#ifdef INTEL
  ....
#elif MOTOROLA
  .....
#else
  #error message_no_processor
#endif
```

1.5.7.2 Änderungsdokumentation des Programmcodes

Programmcodeänderungen werden durch die entsprechenden Einträge in der Versionsverwaltung (PVCS) dokumentiert.

Anhang A – Aufbau eines C-Files (Modul)

Das C-File stellt eine Sammlung von Operationen dar, die logisch oder funktional miteinander verknüpft sind. Dieses File wird als Modul bezeichnet. In jedem File gibt es nur einen Modulkopf am Anfang des Files. Der Aufbau und Inhalt sind nachfolgend dargestellt.

```
/*****************************************************************
PROJEKT:            MSP0208
                    $Header:   $
NAME:               $Workfile: $
ERSTELLDATUM:       2002-09-24
CODING STANDARD:
BESCHREIBUNG:
REVISION:           $Revision: $
                    $Author:   $
                    $Date:     $
AENDERUNGSLISTE:
$Log:$
*****************************************************************/

/*---------------------------------------------------------------
    Header-Dateien (#include)
---------------------------------------------------------------*/

/*---------------------------------------------------------------
    Globale Variablen
---------------------------------------------------------------*/

/*---------------------------------------------------------------
    Globale Konstanten (const)
---------------------------------------------------------------*/

/*---------------------------------------------------------------
    Globale applizierbare Parameter (const)
---------------------------------------------------------------*/

/*---------------------------------------------------------------
    Modullokale Typdefinitionen (typedef)
---------------------------------------------------------------*/

/*---------------------------------------------------------------
    Modullokale Konstanten per Define(#define)
---------------------------------------------------------------*/

/*---------------------------------------------------------------
    Modullokale Makros (#define, inline)
```

```
------------------------------------------------------------------*/

/*------------------------------------------------------------------
    Modullokale Variablen (static)
------------------------------------------------------------------*/

/*------------------------------------------------------------------
    Modullokale Konstanten (static const)
------------------------------------------------------------------*/

/*------------------------------------------------------------------
    Modullokale applizierbare Parameter (static const)
------------------------------------------------------------------*/

/*------------------------------------------------------------------
    Modullokale Operationsprototypen (static)
------------------------------------------------------------------*/
```

Anhang B – Aufbau einer Operation

Nachfolgend ist die Grundstruktur, nach der jede Operation aufgebaut werden soll, dargestellt.

```
/*****************************************************************
OPERATIONSNAME: OperationsName
BESCHREIBUNG:
EINGAENGE:
AUSGAENGE:
AUTOR/DATUM:
*****************************************************************/

void OperationsName(void)
{


} /* Ende OperationsName */
```

Anmerkung:

Unter EINGAENGE / AUSGAENGE sind alle Größen (Parameter, globale Variablen), die gelesen / beschrieben werden, aufzulisten und zu erläutern.

Anhang C – Aufbau eines Header-Files

```
/*****************************************************************
PROJEKT:            MSP0208
                    $Header:   $
NAME:               $Workfile: $
ERSTELLDATUM:       2002-09-24
CODING STANDARD:
BESCHREIBUNG:
REVISION:           $Revision: $
                    $Author:   $
                    $Date:     $
AENDERUNGSLISTE:
$Log:$
*****************************************************************/

#ifndef header_h
#define header_h

/*-----------------------------------------------------------------
    Globale Typdefinitionen (typedef)
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
    Globale Konstanten per Define(#define)
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
    extern-Definitionen von globalen Variablen
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
    extern-Definitionen von globalen Konstanten
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
    extern-Definitionen von Operationsprototypen
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
    Globale Makros (#define)
-----------------------------------------------------------------*/
```