

SPS Programmierrichtlinien

Itris Automation- 31. Januar 2019 - Version 4.0.7

www.itris-automation.com

©Itris Automation

2008-2019

Contents

1 Zweck des Dokuments **4**

2 Struktur des Dokuments **5**
2.1 Gliederung der Regeln . . . 5
2.2 Aufführung der einzelnen Regeln . . . 5

3 Programmierrichtlinien **6**
3.1 Namensgebung . . . 6
3.1.1 N.1 - Alle wichtigen Elemente des Programms müssen benannt werden . . . 6
3.1.2 N.2 - Die Namen der Elemente müssen eine Mindest- oder Maximalgröße haben . . . 7
3.1.3 N.3 - Die physikalische Adresse sollte nicht in der Mnemonik der Elemente benutzt werden . 7
3.1.4 N.4 - Testvariablen dürfen nicht zum fertigen Programm gehören . . . 8
3.1.5 N.5 - Sonderzeichen in Mnemoniken sind verboten . . . 9
3.1.6 N.6 - Schlüsselwörter der Programmiersprache dürfen nicht als Mnemonik verwendet werden 9
3.2 Kommentare . . . 10
3.2.1 C.1 - Alle Programmelemente müssen kommentiert werden . . . 10
3.2.2 C.2 - Die Kommentare müssen mindestens x Zeichen enthalten . . . 10
3.2.3 C.3 - Jedes Netzwerk sollte kommentiert werden . . . 11
3.2.4 C.4 - Kommentare dürfen nicht die Anfangsmarkierung eines Kommentars enthalten . . . 11
3.3 Schreiben . . . 12
3.3.1 E.1 - Alle Variablen müssen vor dem Lesen geschrieben werden . . . 12
3.3.2 E.2 - Die Parameter der benutzerdefinierten Funktionsbausteine müssen korrekt genutzt werden 13
3.4 Struktur . . . 14
3.4.1 S.1 - Rückwärtssprünge sind verboten . . . 14
3.4.2 S.2 - Eine Variable darf nur in einem einzigen Programmteil/Routine geschrieben werden . . 14
3.4.3 S.3 - Eine Variable darf nur in einer einzigen Aufgabe geschrieben werden . . . 15
3.4.4 S.4 - Ein Ausgang darf nur einmal pro Programmzyklus geschrieben werden . . . 15
3.4.5 S.5 - Variablen dürfen nicht lokalisiert werden . . . 16
3.4.6 S.6 - Die Instanzen von FB/DFB/ADD ON dürfen nur ein einziges Mal aufgerufen werden . . 16
3.4.7 S.7 - Die definierten Variablen werden benutzt . . . 17
3.4.8 S.8 - Die Variablen überlagern sich nicht . . . 17
3.4.9 S.9 - Komplexitätsmuster . . . 18
3.4.10 S.12 - Bedingte Anweisungen sollten einen Standardfall haben . . . 18
3.4.11 S.13 - Ein GRAFCET sollte nur einen einzigen Anfangsschritt enthalten . . . 19
3.4.12 S.15 - Die Nutzung direkter Werte ist verboten . . . 19
3.4.13 S.16 - Ein Vergleich von Realen darf nie auf gleich oder ungleich lauten . . . 20
3.4.14 S.17 - Korrekter Einzug bei Programmen in Structured Text . . . 21
3.5 Nützliche Informationen . . . 21
3.5.1 I.1 - Der Code muss zu mindestens x% kommentiert sein . . . 21
3.5.2 I.2 - Vorhandensein von Code im Kommentar . . . 22
3.5.3 I.3 - Das Programm darf keinen redundanten Code bzw. Dead Code beinhalten . . . 23
3.5.4 I.4 - Liste der gesperrten Elemente . . . 24
3.5.5 I.5 - Zyklomatische Anzahl v(G) . . . 24
3.5.6 I.6 - Essentielle Komplexität ev(G) . . . 25

3.5.7 I.7 - Anzahl der Programmzeilen (LoC) . . . 25
3.5.8 I.8 - Anzahl der Schritte in GRAFCET . . . 25
3.5.9 I.9 - Anzahl der Verzweigungen des GRAFCET . . . 26
3.5.10 I.10 - Verhältnis von Copy/Paste in der Programmierung . . . 26
3.5.11 I.11 - Halstead Metrik . . . 27
3.5.12 I.12 - Anzahl der Referenzen an Elemente, die sich außerhalb der POU befinden . . . 28
3.5.13 I.13 - Anzahl der Aurufe jeder Routine / POU . . . 28
3.5.14 I.14 - Anzahl der von einer bestimmten Routine/POU aufgerufenen Routinen/POUs (FanOut) 29
3.5.15 I.15 - Minimale/maximale Aufruftiefe einer Routine/POU . . . 29
3.6 Programmiersprachen . . . 30
3.6.1 L.1 - Die Sprache Instruction List ist nicht erlaubt . . . 30

4 Anhang **31**
4.1 Zusammenfassung der Regeln . . . 31
4.2 Dokumentversionen . . . 33

Chapter 1

Zweck des Dokuments

Dieses Dokument enthält Programmierrichtlinien zur Verbesserung der Lesbarkeit, Wartungsfreundlichkeit und Konsistenz von SPS-Programmen. Beschrieben sind einfache Regeln, die bereits ab der ersten Anwendung einen deutlichen Effekt auf die Code-Qualität haben.

Sämtliche aufgeführte Regeln werden automatisch mit Hilfe des PLC Checkers überprüft. Dieser ist einsetzbar für die meisten Automaten der Hersteller Schneider Electric, Siemens, Rockwell Automation, Beckhoff, 3S (CoDeSys) und Phoenix Contact.

Die Richtlinien wurden von erfahrenen Mitarbeitern erarbeitet und basieren auf den Analysen zahlreicher Codes, die Kunden von Itris Automation Square in diversen Sektoren mit verschiedenen Automaten einsetzen. Sie beruhen außerdem auf bewährten Standards aus anderen Bereichen der Informatik (z.B. eingebettete Systeme), die seit Längerem zu den Grundsätzen von Programmierrichtlinien (best practice) zählen.

Über Itris Automation

Itris Automation ist ein französischer Softwarehersteller, der Entwicklungstools für speicherprogrammierbare Steuerungen (SPS) anbietet, um die Produktivität der Teams und die Qualität von Programmen durch innovative Lösungen zu optimieren. Seine Werkzeuge greifen nahtlos in die SPS-Lösungen der Anbieter.

Über Schneider Electric

Schneider Electric ist führend bei der digitalen Transformation von Energiemanagement und Automatisierung in Haushalten, Gebäuden, Rechenzentren, Infrastruktur und der Fertigungsindustrie.
Mit globaler Präsenz in über 100 Ländern ist Schneider Electric unumstrittener Marktführer bei Energiemanagement – Mittelspannung, Niederspannung und sichere Spannungsversorgung – und Automatisierung. Wir bieten integrierte, effiziente Lösungen, für die wir Energieverteilung, Automatisierungstechnik und Software kombinieren.
In unserem globalen Netzwerk arbeiten wir über unsere Open Platform mit einer großen Community aus Partnern, Integratoren und Entwicklern zusammen, um Echtzeitsteuerung und betriebliche Effizienz zu liefern.
Wir glauben, dass großartige Mitarbeiter und Partner Schneider Electric zu einem großartigen Unternehmen machen und dass unser Engagement für Innovation, Diversität und Nachhaltigkeit dafür sorgt, dass unser Motto „Life Is On“ überall, für jeden und jederzeit gilt.

Chapter 2

Struktur des Dokuments

2.1 Gliederung der Regeln

Die hier gelisteten Regeln sind in folgende Kategorien eingeteilt: Namensgebung, Kommentare, Schreiben, Struktur und Programmiersprachen. Ein Kapital enthält außerdem nützliche Informationen, bei denen es sich streng genommen nicht um Programmier-Regeln handelt. Dennoch erlaubt deren Verwendung eine Steigerung der Code-Qualität. Regeln für Sonderfunktionen bestimmter Automaten (z.B. Schneider Electric) sind zudem auf Anfrage erhältlich.

Jede Kategorie ist in zwei Abschnitte aufgeteilt:
1) Regeln, die unabhängig vom Automaten-Typ angewandt werden können
2) Regeln, die nur bei bestimmten Automaten-Typen angewandt werden können, bzw. an den jeweiligen Automaten-Typ angepasst werden müssen

2.2 Aufführung der einzelnen Regeln

Die Regeln sind entsprechend ihrer Anordnung im PLC Checker sortiert. Der Aufbau jeder Regel ist identisch. Der jeweilige Abschnitt besteht immer aus dem Titel, der Erklärung der Regel, ihrer Begründung und (falls nötig) einem Beispiel-Code, um sie verständlicher zu machen.

Chapter 3

Programmierrichtlinien

3.1 Namensgebung

Die in einem SPS-Programm befindlichen Elemente unterliegen einer Namensgebung (Benennung). Beispiele für Elemente sind Variablen, Routinen (Abschnitte, Subroutinen, FCs ...) und Funktionsblöcke (FB). Durch die Einhaltung der folgenden Regeln soll sichergestellt werden, dass alle Elemente auf eine lesbare und wartungsfreundliche Art benannt werden. Die Bibliotheken werden dabei generell nicht berücksichtigt. Natürlich raten wir davon ab, die in den Bibliotheken definierten Elemente umzubenennen.

3.1.1 N.1 - Alle wichtigen Elemente des Programms müssen benannt werden

name

Alle wichtigen Elemente des Programms müssen benannt werden

- a) Alle Programmvariablen müssen benannt werden
- b) Alle Programmcodes müssen benannt werden
- c) Alle Programmtypen müssen benannt werden

description

Alle wichtigen Elemente des Programms müssen benannt werden (Programm, Module, Variablen, DFB, FB, OB, SR, Routine, Add-On, UDT).

Vertiefende Erklärung : Wenn alle Programmelemente benannt sind, ist die Wartung einfacher und die Programminhalte sind leichter zu verstehen. Darüberhinaus führt das Weglassen von Mnemoniken zu implizierten Links zwischen Code und Hardware.

Level : Obligatorisch

errorMessage

- a) Die Variable VARIABLE hat keinen Namen
- b) Der Code FULLNAME hat keinen Namen
- c) Der Typ FULLNAME hat keinen Namen

3.1.2 N.2 - Die Namen der Elemente müssen eine Mindest- oder Maximalgröße haben

name

Die Namen der Elemente müssen eine Mindest- oder Maximalgröße haben

- a) Die Namen der Programmvariablen müssen mindestens Min Zeichen enthalten
- b) Die Namen der Programmcodes müssen mindestens Min Zeichen enthalten
- c) Die Namen der Programmvariablen dürfen nicht mehr als Max Zeichen enthalten
- d) Die Namen der Programmcodes dürfen nicht mehr als Max Zeichen enthalten
- e) Die Namen der Programmtypen dürfen nicht mehr als Max Zeichen enthalten
- f) Die Namen der Programmtypen müssen mindestens Min Zeichen enthalten
- g) Der Name des Programms muss mindestens Min Zeichen enthalten

description

Die Namen von Programmelementen müssen mindestens "Min" Zeichen haben und dürfen nicht mehr als "Max" Zeichen enthalten.

Vertiefende Erklärung : Eine Mindestanzahl von Zeichen sorgt für eine sinnvollere Namensgebung. Je nach SPS-Hersteller vermeidet eine maximale Zeichenanzahl, dass die Bezeichnung im Display oder Export abgeschnitten wird, und sie verbessert die Lesbarkeit.

Eine maximale Zeichenanzahl erlaubt:

- In Schneider Unity, Schneider PL7pro und Rockwell RSLogix: die Vermeidung einer abgeschnittenen Bezeichnung im Display oder Export, ...
- In Siemens Step7: die Lesbarkeit zu verbessern

Level : Empfohlen

errorMessage

- a) Der Name der Variable VARIABLE ist kürzer als gefordert
- b) Der Name des Codes FULLNAME ist kürzer als gefordert
- c) Der Name der Variable VARIABLE ist länger als gefordert
- d) Der Name des Codes FULLNAME ist länger als gefordert
- e) Der Name des Typs FULLNAME ist länger als gefordert
- f) Der Name des Typs FULLNAME ist kürzer als gefordert
- g) Der Name des Programms NAME ist kürzer als gefordert

3.1.3 N.3 - Die physikalische Adresse sollte nicht in der Mnemonik der Elemente benutzt werden

name

Die physikalische Adresse sollte nicht in der Mnemonik der Elemente benutzt werden

description

Die Mnemoniken der Elemente sollten keine Referenz zu ihrer physikalischen Adresse beinhalten (z.B., “MW2”, “FB4 “. . .).

Vertiefende Erklärung : Die Referenz zu einer physikalischen Adresse in der Gesamtheit des Codes kann schlechte Eigenschaften haben: die Lesbarkeit wird schlechter, und es kann zu gefährlichen Missverständnissen kommen. Dazu ist anzumerken, dass mit der Weiterentwicklung der SPS Arbeitsbereiche solche Referenzen in den Hintergrund rücken. Eine entsprechende Referenz zu einer physikalischen Adresse kann unabsichtlich sein, sie ist deswegen dennoch verboten.

Beispiel :

Physikalische Adresse	Mnemonik	Kommentar
%MW23	Cde_23	Niemals die Benennung einer Variablen auf ihre physikalische Adresse beziehen

Level : Obligatorisch

errorMessage

- a) Die Variable VARIABLE hat eine Mnemonik, die sich auf eine physikalische Adresse bezieht

3.1.4 N.4 - Testvariablen dürfen nicht zum fertigen Programm gehören

name

Testvariablen dürfen nicht zum fertigen Programm gehören

- a) Testvariablen dürfen nicht zum fertigen Programm gehören
- b) Unbekannte Variablen dürfen nicht zum fertigen Programm gehören

description

Die Mnemoniken test, toto, titi, tata, tutu, to_delete, todo, ... sind verboten, und Kommentare sollten keine Korrekturbefehle wie ’to do later’, ’to delete’ oder ’todo’ enthalten, durch die das Programm unfertig wirkt. Das Gleiche gilt für unverständliche bzw. unbekannte Variablen, die ein Entwickler nicht verstehen kann (wo im Namen oder im Kommentar Zusätze wie "unknown", "unbekannt" oder "?" auftauchen). Diese Art der Variablen könnte z.B. von einer vorhergehenden Version des Programms stammen, die noch von einem anderen Entwickler ohne ausreichenden Kommentar erstellt wurde.

Vertiefende Erklärung : Zurückgelassene Korrekturanweisungen reduzieren die Lesbarkeit des fertigen Programms und verhalten sich unter Umständen nicht wie erwartet.

Note: Die Liste der unerlaubten Bezeichnungen ist konfigurierbar.
Level : Obligatorisch im fertigen Programm (und ohne Relevanz während der Entwicklung)

errorMessage

- a) Die Testvariable VARIABLE ist noch präsent in der Anwendung
- b) Die unbekannte Variable VARIABLE ist noch präsent in der Anwendung

3.1.5 N.5 - Sonderzeichen in Mnemoniken sind verboten

name

Sonderzeichen in Mnemoniken sind verboten

- a) Sonderzeichen in Mnemoniken der Variablen sind verboten
- b) Sonderzeichen in Mnemoniken der Codes sind verboten

description

Sonderzeichen wie z.B. à, ä, é, ö, ç, ß, Leerzeichen, doppelter Unterstrich oder Anführungsstriche sind in den Mnemoniken untersagt. Einzig alphanumerische Zeichen und der Unterstrich sind erlaubt.

Vertiefende Erklärung : Die Nutzung von Sonderzeichen in der Benennung von Programmelementen und -variablen kann die Übertragbarkeit auf zukünftige SPS-Versionen und/oder Plattformen beeinträchtigen. Außerdem wird die Lesbarkeit im internationalen Kontext geschmälert. Schließlich erlaubt die Einschränkung auf alphanumerische Zeichen eine bessere Unterscheidung der Variablennamen.

Beispiel :

```
start__motor := true;

# .... viel später im Code

start_motor := false;       # Den doppelten Unterstrich '__' zuzulassen führt zu Verwirrung bei
diesen beiden Namen.
```

Level : Obligatorisch

errorMessage

- a) Die Variable VARIABLE hat eine Mnemonik, die ein Sonderzeichen enthält
- b) Der Code FULLNAME hat eine Mnemonik, die ein Sonderzeichen enthält

3.1.6 N.6 - Schlüsselwörter der Programmiersprache dürfen nicht als Mnemonik verwendet werden

name

Schlüsselwörter der Programmiersprache dürfen nicht als Mnemonik verwendet werden

description

Schlüsselwörter der Programmiersprache (z.B. "if", "set", "start"...) sollten nicht als Mnemonik verwendet werden. Es kann allerdings sein, dass sie in der Mnemonik enthalten sind. Meist werden diese Fälle von der SPS workbench behandelt, aber es gibt auch Ausnahmen.

Vertiefende Erklärung : Die Nutzung von Schlüsselwörtern in Mnemoniken beeinflusst die Lesbarkeit und Übertragbarkeit.

Beispiel :

```
start := true;        # Schlechte Mnemonik, start ist ein Schlüsselwort
start_motor := true;  # Richtige Mnemonik
```

Level : Obligatorisch

errorMessage

- a) Die Variable VARIABLE ist ein Schlüsselwort der Programmiersprache

3.2 Kommentare

Neben der Namensgebung sollten auch bei der Kommentierung von Programmen einige Regeln beachtet werden. Denn umso expliziter und detaillierter ein Kommentar ist, desto einfacher kann der entsprechende Code verstanden werden. Normalerweise fallen die Kommentare in den Bibliotheken nicht unter diese Regelung.

3.2.1 C.1 - Alle Programmelemente müssen kommentiert werden

name

Alle Programmelemente müssen kommentiert werden

- a) Alle Programmvariablen müssen kommentiert werden
- b) Alle Programmtypen müssen kommentiert werden
- c) Alle Programmcodes müssen kommentiert werden

description

Alle Programmelemente sollten kommentiert sein (Program, Section, Modules, Variables, DFB, FB, OB, SR).

Vertiefende Erklärung : Wenn alle Programmelemente ausreichend kommentiert sind fallen das Lesen und Verstehen des Programms leichter. Die Wartung des Programms wird dadurch erleichtert.

Level : Obligatorisch

errorMessage

- a) Die Variable VARIABLE hat keinen Kommentar
- b) Der Typ FULLNAME hat keinen Kommentar
- c) Der Code FULLNAME hat keinen Kommentar

3.2.2 C.2 - Die Kommentare müssen mindestens x Zeichen enthalten

name

Die Kommentare müssen mindestens x Zeichen enthalten

- a) Die Kommentare der Variablen müssen mindestens x Zeichen enthalten
- b) Die Kommentare der Codes müssen mindestens x Zeichen enthalten
- c) Die Kommentare der Variablen dürfen nicht leer sein
- d) Die Kommentare der Codes dürfen nicht leer sein
- e) Die Kommentare der Typen müssen mindestens x Zeichen enthalten
- f) Die Kommentare der Typen dürfen nicht leer sein

description

Die Kommentare von Programmelementen sollten eine Mindestzeichenanzahl (7 für Variablen, 15 für Code) haben, um nicht-aussagekräftige Kommentare zu vermeiden.

Vertiefende Erklärung : Eine Mindestlänge führt zu aussagekräftigeren Kommentaren. Eine maximale Länge unterstützt dabei, dass die Kommentare dennoch lesbar bleiben.

Level : Obligatorisch

errorMessage

- a) Die Variable VARIABLE hat einen zu kurzen Kommentar: 'STR'
- b) Der Code FULLNAME hat einen zu kurzen Kommentar: 'STR'
- c) Die Variable VARIABLE hat einen leeren Kommentar: 'STR'
- d) Der Code FULLNAME hat einen leeren Kommentar: 'STR'
- e) Der Typ FULLNAME hat einen zu kurzen Kommentar: 'STR'
- f) Der Typ FULLNAME hat einen leeren Kommentar: 'STR'

3.2.3 C.3 - Jedes Netzwerk sollte kommentiert werden

name

Jedes Netzwerk sollte kommentiert werden

description

Netzwerke sollten kommentiert werden. Diese Regel ist spezifisch für die Programmiersprachen aus dem IEC61131 Standard. Darüber hinaus existiert der Begriff Netzwerk in manchen SPS-Typen gar nicht.

Vertiefende Erklärung : Die ausgewiesene Rate für Kommentare ist hier global, es ist wichtig für eine gute Verteilung der Kommentare im ganzen Netzwerk zu sorgen.

Level : Obligatorisch

errorMessage

- a) Das Netzwerk LOCATION hat keinen Kommentar

3.2.4 C.4 - Kommentare dürfen nicht die Anfangsmarkierung eines Kommentars enthalten

name

Kommentare dürfen nicht die Anfangsmarkierung eines Kommentars enthalten
Note: *Der Titel darf nicht die id von vorher haben*

description

Kommentare dürfen nicht die Anfangsmarkierung eines Kommentars enthalten (z.B. “/ *”, “(*”). Darüber hinaus sind eingebettete Kommentare verboten (IEC11631 §2.1.5, system requirement 1.5.1 d).

Vertiefende Erklärung : Das Vorhandensein einer Kommentar-Anfangsmarkierung kann ein Symptom einer vergessenen Kommentar-Endmarkierung sein, welches zu Ausfällen in Teilen des Programms führen kann. In einigen SPS-Umgebungen gibt es eine globale Option, die eingebettete Kommentare zuläßt oder nicht.

Beispiel

```
/* Ein Kommentar mit einer vergessenen Endmarkierung

Critical_section_that_must_be_executed_which_is_found_in_the comment();

/* <- Hier wurde der gesamte Kommentar kommentiert, da die Endmarkierung des vorherigen Kommentars
vergessen wurde. */
```

Level : Empfohlen

errorMessage

- a) Der Kommentar LOCATION enthält die Anfangsmarkierung eines Kommentars: STR

3.3 Schreiben

3.3.1 E.1 - Alle Variablen müssen vor dem Lesen geschrieben werden

name

Alle Variablen müssen vor dem Lesen geschrieben werden

description

Außer für einige ganz bestimmte Kategorien von Variablen sollten alle Variablen eines SPS-Zyklus geschrieben werden bevor sie gelesen werden.

Vertiefende Erklärung : Eine Variable, die gelesen wird bevor sie geschrieben wurde, erhöht Verzögerungen im Programm. Effizienter Code sollte folgendes Diagramm während eines SPS-Zyklus respektieren:

Zusätzlich sollte sichergestellt werden, dass Variablen, die nirgendwo im Code geschrieben werden, auf andere Weise initialisiert werden oder von außerhalb des Programms kommen.

In einigen Fällen kann diese Regel ignoriert werden:

- bei in der Datenbasis initialisierten Variablen,
- bei Variablen, die einer Kommunikation entstammen,
- Zustandsvariablen, die den Wert eines vorherigen SPS-Zyklus speichern.

Beispiel:

```
if urgenteAlarme then
  alarmeTraitement();
end if;

# ...
```

```
urgenteAlarme := presenceDefault AND ...;
```

In diesem Beispiel wird die Alarmfunktion "alarmeTraitement" im nächsten SPS-Zyklus aufgerufen, was zu einer Verlängerung der durchschnittlichen Antwortzeit führt. Die Verzögerung beträgt etwa einen SPS-Zyklus.

Level : Empfohlen

errorMessage

- a) Die Variable VARIABLE wurde an der Stelle LOCATION gelesen bevor sie geschrieben wurde

3.3.2 E.2 - Die Parameter der benutzerdefinierten Funktionsbausteine müssen korrekt genutzt werden

name

Die Parameter der benutzerdefinierten Funktionsbausteine sollten korrekt genutzt werden

- a) Die Eingangsparameter der benutzerdefinierten Funktionsbausteine sollten gelesen werden
- b) Die Eingangsparameter der benutzerdefinierten Funktionsbausteine sollten nicht geschrieben werden
- c) Die Eingangs-/Ausgangsparameter der benutzerdefinierten Funktionsbausteine sollten gelesen werden
- d) Die Eingangs-/Ausgangsparameter der benutzerdefinierten Funktionsbausteine sollten geschrieben werden
- e) Die Ausgangsparameter der benutzerdefinierten Funktionsbausteine sollten geschrieben werden
- f) Die Ausgangsparameter der benutzerdefinierten Funktionsbausteine sollten nicht gelesen werden

description

Diese Regel ist in drei Regeln unterteilt:

- a und b - Eingangsparameter von benutzerdefinierten Funktionsbausteinen sollten gelesen und nicht geschrieben werden.
- c und d - Eingangs-/Ausgangsparameter von benutzerdefinierten Funktionsbausteinen sollten geschrieben und gelesen werden.
- e und f - Ausgangsparameter von benutzerdefinierten Funktionsbausteinen sollten geschrieben werden.

Vertiefende Erklärung: Falsch genutzte formelle Parameter sind Beweis für unfertigen Code, ebenso wie Code der unnütze Elemente richtigen Benehmens hat. In manchen SPSn kann der fehlerhafte Einsatz von Parametern unerwünschte, potentiell gefährliche Effekte haben wie z.B. Datenkorruption.

Note: Die Überprüfung der Benutzung privater Parameter von Bausteinen wird von Regel S7 durchgeführt.
Level: Obligatorisch

errorMessage

- a) Der Eingangsparameter VARIABLE wird nicht gelesen
- b) Der Eingangsparameter VARIABLE wird an der Stelle LOCATION geschrieben
- c) Der Eingangs-/Ausgangsparameter VARIABLE wird nicht gelesen
- d) Der Eingangs-/Ausgangsparameter VARIABLE wird nicht geschrieben
- e) Der Ausgangsparameter VARIABLE wird nicht geschrieben
- f) Der Ausgangsparameter VARIABLE wird gelesen

3.4 Struktur

3.4.1 S.1 - Rückwärtssprünge sind verboten

name

Rückwärtssprünge sind verboten

description

Rückwärtssprünge sind verboten.

Vertiefende Erklärung : Rückwärtssprünge können Endlosschleifen anstoßen, die zu einem Halt der Steuerung führen können. Das Vermeiden von Rückwärtssprüngen sorgt für ein leichteres Programmverständnis und vereinfacht die Wartung.

Beispiel:

```
L3:
  /* code always executed */

Always
True
Condition                       L3
-| |----------------------------->>
```

Level : Obligatorisch

errorMessage

- a) Ein Rückwärtssprung wurde an der Stelle LOCATION entdeckt

3.4.2 S.2 - Eine Variable darf nur in einem einzigen Programmteil/Routine geschrieben werden

name

Eine Variable darf nur in einem einzigen Programmteil bzw. einer Routine geschrieben werden

description

Eine Variable darf nur in einem einzigen Programmteil bzw. einer Routine geschrieben werden.

Vertiefende Erklärung : Eine einfache klare Programmarchitektur erleichtert das Verstehen des Programms und damit auch seine Wartung. Darüber hinaus kann das Schreiben von Variablen in mehreren Programmteilen zu fehlerhaftem Benehmen des Programms führen.

Level : Empfohlen

errorMessage

- a) Die Variable VARIABLE wird in mehreren Programmteilen geschrieben

3.4.3 S.3 - Eine Variable darf nur in einer einzigen Aufgabe geschrieben werden

name

Eine Variable darf nur in einer einzigen Aufgabe geschrieben werden

description

Eine Variable darf nur in einer einzigen Aufgabe geschrieben werden.

Vertiefende Erklärung : Wenn eine Variable in verschiedenen Aufgaben geschrieben wird, kann es zu unvorhersehbarem Verhalten des Programms kommen. Es entsteht z.B. ein Zugriffskonflikt, bei dem eine Aufgabe einen Wert verpasst, weil dieser gerade in der anderen Aufgabe geschrieben wird.

Level : Obligatorisch

errorMessage

- a) Die Variable VARIABLE wird in verschiedenen Aufgaben geschrieben, unter anderem an der Stelle LOCATION

3.4.4 S.4 - Ein Ausgang darf nur einmal pro Programmzyklus geschrieben werden

name

Ein Ausgang darf nur einmal pro Programmzyklus geschrieben werden

description

Ein Ausgang darf nur einmal pro Programmzyklus geschrieben werden.

Vertiefende Erklärung : Ausgänge mehrfach zu schreiben, kann zu Problemen bei der Zuverlässigkeit führen. Darüber hinaus ist ein Programm mit mehrfach geschriebenen Ausgängen schwerer zu verstehen und damit auch zu warten. Es sollte auch erwähnt werden, dass das Schreiben in Schleifen als mehrfaches Schreiben angesehen wird und somit eine Verletzung dieser Regel darstellt.

Beispiel :

```
if very_complicated_condition then
   Critical_Output := TRUE;
end if;
```

```
#.... viel später im Code

if another_very_complicated_condition then
   Critical_Output := FALSE;
end if;
```

Falls der Output nicht ausgeführt wird wie er sollte, wird eine Identifizierung, welche Zuweisung ausgeführt wurde, sehr schwierig.

Level : Obligatorisch

errorMessage

- a) Der Ausgang VARIABLE wird mehrfach benutzt

3.4.5 S.5 - Variablen dürfen nicht lokalisiert werden

name

Variablen dürfen nicht lokalisiert werden

description

Das Mapping von Variablen sollte nur für solche gelten, die von außerhalb des Programms angesteuert werden, z.B. von communication tags, I/O variables oder special function variables. Alle internen Variablen sollten nicht lokalisiert werden.

Vertiefende Erklärung : Die Nutzung lokalisierter Variablen führt zu einer stärkeren Speicherorganisation und beeinträchtigt somit die Entwicklungsfähigkeit des Programms. Sie vermindert außerdem die Übertragbarkeit, da sich die Speicherorganisationen der SPSn unterscheiden. Der Code kann dann in einer anderen Anwendung nicht benutzt werden, da der Speicherort vielleicht schon für andere Zwecke reserviert ist. Darüber hinaus kann es auch zu Fehlern in der Speicherorganisation kommen, bei denen verschiedene Variablen am gleichen Speicherort abgespeichert werden. Diese Fehler können über einen längeren Zeitraum unentdeckt bleiben und erst später im Entwicklungsprozess oder sogar erst während der Produktion angetroffen werden.

Level : Empfohlen

errorMessage

- a) Die Variable VARIABLE wurde lokalisiert

3.4.6 S.6 - Die Instanzen von FB/DFB/ADD ON dürfen nur ein einziges Mal aufgerufen werden

name

Die Instanzen von FB/DFB/ADD_ON dürfen nur ein einziges Mal aufgerufen werden

- a) Die Instanzen von FB/DFB/ADD_ON müssen aufgerufen werden
- b) Die Instanzen von FB/DFB/ADD_ON werden nicht mehrfach aufgerufen

description

Die Instanzen von FB/DFB/ADD_ON dürfen nur ein einziges Mal aufgerufen werden.

Vertiefende Erklärung : Ein Funktionsbaustein benutzt seine Variablen um dessen Status zwischen zwei aufeinanderfolgenden Zyklen abzuspeichern. Jede Ausführung des Funktionsbausteins sollte demnach dessen Variablen verändern, um diese Entwicklung aufzuzeigen. Wenn die FBs mehrmals ausgeführt werden, sieht es so aus als ob sich der Zyklus doppelt so schnell wie die anderen Instanzen entwickelt.

Das häufige mehrmalige Aufrufen einer gleichen Instanz ist das Ergebnis eines copy/paste-Prozesses, der nicht aktualisiert wurde.

Dies kann zu Fehlern führen, die nur schwer zu finden sind.

Level : Obligatorisch für das endgültige Programm (während der Entwicklung Regel weglassen)

errorMessage

- a) Die Instanz VARIABLE von FB/DFB/ADD_ON wird gar nicht aufgerufen, soll aber einmal aufgerufen werden
- b) Die Instanz VARIABLE von FB/DFB/ADD_ON wird STR Mal aufgerufen, soll aber nur einmal aufgerufen werden

3.4.7 S.7 - Die definierten Variablen werden benutzt

name

Die definierten Variablen werden benutzt

- a) Die definierten Variablen (außer reservierten) werden benutzt
- b) Die reservierten Variablen werden nicht benutzt
- c) Die definierten Typen werden benutzt

description

Definierte Variablen und Typen müssen benutzt werden. Dies gilt nicht für reservierte Variablen; diese dürfen nicht benutzt werden.

Vertiefende Erklärung : Unbenutzte Variablen (außer reservierte) und Typen sind manchmal Ausdruck einer vergessenen Bearbeitung oder unvollständigen Aktualisierung der Datenbasis. Die benutzten reservierten Variablen sind manchmal Ausdruck eines Umbenennungsprozesses, der nicht oder nicht vollständig umgesetzt wurde.

Level : Obligatorisch für das endgültige Programm (während der Entwicklung Regel weglassen)

errorMessage

- a) Die Variable VARIABLE wird nicht benutzt
- b) Die reservierte Variable VARIABLE wird benutzt
- c) Der Typ LOCATION wird nicht benutzt

3.4.8 S.8 - Die Variablen überlagern sich nicht

name

Die Variablen überlagern sich nicht

description

Wenn sich Variablen überlagern sollte nach dem Grund gesucht werden.

Vertiefende Erklärung : Lokalisierte Variablen, die sich überlagern, sind vergleichbar mit einem impliziten Transfer von Werten untereinander. Dies erschwert manchmal die Lesbarkeit des Programms. Dabei kann es sich auch um ein Lokalisierungsproblem handeln.

Level : Obligatorisch

errorMessage

- a) Die Variable VARIABLE wird durch eine andere überlagert

3.4.9 S.9 - Komplexitätsmuster

name

Komplexitätsmuster

- a) Schleifen
- b) Rückwärtssprünge
- c) Ebenen der Verschachtelung (3 Ebenen)
- d) Ebenen der Verschachtelung (4 oder mehr Ebenen)

description

In komplexen Programmen können sich Fehler leichter verstecken.

Vertiefende Erklärung : Einige Programmteile sind komplizierter als andere. Dies trifft besonders auf solche zu, bei denen die Ausführung aufeinanderfolgender Prozesse oft unterbrochen wird. Diese Regel sucht nach Schleifen, Rückwärtssprüngen und Kennzahlen der Verschachtelung.

Level : Empfohlen

errorMessage

- a) Eine Schleife wurde gefunden an der Stelle LOCATION mit folgendem Kontext: STR
- b) Ein Rückwärtssprung wurde entdeckt an der Stelle LOCATION
- c) Der Code an der Stelle LOCATION ist zu verschachtelt (sollte auf ein SR/DFB/FB/Add-on aufgeteilt werden) STR
- d) Der Code an der Stelle LOCATION ist zu verschachtelt (sollte auf ein SR/DFB/FB/Add-on aufgeteilt werden) STR

3.4.10 S.12 - Bedingte Anweisungen sollten einen Standardfall haben

name

Bedingte Anweisungen sollten einen Standardfall haben

description

Wenn man eine bedingte Anweisung IF oder CASE in Structured Text benutzt, ist es wichtig, immer auch einen Standardfall einzubinden: einen ELSE-Teil für IF-Anweisungen und einen Standardfall (DEFAULT) für CASE-Anweisungen.

Vertiefende Erklärung : Wenn bei bedingten Anweisungen immer ein Standardfall vorhanden ist, weiß der Leser, dass alle Entscheidungswege berücksichtigt wurden. Wenn nicht, dann könnte es sein, dass der Programmierer einen wichtigen Fall vergessen hat. Diese Regel sorgt demnach dafür, dass jede IF-Anweisung auch einen ELSE-Teil und jede CASE-Anweisung auch einen OTHERS-Fall hat, auch wenn diese Fälle ein 'ELSE NULL;'-Fall sind.

Note : Achtung: diese Regel wird nur auf Programmteile angewandt, die in Structured Text geschrieben wurden.

Level : Empfohlen

errorMessage

- a) Die bedingte Anweisung an der Stelle LOCATION hat keinen Standardfall (ELSE oder DEFAULT)

3.4.11 S.13 - Ein GRAFCET sollte nur einen einzigen Anfangsschritt enthalten

name

Ein GRAFCET sollte nur einen einzigen Anfangsschritt enthalten

- a) Ein GRAFCET (außer Makroschritte) muss einen Anfangsschritt enthalten
- b) Ein GRAFCET (außer Makroschritte) darf nur einen Anfangsschritt enthalten
- c) Ein Makroschritt darf keinen Anfangsschritt enthalten

description

Ein GRAFCET sollte mindestens einen aber auch nicht mehr als einen Anfangsschritt enthalten. Dies gilt nicht für Makroschritte, die gar keine Anfangsschritte enthalten dürfen.

Vertiefende Erklärung : Mehr als einen Anfangsschritt in einem GRAFCET zu haben ist fehleranfällig. GRAFCET ohne Anfangsschritt sollten auch gefunden werden. Solche Diagramme werden gewöhnlich für andere Programmteile verwendet. Ihr Verhalten ist schwer zu verstehen und vermindert damit die Wartbarkeit des Programms.

Level : Obligatorisch

errorMessage

- a) Der GRAFCET FULLNAME enthält keinen Anfangsschritt
- b) Der GRAFCET FULLNAME enthält mit STR mehr als einen Anfangsschritt
- c) Der Makroschritt FULLNAME enthält STR Anfangsschritt(e), darf aber keine Anfangsschritte enthalten

3.4.12 S.15 - Die Nutzung direkter Werte ist verboten

name

Die Nutzung direkter Werte ist verboten

description

Direkte Werte oder Konstanten sollten nicht im Code genutzt werden.

Vertiefende Erklärung : Direkte Werte oder Konstanten im Code zu nutzen übermittelt weniger Informationen als eine bestimmte wohldefinierte Konstante. Bei Veränderungen zur Programmoptimierung ist die definierte Konstante notwendig.

Beispiel: Nicht so:

```
FOR I IN 1..16 DO
  ...
END_FOR;
```

Stattdessen eher so:

```
FOR I IN 1..ELEMENT_MAX DO
  ...
END_FOR;
```

Level : Empfohlen

errorMessage

- a) Ein direkter Wert STR wird an der Stelle LOCATION genutzt

3.4.13 S.16 - Ein Vergleich von Realen darf nie auf gleich oder ungleich lauten

name

Ein Vergleich von Realen darf nie auf gleich oder ungleich lauten

description

Reale sollten immer durch das Kleiner-als-, Kleiner-gleich-, Größer-als- oder Größer-gleich-Zeichen verglichen werden.

Vertiefende Erklärung : Bei nicht ganz präzisierbaren Werten mit nur leicht unterschiedlichen Nachkommastellen, die durch Gleich oder Ungleich verglichen werden, wäre das Ergebnis immer TRUE oder FALSE.

Beispiel: Nicht so:

```
IF Temperature = 20.0 THEN
...
```

Stattdessen eher so:

```
EPSILON : REAL := 0.1; (* measure uncertainty *)

IF ABS(Temperature - 20.0) <= EPSILON THEN
...
```

Level : Obligatorisch

errorMessage

- a) Reale werden durch Gleichheits- oder Ungleichheitszeichen an der Stelle LOCATION verglichen

3.4.14 S.17 - Korrekter Einzug bei Programmen in Structured Text

name

Der in Structured Text geschriebene Code sollte korrekt eingezogen sein

- a) Der in Structured Text geschriebene Code sollte korrekt eingezogen sein
- b) Es darf nur eine Instruktion pro Codelinie geben
- c) Bedingte Anweisungen und Schleifenanweisungen müssen korrekt eingerückt sein

description

Während Code in Structured Text geschrieben wird, sollte darauf geachtet werden, dass der Code korrekt eingezogen wird, damit er lesbar bleibt.

Detailed explanation : Ein schlecht eingezogener Code ist schwer zu lesen und zu verstehen. Ein guter Einzug zeigt sich darin, dass alle Instruktionen einer gleichen Programmiertiefe an der selben Stelle einer Zeile beginnen.

Example: Nicht:

```
a := B + C; E := D + H;
     G := F + K;
```

Lieber so:

```
a := B + C;
E := D + H;
G := F + K;
```

Level : Empfohlen

errorMessage

- a) Der Codeeinzug ist nicht richtig in LOCATION
- b) Es befindet sich mehr als eine Instruktion pro Linie in LOCATION
- c) Die LOCATION -Anweisung ist nicht korrekt eingerückt

3.5 Nützliche Informationen

3.5.1 I.1 - Der Code muss zu mindestens x% kommentiert sein

name

Der Code muss zu mindestens x% kommentiert sein

- a) Der Code in Ladder muss mindestens x% kommentierten Code enthalten (also maximal 100-x% nicht kommentierten Code). x=30
- b) Literalcode muss mindestens x% kommentierten Code enthalten (also maximal 100-x% nicht kommentierten Code). x=30

description

Es wird empfohlen, das Programm zu einem Mindestprozentsatz zu kommentieren.

Vertiefende Erklärung : Es wird empfohlen, einen Mindestprozentsatz von 30% an Kommentaren im Programm zu haben (Verhältnis von kommentierten Elementen zu allen Elementen). Ein solches Verhältnis stellt sicher, dass während der Programmentwicklung die Gesichtspunkte Wartung und Lesbarkeit ausreichend berücksichtigt wurden.

Level : Obligatorisch

errorMessage

- a) Der Code in Ladder wurde nicht ausreichend kommentiert
- b) Der Literalcode wurde nicht ausreichend kommentiert

3.5.2 I.2 - Vorhandensein von Code im Kommentar

name

Vorhandensein von Code im Kommentar

- a) Kommentare dürfen keinen Code enthalten (PL7-Pro)
- b) Kommentare dürfen keinen Code enthalten (Unity Pro)
- c1) Kommentare dürfen keinen Code enthalten (STEP7, STEP5, TIAPORTAL, PLCOPEN, CODESYS)
- c2) Kommentare dürfen keinen Code enthalten (STEP7, STEP5, TIAPORTAL)
- c3) Kommentare dürfen keinen Code enthalten (STEP7, STEP5, TIAPORTAL)
- c4) Kommentare dürfen keinen Code enthalten (STEP7, STEP5, TIAPORTAL)
- c5) Kommentare dürfen keinen Code enthalten (STEP7, STEP5, TIAPORTAL)
- c6) Kommentare dürfen keinen Code enthalten (STEP7, STEP5, TIAPORTAL)
- c7) Kommentare dürfen keinen Code enthalten (STEP7, STEP5, TIAPORTAL)
- c8) Kommentare dürfen keinen Code enthalten (STEP7, STEP5, TIAPORTAL)
- c9) Kommentare dürfen keinen Code enthalten (STEP7, STEP5, TIAPORTAL)
- c10) Kommentare dürfen keinen Code enthalten (STEP7, STEP5, TIAPORTAL)
- c11) Kommentare dürfen keinen Code enthalten (STEP7, STEP5, TIAPORTAL)
- d) Kommentare dürfen keinen Code enthalten (RSLogix 5000)

description

Es ist wichtig, keinen Code im Kommentar zu haben.

Vertiefende Erklärung : Code im Kommentar kann verschiedene Ursachen haben. Er kann optionaler Code sein, der nicht zum Hauptprogramm gehört. Oder es ist ein Stück Code, der zu Testzwecken im Kommentar geparkt wurde, damit er beim Test nicht ausgeführt wird. Es können auch zusätzliche Instruktionen sein, die beim Testen helfen sollen. Und schließlich kann es ein Stück Code sein, dass nicht ordentlich funktioniert hat und in einem Kommentar abgelegt und später dort vergessen wurde. Fertige Produktionsprogramme sollten keine Hinweise mehr auf die Entwicklungsphase enthalten. Code im Kommentar läßt Zweifel aufkommen bei dem, der den Code nicht selbst

geschrieben hat, ihn aber warten soll.

Level : Obligatorisch für das endgültige Programm (während der Entwicklung Regel weglassen)

errorMessage

- a) Der Kommentar zu Code LOCATION scheint Code zu enthalten. STR
- b) Der Kommentar zu Code LOCATION scheint Code zu enthalten. STR
- c1) Der Kommentar zu Code LOCATION scheint Code zu enthalten. STR
- c2) Der Kommentar zu Code LOCATION scheint Code zu enthalten. STR
- c3) Der Kommentar zu Code LOCATION scheint Code zu enthalten. STR
- c4) Der Kommentar zu Code LOCATION scheint Code zu enthalten. STR
- c5) Der Kommentar zu Code LOCATION scheint Code zu enthalten. STR
- c6) Der Kommentar zu Code LOCATION scheint Code zu enthalten. STR
- c7) Der Kommentar zu Code LOCATION scheint Code zu enthalten. STR
- c8) Der Kommentar zu Code LOCATION scheint Code zu enthalten. STR
- c9) Der Kommentar zu Code LOCATION scheint Code zu enthalten. STR
- c10) Der Kommentar zu Code LOCATION scheint Code zu enthalten. STR
- c11) Der Kommentar zu Code LOCATION scheint Code zu enthalten. STR
- d) Der Kommentar zu Code LOCATION scheint Code zu enthalten. STR

3.5.3 I.3 - Das Programm darf keinen redundanten Code bzw. Dead Code beinhalten

name

Das Programm darf keinen Dead Code beinhalten

- a) Das Programm darf keinen Dead Code beinhalten
- b) Das Programm darf keinen Code beinhalten, der nie aufgerufen wird
- c) Keine AFI Anweisung (Always False Instructions) für Rockwell

description

Das Programm darf keinen Dead Code beinhalten.

Vertiefende Erklärung : Programme mit Dead Code sind schwerer zu lesen und zu warten. Es kann ein Symptom für ein funktionelles Problem sein. Vorsicht beim Auffinden von nicht zugeordneten Funktionen und Programmteilen nach nicht bedingten Sprüngen (unconditional jumps or gotos): Diese Regel ist nicht erschöpfend (z.B. wird Dead Code als Folge einer Auswertung nicht aufgespürt).

Level : Fatal

errorMessage

- a) Der Code LOCATION wird nie ausgeführt
- b) Der Code LOCATION wird nie aufgerufen
- c) Eine AFI-Anweisung (Always False Instructions) wurde entdeckt an der Stelle LOCATION

3.5.4 I.4 - Liste der gesperrten Elemente

name

Liste der gesperrten Elemente

description

Gesperrte oder verschlossene Elemente können nicht von PLC Checker übersetzt und verarbeitet werden. Sie sollten wenigstens identifiziert werden.

Vertiefende Erklärung : Bei Programmen bzw. Anwendungen in denen der Großteil des Codes gesperrt ist, ist es wichtig diese Tatsache zu kennen.

Level : Obligatorisch

errorMessage

- a) Das Element FULLNAME ist gesperrt und kann nicht von PLC Checker analysiert werden
- b) Das Element FULLNAME ist gesperrt und kann nicht von PLC Checker analysiert werden

3.5.5 I.5 - Zyklomatische Anzahl v(G)

name

Zyklomatische Anzahl v(G)

- a) Zyklomatische Anzahl v(G) (< =15)
- b) Zyklomatische Anzahl v(G) (> 15)

description

Die zyklomatische Anzahl gibt Aufschluss über die Komplexität des Programms. Es ist wichtig, diese Zahl auf einem angemessen niedrigen Niveau zu halten.

Vertiefende Erklärung : Die zyklomatische Anzahl ergibt sich aus der Menge der verschiedenen Pfade, die in einer Routine ausgeführt werden können. Je höher diese Anzahl ist, desto schwieriger wird es, das richtige Benehmen dieser Routine zu überprüfen. In solchen Fällen sollten die Routinen in Unterroutinen aufgeteilt werden, die dann alle eine zyklomatische Anzahl unter dem Schwellenwert aufweisen.

Level : info

errorMessage

- a) Das Element FULLNAME hat eine zyklomatische Anzahl von STR
- b) Das Element FULLNAME hat eine zyklomatische Anzahl von STR

3.5.6 I.6 - Essentielle Komplexität ev(G)

name

Essentielle Komplexität ev(G)

- a) Essentielle Komplexität ev(G) (= 1)
- b) Essentielle Komplexität ev(G) (> 1)

description

Die essentielle Komplexität gibt Aufschluss über die unstrukturierten Anweisungen. Es ist wichtig diesen Wert sehr niedrig zu halten.

Vertiefende Erklärung : Die essentielle Komplexität bildet sich aus der Anzahl der unstrukturierten Anweisungen in einer Routine. Je höher diese Anzahl ist, desto schwieriger wird es, das richtige Benehmen dieser Routine zu überprüfen. In den meisten Fällen können die unstrukturierten durch strukturierte Anweisungen ersetzt werden.

Level : Info

errorMessage

- a) Das Element FULLNAME hat eine essentielle Komplexität von STR
- b) Das Element FULLNAME hat eine essentielle Komplexität von STR

3.5.7 I.7 - Anzahl der Programmzeilen (LoC)

name

Anzahl der Programmzeilen (LoC)

description

Die Bezugsquelle Programmzeilen zählt die Anzahl der Anweisungen in einer POU. Diese Größe sollte nicht außer Kontrolle geraten.

Vertiefende Erklärung : Ausgiebige Abläufe oder Funktionsbausteine können schwer zu testen, lesen und warten sein. Es kann sinnvoll sein, sehr große Abläufe in zwei oder mehr Abläufe von angemessener Größe aufzuteilen.

Level : Info

errorMessage

- a) Das Element FULLNAME hat STR Programmzeilen (LoC)

3.5.8 I.8 - Anzahl der Schritte in GRAFCET

name

Anzahl der Schritte in GRAFCET

description

Dies ist die Anzahl der Schritte in einem GRAFCET.

Vertiefende Erklärung : Diese Größe weist die Anzahl der Schritte im jeweiligen GRAFCET aus. Diese Zahl kann anschließend genutzt werden, um sehr komplexe GRAFCET aufzudecken oder einen Überblick über die Größenverhältnisse der GRAFCET zueinander zu erhalten.

Level : Info

errorMessage

- a) Der GRAFCET FULLNAME hat STR Schritte.

3.5.9 I.9 - Anzahl der Verzweigungen des GRAFCET

name

Anzahl der Verzweigungen des GRAFCET

description

Diese Zahl ist die Summe der Verzweigungen innerhalb jeder AND/OR Abzweigung, die in diesem GRAFCET erfasst wurde.

Vertiefende Erklärung : Diese Regel errechnet die Anzahl der Verzweigungen, die in diesem GRAFCET verfügbar sind. Der ermittelte Wert gibt Aufschluß über die Komplexität dieses GRAFCET. Diese Zahl entspricht dem v(G) von imperativen Programmiersprachen.

Level : Info

errorMessage

- a) Der GRAFCET FULLNAME hat STR Verzweigungen

3.5.10 I.10 - Verhältnis von Copy/Paste in der Programmierung

name

Verhältnis von Copy/Paste in der Programmierung

description

Diese Zahl zeigt an, wie viel Prozent des Programms sich wiederholen.

Vertiefende Erklärung : Code durch kopieren wieder zu verwenden ist nicht gut. Die Durchführung der Wartung und vor allem die Weiterentwicklung des Programms sind teurer, denn Veränderungen und Korrekturen sollten auf jede Copy/Paste-Instanz angewandt werden. Darüber hinaus ist ein manuelles Copy/Paste fehleranfällig: nach dem Copy-/Paste sollte dieses Stück Code verändert werden. Wenn es nicht korrekt verändert wird, kommen unterschwellige Fehler hinein, die nur schwer zu finden sind.

Level : Info

errorMessage

- a) Das Verhältnis des Copy/Paste im Programm ist STR%.

3.5.11 I.11 - Halstead Metrik

name

Halstead Metrik

- a) Halstead Metrik: Anzahl der verwendeten unterschiedlichen Operatoren
- b) Halstead Metrik: Anzahl der verwendeten Operatoren
- c) Halstead Metrik: Anzahl der verwendeten unterschiedlichen Operanden
- d) Halstead Metrik: Anzahl der verwendeten Operanden
- e) Halstead Metrik: Vokabulargröße
- f) Halstead Metrik: Implementierungslänge
- g) Halstead Metrik: Halstead-Länge
- h) Halstead Metrik: Halstead-Volumen
- i) Halstead Metrik: Schwierigkeit
- j) Halstead Metrik: Aufwand
- k) Halstead Metrik: Implementierungszeit
- l) Halstead Metrik: Anzahl der gelieferten Programmfehler

description

Alle Halstead Metriken werden mit Hilfe dieser Regel errechnet und aufgelistet. Es sind dies die verschiedenen Metriken vorgestellt von Maurice Halstead (Anzahl der Operatoren, Anzahl der Operanden, Programmvolumen, Schwierigkeitsniveau...).

Vertiefende Erklärung : Diese Metriken helfen bei der Einschätzung, wie viele Fehler in etwa im Programm bei dessen Auslieferung sind.

Level : Info

errorMessage

- a) Halstead Metrik: für FULLNAME, Anzahl der verwendeten unterschiedlichen Operatoren ist STR.
- b) Halstead Metrik: für FULLNAME, Anzahl der verwendeten Operatoren ist STR.
- c) Halstead Metrik: für FULLNAME, Anzahl der verwendeten unterschiedlichen Operanden ist STR.
- d) Halstead Metrik: für FULLNAME, Anzahl der verwendeten Operanden ist STR.
- e) Halstead Metrik: für FULLNAME, Vokabulargröße ist STR.
- f) Halstead Metrik: für FULLNAME, Implementierungslänge ist STR.
- g) Halstead Metrik: für FULLNAME, Halstead-Länge ist STR.
- h) Halstead Metrik: für FULLNAME, Halstead-Volumen ist STR.

- i) Halstead Metrik: für FULLNAME, Schwierigkeit ist STR.
- j) Halstead Metrik: für FULLNAME, Aufwand ist STR.
- k) Halstead Metrik: für FULLNAME, Implementierungszeit ist STRs.
- l) Halstead Metrik: für FULLNAME, Anzahl der gelieferten Programmfehler ist STR.

3.5.12 I.12 - Anzahl der Referenzen an Elemente, die sich außerhalb der POU befinden

name

Anzahl der Referenzen an Elemente, die sich außerhalb der POU befinden

description

Um die Wiederverwendbarkeit von SPS-Programmcode sicherzustellen, sollten Referenzen an Elemente, die sich außerhalb einer POU befinden, vermieden werden.

Vertiefende Erklärung : Der Austausch mit externen Elementen sollte über definierte Schnittstellen erfolgen, wie formelle Parameter von Eingängen, Ausgängen oder Ein-/Ausgängen der POU. Das Ziel ist es, keine externen Referenzen in POU-code zu haben, der wiederverwendet werden soll.

Note : Referenzen an Systemvariablen können zugelassen werden, denn ihre Signifikanz und Adresse ändern sich nicht von einem Programm zum anderen.

Level : Info

errorMessage

- a) Der POU FULLNAME hat STR externe Referenzen

3.5.13 I.13 - Anzahl der Aurufe jeder Routine / POU

name

Anzahl der Aufrufe jeder Routine / POU

description

Anzahl der Aufrufe jeder Routine/POU in einem Programm

Vertiefende Erklärung : Abgesehen von der Ausnahme, dass ein POU, der nie aufgerufen wird, Dead Code entspricht, gibt es keine Begrenzung für die Anzahl der Aufrufe eines POU. Diese Anzahl erlaubt einen Überblick über die Wiederverwendung von POUs.

Level : Info

errorMessage

- a) Der POU FULLNAME wird STR Male aufgerufen

3.5.14 I.14 - Anzahl der von einer bestimmten Routine/POU aufgerufenen Routinen/POUs (FanOut)

name

Anzahl der von einer bestimmten Routine/POU aufgerufenen Routinen/POUs (FanOut)

description

Diese Regel zählt die Anzahl der von einer bestimmten Routine/POU aufgerufenen Routinen/POUs (FanOut)

Vertiefende Erklärung : Ein POU, der keine Referenz zu einem anderen Block hat ist ein autonomer POU. Wenn die ermittelte Anzahl erhöht ist, könnte der betreffende POU in kleinere Blöcke geteilt werden, um die Struktur des Programms zu verbessern.

Level : Info

errorMessage

- a) Die Routine/POU FULLNAME ruft STR andere Routinen/POUs auf

3.5.15 I.15 - Minimale/maximale Aufruftiefe einer Routine/POU

name

Aufruftiefe einer Routine/POU

- a) Minimale effektive Aufruftiefe einer Routine/POU
- b) Maximale effektive Aufruftiefe einer Routine/POU
- c) Maximale theoretische Aufruftiefe einer Routine/POU

description

Diese Regel misst das Minimum und Maximum der Aufruftiefe eines Programmblocks (POU).

Vertiefende Erklärung : Diese Berechnung erlaubt es, die Aufruftiefe von Blöcken in der Struktur eines Programms herauszufinden. Ein Block kann auf Höhe einer Aufgabe aufgerufen werden, wobei die Tiefe gering wäre, oder er kann tiefer in der Struktur des Programms aufgerufen werden, was eine größere Tiefe bedeuten würde. Ein Block kann auch mehrmals im Programm aufgerufen werden, was bedeuten würde, dass er verschiedene Tiefen hätte. Eine geringe Tiefe entspricht einem Organisations- oder Verwaltungsblock im Programm. Eine größere Tiefe entspricht einem spezialisierten Programmblock.

Level : Info

errorMessage

- a) Das Element FULLNAME hat eine minimale effektive Aufruftiefe von STR.
- b) Das Element FULLNAME hat eine maximale effektive Aufruftiefe von STR.
- c) Das Element FULLNAME hat eine maximale theoretische Aufruftiefe von STR.

3.6 Programmiersprachen

3.6.1 L.1 - Die Sprache Instruction List ist nicht erlaubt

name

Die Sprache Instruction List ist nicht erlaubt.

description

Es wird empfohlen, die Sprache Instruction List (IL) zu vermeiden, denn es ist eine schwer lesbare niedere Sprache.

Vertiefende Erklärung : Je nach SPS-Typ kann es schwierig sein, die Sprache IL zu vermeiden. So kann es zum Beispiel sein, dass wie bei Siemens Step5 oder Step7 alle Teile, die in Ladder und Function Block Diagrams geschrieben wurden in IL abgespeichert sind. Bei anderen SPS-Typen wie von Schneider oder Rockwell ist es viel leichter, die Sprache IL zu vermeiden.

Level : Fatal

errorMessage

- a) Sprache Anweisungsliste ist nicht erlaubt in FULLNAME.

Chapter 4

Anhang

4.1 Zusammenfassung der Regeln

Dieser Anhang fasst alle Regeln und sonstige nützliche Informationen aus diesem Dokument noch einmal zusammen.

Regeln zur Namensgebung

- N.1 - Alle SPS-Typen - Alle wichtigen Elemente des Programms müssen benannt werden
- N.2 - Alle SPS-Typen - Die Namen der Elemente müssen eine Mindest- oder Maximalgröße haben
- N.3 - Alle SPS-Typen - Die physikalische Adresse sollte nicht in der Mnemonik der Elemente benutzt werden
- N.4 - Alle SPS-Typen - Testvariablen dürfen nicht zum fertigen Programm gehören
- N.5 - Alle SPS-Typen - Sonderzeichen in Mnemoniken sind verboten
- N.6 - Alle SPS-Typen - Schlüsselwörter der Programmiersprache dürfen nicht als Mnemonik verwendet werden

Regeln zu den Kommentaren

- C.1 - Alle SPS-Typen - Alle Programmelemente müssen kommentiert werden
- C.2 - Alle SPS-Typen - Die Kommentare müssen mindestens x Zeichen enthalten
- C.3 - Alle SPS-Typen - Jedes Netzwerk sollte kommentiert werden
- C.4 - Alle SPS-Typen - Kommentare dürfen nicht die Anfangsmarkierung eines Kommentars enthalten

Regeln zum Schreiben

- E.1 - Alle SPS-Typen - Alle Variablen müssen vor dem Lesen geschrieben werden
- E.2 - Alle SPS-Typen - Die Parameter der benutzerdefinierten Funktionsbausteine müssen korrekt genutzt werden

Regeln zur Struktur

- S.1 - Alle SPS-Typen - Rückwärtssprünge sind verboten
- S.2 - Alle SPS-Typen - Eine Variable darf nur in einem einzigen Programmteil bzw. einer Routine geschrieben werden
- S.3 - Alle SPS-Typen - Eine Variable darf nur in einer einzigen Aufgabe geschrieben werden
- S.4 - Alle SPS-Typen - Ein Ausgang darf nur einmal pro Programmzyklus geschrieben werden

- S.5 - Alle SPS-Typen - Variablen dürfen nicht lokalisiert werden
- S.6 - Alle SPS-Typen - Die Instanzen von FB/DFB/ADD ON dürfen nur ein einziges Mal aufgerufen werden
- S.7 - Alle SPS-Typen - Die definierten Variablen werden benutzt
- S.8 - Alle SPS-Typen - Die Variablen überlagern sich nicht
- S.9 - Alle SPS-Typen - Komplexitätsmuster
- S.12 - Alle SPS-Typen - Bedingte Anweisungen sollten einen Standardfall haben
- S.13 - Alle SPS-Typen - Ein GRAFCET sollte nur einen einzigen Anfangsschritt enthalten
- S.15 - Alle SPS-Typen - Die Nutzung direkter Werte ist verboten
- S.16 - Alle SPS-Typen - Ein Vergleich von Realen darf nie auf gleich oder ungleich lauten
- S.17 - Alle SPS-Typen - Korrekter Einzug bei Programmen in Structured Text

Nützliche Informationen

- I.1 - Alle SPS-Typen - Der Code muss zu mindestens x% kommentiert sein
- I.2 - Alle SPS-Typen - Vorhandensein von Code im Kommentar
- I.3 - Alle SPS-Typen - Das Programm darf keinen redundanten Code bzw. Dead Code beinhalten
- I.4 - Alle SPS-Typen - Liste der gesperrten Elemente
- I.5 - Alle SPS-Typen - Zyklomatische Anzahl v(G)
- I.6 - Alle SPS-Typen - Essentielle Komplexität ev(G)
- I.7 - Alle SPS-Typen - Anzahl der Programmzeilen (LoC)
- I.8 - Alle SPS-Typen - Anzahl der Schritte in GRAFCET
- I.9 - Alle SPS-Typen - Anzahl der Verzweigungen des GRAFCET
- I.10 - Alle SPS-Typen - Verhältnis von Copy/Paste in der Programmierung
- I.11 - Alle SPS-Typen - Halstead Metrik
- I.12 - Alle SPS-Typen - Anzahl der Referenzen an Elemente, die sich außerhalb der POU befinden
- I.13 - Alle SPS-Typen - Anzahl der Aufrufe jeder Routine/POU
- I.14 - Alle SPS-Typen - Anzahl der von einer bestimmten Routine/POU aufgerufenen Routinen/POUs (FanOut)
- I.15 - Alle SPS-Typen - Minimale/maximale Aufruftiefe einer Routine/POU

Programmiersprachen

- L.1 - Alle SPS-Typen - Die Sprache Anweisungsliste ist nicht erlaubt

4.2 Dokumentversionen

Datum	Änderungen
17.11.2016	Erste deutsche Version der Programmierrichtlinien basierend auf der GQR-Datei, Version 3.1.1
18.01.2017	Komplette deutsche Version der Programmierrichtlinien basierend auf der GQR-Datei, Version 3.1.1
31.01.2019	Komplette deutsche Version der Programmierrichtlinien basierend auf der GQR-Datei, Version 4.0.7
14.03.2019	Added "About Itris" and "About Schneider" to Zweck des Dokuments