1 Einleitung

1.1 Die Idee des Buchs

Hat der Programmierer von programmierbaren Controllern (*SPS*) tatsächlich den Wunsch, objektorientiert zu programmieren? Aus den Kommentaren zum Thema Objekte, erfährt man eigentlich nur, dass dies heutzutage nicht nur Standard, sondern auch selbstverständlich ist. Bei der Frage „Welche Objekte verwendest du?“ ist es dann mit der Selbstverständlichkeit allerdings schnell vorbei. In der Praxis findet man sogar ausreichend Beispiele, dass auch in der IT-Welt diese Frage häufig nicht richtig und zufriedenstellend beantwortet werden kann.

Wo liegt nun das Interesse an der so genannten objektorientierten Programmierung? Meist liefert der Auftraggeber die Antwort! Er möchte wissen, wie die gelieferte Software funktioniert; und dies ist nur möglich, wenn sie nachvollziehbar ist. Nicht nur ein sehr wichtiger Aspekt in der Anwendung zur Steuerung und Regelung von Prozessen, sondern mittlerweile auch ein zusätzliches Beurteilungsmerkmal in der Steuerungstechnik für sicherheitsgerichtete Produktionsprozesse. Gerade die SPS ist dazu prädestiniert und wird deshalb sehr häufig für diese Prozesse eingesetzt. Die Forderung zur objektorientierten Programmierung ist zweifellos berechtigt und hat sich im IT-Bereich bereits bewährt.

Wie kann nun eine Programmiersprache wie *Step 7/SCL* objektorientiert angewendet werden?

1.2 Der Weg zur Anwendung

Gehen wir davon aus, dass Sie das Thema „Objektorientierte Programmierung“ kennen lernen möchten und somit in unserer weiteren Betrachtung kein Zweifel mehr besteht, sich damit auseinander zu setzen. Alle sind motiviert und möchten nun gerne wissen, wie wir das mit *Step 7/SCL* machen. Nun – zunächst bietet der Editor von *Step 7* eigentlich nicht die Voraussetzung, direkt objektorientiert zu programmieren. Der Entschluss, dies zu lernen und sich darauf umzustellen, ist vorerst die entscheidende Voraussetzung, damit wir später unser Wissen auf die gegebene *SPS* umsetzen können.

Mit dieser Motivation fragen wir uns nun, wie erlernt man „Objektorientierte Programmierung“? Wir müssen uns klar darüber sein, dass unser Vorhaben nicht nur eine Programmiersprache voraussetzt, sondern auch das nötige Wissen über objektorientiertes Design (*OOD*) und objektorientierte Programmierung (*OOP*). Wenn wir uns dieses Wissen – nur soweit notwendig für unser Vorhaben – angeeignet haben, können wir trotzdem immer noch nicht

so richtig objektorientiert programmieren. Wir sollten deshalb akzeptieren, dass ein objektorientierter Entwurf und die Programmiersprache zwei Paar Schuhe sind.

Eine ganz wichtige Akzeptanz.

Objektorientierte Systeme müssen dokumentiert werden und auch das ist Industriestandard. Dazu gehören die Analyse und das Design. Inwieweit wir uns dies aneignen müssen und was man darunter versteht, ist wieder ein anderes Kapitel. Im wahrsten Sinne des Wortes werden wir natürlich als engagierte Techniker nur einen begrenzten Teil der *OOD* und *OOP* erlernen, um danach mit dem objektorientierten Programmieren loslegen zu können. Wir beschäftigen uns also zunächst ein wenig mit dem Designwerkzeug *UML*, um damit die für unser Vorhaben existierenden Diagramme lesen und verstehen zu lernen. Danach werden wir dieses Wissen schon in Bezug auf die Automatisierung üben und festigen. So entstehen für uns interessante *Design-Patterns* (Entwurfsmuster) und wenn wir das Sprachangebot von *Step 7/SCL* betrachten, dann haben wir unser Ziel erreicht, denn die gestellten Aufgaben in diesem Buch sind alle durch die Umsetzung in *SCL* mit dem TIA Portal erfolgreich für die S7-1500 und soweit möglich und sinnvoll auch für die S7-1200 übersetzt und getestet worden!

1.3 Voraussetzungen für den Leser

Die Motivation soll vom Herzen kommen und das Thema entschieden sein, denn Automatisierer tun sich sehr schwer, sich vom Spaghetti-Code und freiem Leben der königlichen Programmierung nach eigenem Stil zu trennen. Grundsätzlich glaube ich, dass die Gruppe, die sich am schwersten tut, *OOP* tatsächlich anzunehmen, die Gruppe der *SPS*-Programmierer ist. Man denkt schließlich in Merkern, Bits und Bytes. Und da sind wir auch schon bei den Voraussetzungen, dieses Buch zu lesen und zu verstehen. Das Basiswissen der Programmiersprache Step 7 sollte einigermaßen verstanden sein. *SCL*-Programmierer, welche *FUB*, *KOP* und *AWL* kennen, haben eine gute Voraussetzung. Hochsprachenanwender sind für den Bereich *SCL* noch besser vorbereitet.

1.4 Beispiele und Übungen

Zum besseren Verständnis werden wir alle Themen mit Beispielen bestücken. Nichts soll rein theoretisch abgehandelt werden. Als Basis werden wir hin und wieder, nur um vergleichen zu können, auch kleine Beispiele aus der Programmiersprache *C++* zeigen. Im Wesentlichen allerdings sind viele Übungen und praktische Abschnitte erkennbar, die sicherlich so manch traditioneller Programmierer schon einmal in seinen eigenen Anwendungen gesehen hat. In den ersten Schritten mit *SCL* werden diese mit *FUP* verglichen, so kann hier eine gute Verbindung zu der doch heute oft verwendeten Programmierdarstellung *FUP* geschaffen werden. Die getesteten Programmbeispiele befinden sich auf der beigefügten *CD-ROM* als TIA-Projekte.

1.5 Kapitelübersicht zu diesem Buch

Teil I – Einführung in das TIA Portal

Das erste Werkzeug zur Anwendung unseres Vorhabens wird das Framework TIA Portal sein. Dazu finden Sie am Ende dieses Kapitels für die persönliche Installation verschiedene Download-Links von Trial-Versionen für das TIA Portal V14. Um keine Missverständnisse aufkommen zu lassen, möchte ich darauf hinweisen, dass diese Einführung kein Handbuch zum TIA Portal ersetzen kann und dies auch nicht beabsichtigt war. Wir werden an kleinen Beispielen das TIA Portal automatisch kennen lernen, um später ohne Probleme unser Vorhaben editieren und übersetzen zu können. Die Themen zur Installation, Systemvoraussetzungen und Migrieren sind **nicht** Umfang dieses Buchs. Diese Themen können im Informationssystem zum TIA Portal ausreichend nachgelesen werden.

► Kapitel 2: Start mit dem TIA Portal

Der Einstieg erfolgt mit dem Kennenlernen des neuen Editors. Es erfolgt zunächst eine Übersicht der Bedienoberflächen und Einstellungen für die gemeinsamen Projekte zu diesem Buch. Ein Überblick zum Informationssystem soll dem Anwender in Kürze zeigen, welche Infos nachgelesen werden können. Danach erstellen wir je ein leeres Basis-Projekt für die S7-1500- und S7-1200-CPU für die folgenden Projekte im Buch und starten den Debug-Modus mit *S7-PLCSIM*.

► Kapitel 3: Erstellen der Bausteine *FB* und *FC*

Die Grundlage ein Anwenderprogramm zu erstellen, erfordert ein minimales Basiswissen bezüglich der Editor-Eigenschaften und der Bausteinschnittstelle. Dies wird mit der Erstellung der Bausteine wie *FC*- und *FB*-Komponenten erarbeitet und stellt so eine wichtige Grundlage zum späteren Vorhaben dar.

► Kapitel 4: Schnittstellen

Die Schnittstellen zum *FC* und *FB* sind Grundlage der Kommunikation und Datenverarbeitung. Dazu gehören auch die Variablen für die Kommunikation zur Außenwelt wie Eingänge, Ausgänge und Peripheriezugriffe. Dieses Kapitel zeigt die Möglichkeiten der Daten-Schnittstelle zum *FB* und *FC* mit elementaren und anwenderorientierten Datentypen (*UDT*) und der Objekt-Schnittstelle.

Teil II – Einführung in *SCL*

Es wird versucht, diesen Abschnitt so zu behandeln, dass jeder mit entsprechender Bereitschaft und Willen in der Lage ist, vom *FUP*-Programmierer auf *SCL* umzusteigen. Dazu werden die Themen in *SCL*, solange dies sinnvoll ist, auch in *FUP* dargestellt. Das soll dem Umsteiger Anregungen bringen, seine traditionelle Welt in *FUP* mit *SCL* vergleichen zu können. Auch wenn das zunächst nicht praktikabel erscheint, soll es zumindest einen weichen Übergang in die *SCL*-Welt ermöglichen.

► Kapitel 5: *SCL*-Einstieg mit Operatoren

Nach dem Motto „Aller Anfang ist schwer“, werden zunächst bekannte Dinge aus der *FUP*-Welt nach *SCL* umgesetzt. Das entspricht nicht der taktischen Vorgehensweise, Personen auf *SCL* umzuformen, sondern soll mehr eine Darstellung in Plauderform sein, um die Möglichkeiten vorzustellen und zu vergleichen, ohne bereits besondere Programmierkenntnisse in *SCL* zu besitzen. Die Vorstellung der Operatoren ist Thema dieses Kapitels und zeigt eine Übersicht der in *SCL* möglichen Operatoren. Da diese bereits überwiegend dem Programmierer bekannt sind, werden gezielt ausgesuchte Operatoren vertieft betrachtet.

► Kapitel 6: Datentypen und Deklarationen

Aller Anfang ist schwer. Besonders intensiv muss der Umgang mit den Datentypen und deren Deklarationen geübt werden. Das Thema ist trocken und für manchen Programmierer möglicherweise ein Dorn im Auge, wenn es zudem darum geht, fortgeschrittene Datentypen und *Pointer* zu verstehen oder gar anzuwenden. Die in *SCL* besonderen Eigenschaften zur Übereinstimmung der Datentypen sind schon extrem penibel und fordern so manches Mal ein besonderes Verständnis für den Anwender dieser Programmiersprache. Das Kapitel zeigt die erforderlichen Kenntnisse zu den Datentypen. Das Thema *ANY-Pointer* und die damit verbundene Sicht auf Daten ist ein besonders wichtiger Abschnitt für die fortgeschrittene Programmierung.

► Kapitel 7: Kontrollstrukturen

Das Thema ist den meisten Programmierern bekannt und soll sich deswegen auf die Anweisungen konzentrieren, die in *FUP* sonst nicht so üblich sind. Damit sind überwiegend die Schleifen und *CASE*-Anweisungen gemeint. Eine wichtige Erkenntnis zu diesem Thema ist, dass später nur zyklisch programmierte Anweisungen in den Bausteinen vorgesehen sind. Sie lernen hier eine etwas andere Art zu programmieren. Ich sage immer: „Die *SPS* freut sich, wenn so programmiert wird.“

► Kapitel 8: Zeiten in *SCL*

Nach dem Motto, wie macht man eine Zeit in *SCL*, soll das Thema hier nicht vergessen werden. Da fertige Editorfunktionen im TIA Portal angeboten werden, muss man sich nicht ganz so intensiv mit diesem Thema beschäftigen wie es noch in der Version *Step 7 V5.x* der Fall war. Dieses Thema soll die nötigen Grundlagen dazu liefern.

► Kapitel 9: Zähler in *SCL*

Die Zählerfunktionen gehören ebenfalls zu den Basisfunktionen und sollen in diesem Kapitel die nötigen Schritte zeigen, Zähler in *SCL* anzuwenden. Dieses Thema soll die nötigen Grundlagen dazu liefern.

► Kapitel 10: Sprungbefehle und Unterprogramme

Da es im TIA Portal im *SCL*-Programm keine Definition der Sprunglabel mehr gibt, wird im Hinblick auf die später besprochene Statusmaschine dieses Thema einem eigenen Kapitel zugeordnet. Konstanten und Sprünge sind auch wesentliche Voraussetzungen, Objekt-

Funktionen aufzurufen und werden uns später aus der Betrachtung der objektorientierten Programmierung sehr dienlich sein.

► Kapitel 11: Multiinstanzen

In der *OOD* gibt es Konstruktionen, die unter anderem auch Multiinstanzen zum Einsatz bringen. Diese Anwendungen werden zunächst in diesem Kapitel neutral betrachtet und kommen dann später noch einmal auf den Tisch, wenn es darum geht, *Assoziationen* zu programmieren.

► Kapitel 12: Übungen mit *SCL* (S7-1500)

Das nun so erlernte Wissen über *SCL* soll in einigen Beispielen trainiert werden. Damit dies auch eine vertiefte Wirkung zeigt, beschäftigen wir uns mit der prozeduralen Programmierung verschiedener Beispiele, wie Flankenprogrammierung, Analogwerte schreiben und lesen sowie ein wenig Textverarbeitung.

► Kapitel 13: Übungen mit traditionellem Programmierstil

Am Beispiel der Statusmaschine und dessen Automatisierung soll eine komplette Abhandlung in *SCL* umgesetzt werden. Diese schon anspruchsvolle Aufgabe zeigt einen typischen Anwendungsfall, welcher in anderen Hochsprachen wie *C++* schon selbstverständlich ist.

Teil III – Einführung in *UML*

Das Werkzeug *UML* ist für mich als Automatisierer ganz klar ein unabdingbares Produkt und ich bin froh, ein mir besonders am Herzen liegendes Tool vorstellen zu können. Es handelt sich dabei um Enterprise Architect (*EA*). Die behandelten Themen zeigen nur einen geringen Umfang der Möglichkeiten mit diesem Tool. Sie als Automatisierer können sich später weiterbilden, wenn ihnen daran liegt, mehr über *UML* zu erfahren. In jedem Fall können wir mit diesem Tool alles Notwendige erledigen, um unsere Automatisierung zu planen und zu dokumentieren, ohne Bedenken oder sogar Angst zu haben, das Thema *UML* zu intensiv studieren zu müssen.

► Kapitel 14: Einführung in Enterprise Architect

Wir werden ähnlich wie beim Framework TIA Portal einen Einstieg über einen so genannten Rundweg finden. Sie lernen die wichtigsten Diagramme aus Sicht der Automatisierung für das TIA Portal. Damit wir gezielt vorgehen können, beginnen wir danach mit der Erstellung eines einfachen *Deployment*-Diagramms (Verteilungsdiagramm).

► Kapitel 15: Erstellen der Klassendiagramme

Für den Anwender und Programmierer der *SPS* ist der Begriff Klasse nicht bekannt und in *SCL* eigentlich sogar fehl am Platz. Trotzdem werden wir in gewisser Weise Quasi-Klassen mit *FBs* erstellen und auch so anwenden. Dazu dienen die Klassendiagramme und deren Philosophie.

► Kapitel 16: Komponentendiagramme

Unser Programm muss geplant werden. Komponenten zeigen das grobe Zusammenspiel der Softwarebausteine, wie *Assoziation*, *Aggregation* und *Komposition*. Auf einfache Art und Weise können wir so unser Vorhaben beschreiben und die Zusammenhänge mit dem *OB1* darstellen. Das Kapitel zeigt auch einen kleinen Ausschnitt zum Modulieren mit Komponenten.

► Kapitel 17: Beispiele und Training mit dem *EA*-Framework

Abschließend zum Thema *UML* soll ganz gezielt ein schmaler Bereich für unser Vorhaben hervorgehoben und trainiert werden. Es geht darum, die so bekannten Strukturen aus dem *UML* in das Konzept der *SPS* mit dem bestehenden *SCL*-Editor umzusetzen. Hier sieht der Leser ein schönes Kran-Beispiel aus der Praxis und lernt, wie man mit den Diagrammen eine Software planen kann.

Teil IV – *OOP* mit *SCL*

Viele Missverständnisse können entstehen, wenn dieses Thema mit *SPS*-Automatisierern diskutiert wird. Das ist mit einer *SPS* nicht möglich, ist oft die Antwort. Wir wollen keine Probleme suchen, sondern Lösungen, und stehen deswegen *OOP* mit *SCL* positiv gegenüber. Dieser Abschnitt zeigt die Umsetzung der genannten Strukturen aus der *UML* in die Programmiersprache *SCL* im Zusammenwirken mit *FUP* im *OB1*. Ein völlig neuer Weg – für die meisten zumindest. Für *C++*-Programmierer oder überhaupt Programmierer, die den Umgang mit höheren, objektorientierten Sprachen kennen, stellt das hier Gezeigte nur eine etwas andere Form dar, da der *SCL*-Editor nicht die Voraussetzung mitbringt, objektorientiert zu übersetzen. Das was der Editor nicht kann, müssen wir zu Fuß selbst erstellen. Letztendlich können wir so arbeiten wie ein *C++*-Programmierer – besser gesagt, fast so!

► Kapitel 18: Der *FB* als Klasse

Die Eigenschaften einer Klasse mit deren Daten und Beziehungen zur Außenwelt werden in einen *FB* verlagert. Das ist die Basis, um überhaupt objektorientierte Beziehungen aufzubauen. Hier wird die grundsätzliche Struktur für eine quasi Klasse als *FB* erarbeitet. Dazu gehört auch die Behandlung über die Verwaltung von dynamischen Speicher.

► Kapitel 19: Beziehungen programmieren

Nachdem das Objekt in Form einer Art Klasse in einem *FB* zur Verfügung steht, geht es uns darum, die Beziehungen der Objekte zu programmieren. *Aggregation*, *Komposition* und normale *Assoziation* sind in diesem Kapitel Thema und sollen in *SCL* umgesetzt werden. Hier entstehen Regeln für die Umsetzung der *OOP* in eine *SCL*-Struktur, welche im weiteren Buchverlauf immer wieder angewendet werden.

► Kapitel 20: Klassen ableiten

Der *FB* bietet die Anwendung von Multiinstanzen und soll nun Kinder bekommen. Die so genannte direkte Ableitung eines *FB*s ist eine wichtige Voraussetzung für die Wiederver-

wendbarkeit von Software. Auch wenn dies in *SCL* nicht so elegant wie in anderen Hochsprachen zu lösen ist, stellt die Multiinstanz eine interessante Alternative für objektorientiertes Design dar.

► Kapitel 21: *Polymorphie* mit Objekten

Ein sehr schwieriges Thema ist die *Polymorphie*. Die spezielle Anwendung der Ableitung wird nun objektorientiert betrachtet und angewendet. Je nach Aufgabenstellung kann die *Polymorphie* ein wichtiges Werkzeug für die moderne Software-Architektur sein. Ihre Anwendung ist in der Welt der *SPSen* so gut wie nicht bekannt. Hier wird jedoch beispielhaft gezeigt, wie das scheinbar Unmögliche in die Tat umgesetzt werden kann.

► Kapitel 22: Statusmaschine / Zustandsautomat

Wer kennt sie nicht? In der Automatisierung ist die Statusmaschine auch unter der „CASE“-Anweisung oft realisiert. Hier allerdings soll die Statusmaschine als Zustandsautomat nach dem *Design-Pattern* „Zustand“ (*State*) angewendet werden. Dieses Kapitel zeigt die Anwendung einer abstrakten Klasse und die Vorgehensweise, wie ein objektorientiertes Projekt aus dem *UML*-Plan in *SCL* umgesetzt wird.

► Kapitel 23: Entwurfsmuster / *Design-Patterns*

Nachdem nun hoffentlich die Akzeptanz zur objektorientierten Programmierung erreicht wurde, ist es Zeit, ein wenig die *Design-Patterns* für den Einsatz in *SCL* zu betrachten. Es geht hier um eine typische, ausgesuchte Anwendung, die für die Automatisierung sicherlich sehr interessant ist. Zumindest festigt das letzte Kapitel die Verwendung der im Buch gezeigten Techniken und soll den Leser anregen, auch eigene *Design-Patterns* zu entwickeln.

Trial-Versionen für das TIA Portal

Ein „**SIMATIC STEP 7 inkl. PLCSIM V14 SP1 TRIAL**“ im Totally Integrated Automation Portal (TIA Portal) ist 21 Tage nutzbar und steht als Download zur Verfügung:
https://support.industry.siemens.com/cs/de/de/view/109745153

Ein „**SINAMICS Startdrive V14 SP1 TRIAL**“ im Totally Integrated Automation Portal (TIA Portal) ist 21 Tage nutzbar und steht als Download zur Verfügung:
https://support.industry.siemens.com/cs/de/de/view/68034568

Weitere Informationen zum TIA Portal finden Sie unter:
http://support.automation.siemens.com/WW/view/de/65601780

Weitere Informationen erhalten Sie im Internet unter:
«https://www.siemens.de/sce/promotoren»
«https://www.siemens.de/sce/S7-1500»
«https://www.siemens.de/sce/tp»
«https://www.siemens.de/sce»