Stefan Henneken

Anwendung der SOLID-Prinzipien mit der IEC 61131-3

5 Prinzipien für objektorientiertes Softwaredesign in der SPS-Programmierung

Stefan Henneken

Anwendung der SOLID-Prinzipien mit der IEC 61131-3

5 Prinzipien für objektorientiertes Softwaredesign
in der SPS-Programmierung

Impressum

Bibliografische Information der Deutschen Nationalbibliothek:
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.dnb.de abrufbar.

2. leicht überarbeitete Auflage November 2023

Herstellung und Verlag: BoD – Books on Demand, Norderstedt

ISBN: 978-3-7578-7070-6

1 Inhaltsverzeichnis

2 Vorwort

Die SOLID-Prinzipien sind ein wesentlicher Bestandteil der objektorientierten Softwareentwicklung und haben sich als wertvolle Werkzeuge erwiesen, um sauberen, wartbaren und erweiterbaren Code zu entwickeln. In der industriellen Automatisierungstechnik, insbesondere in der Programmierung von Steuerungen mit IEC 61131-3, ist es von besonderer Bedeutung, robuste und zuverlässige Systeme zu entwickeln.

In diesem Buch werden die SOLID-Prinzipien im Detail vorgestellt und anhand von Beispielen in IEC 61131-3 erläutert. Es wird auch verdeutlicht, wie durch die Anwendung dieser Prinzipien die Wartbarkeit, die Erweiterbarkeit und die Zuverlässigkeit von Softwaresystemen verbessert wird.

Zusätzlich zu den SOLID-Prinzipien werden auch die Prinzipien KISS, DRY, LoD und YAGNI vorgestellt. Diese zählen zwar nicht zu der Gruppe der SOLID-Prinzipien, sind zu diesen aber eine hilfreiche Ergänzung.

Ich hoffe, dass dieses Buch für alle, die in der industriellen Automatisierungstechnik tätig sind, von Nutzen sein wird und dass es Ihnen dabei helfen kann, bessere und zuverlässigere Systeme zu entwickeln. Vielen Dank, dass Sie sich für unser Buch entschieden haben und viel Erfolg bei der Anwendung der SOLID-Prinzipien in Ihrer Arbeit!

Stefan Henneken

3 SOLID - Fünf Grundsätze für bessere Software

Neben der Syntax einer Programmiersprache und dem Verständnis der wichtigsten Bibliotheken und Frameworks, gehören weiterer Methodiken – wie zum Beispiel Design Pattern – zu den Grundlagen der Softwareentwicklung. Neben den Design Pattern sind Designprinzipien ebenfalls ein hilfreiches Werkzeug bei der Entwicklung von Software. SOLID ist ein Akronym für fünf solcher Designprinzipien, die dem Entwickler dabei unterstützen Software verständlicher, flexibler und wartbarer zu entwerfen.

In größeren Softwareprojekten existiert eine Vielzahl von Funktionsblöcken, die über Vererbung und Referenzen miteinander in Verbindung stehen. Durch die Aufrufe der Funktionsblöcke und deren Methoden agieren diese Einheiten untereinander. Dieses Zusammenspiel der Codeeinheiten, kann bei falschem Design das Erweitern oder Auffinden von Fehlern unnötig erschweren. Für die Entwicklung von nachhaltiger Software sollten die Funktionsblöcke so modelliert werden, damit diese einfach zu erweitern sind.

Viele Design Pattern wenden die SOLID-Prinzipien an, um für die jeweilige Aufgabenstellung einen Architekturansatz vorzuschlagen. Die SOLID-Prinzipien sind auch nicht als Regeln zu verstehen, sondern mehr als Ratschläge. Sie sind eine Untermenge vieler Prinzipien, die der amerikanische Software-Ingenieur und Dozent Robert C. Martin (auch bekannt als Uncle Bob) in seinem Buch *Clean Architecture: Das Praxis-Handbuch für professionelles Softwaredesign* vorgestellt hat. Die SOLID-Prinzipien sind im Einzelnen:

- **S**ingle Responsibility Principle (SRP)
- **O**pen/Closed Principle (OCP)
- **L**iskov Substitution Principle (LSP)
- **I**nterface Segregation Principle (ISP)
- **D**ependency Inversion Principle (DIP)

Die hier gezeigten Prinzipien sind Hinweise, die es einem Entwickler erleichtert die Codequalität zu verbessern. Der Aufwand amortisiert sich nach kurzer Zeit, da Änderungen einfacher, Tests und Fehlersuche beschleunigt werden. Somit sollte das Wissen über diese fünf Designprinzipien zur Basis eines jeden Softwareentwicklers gehören.

3.1 Single Responsibility Principle (SRP)

Ein Funktionsblock sollte nur eine einzige Verantwortung haben. Wird die Funktionalität eines Programms geändert, sollte dieses nur Auswirkungen auf wenige Funktionsblöcke haben. Viele kleine Funktionsblöcke, sind besser als wenige große. Der Code wirkt auf dem ersten Blick zwar umfangreicher, ist dadurch aber einfacher zu organisieren. Ein Programm mit vielen kleineren Funktionsblöcken, für jeweils spezielle Aufgaben, ist einfacher zu pflegen, als wenige große Funktionsblöcke, die den Anspruch erheben, alles zu können.

3.2 Open/Closed Principle (OCP)

Nach dem *Open/Closed Principle* sollten Funktionsblöcke offen für Erweiterungen, aber geschlossen für Änderungen sein. Die Umsetzung von Erweiterungen sollte nur durch Hinzufügen von Code, nicht durch Ändern von vorhandenen Code erreicht werden. Ein gutes Beispiel für dieses Prinzip ist die Vererbung. Ein neuer Funktionsblock erbt von einem schon vorhandenen Funktionsblock. Neue Funktionen können so hinzugefügt werden, ohne den vorhanden Funktionsblock verändern zu müssen. Es muss nicht einmal der Programmcode vorliegen.

3.3 Liskov Substitution Principle (LSP)

Das *Liskov Substitution Principle* fordert, dass abgeleitete Funktionsblöcke immer anstelle ihrer Basis-FBs einsetzbar sein müssen. Abgeleitete FBs müssen sich so verhalten wie ihr Basis-FB. Ein abgeleiteter FB darf den Basis-FB erweitern, aber nicht einschränken.