Qualitätssicherung durch statische Code-Analyse

Wenn der Bug gar nicht erst auf die Maschine kommt

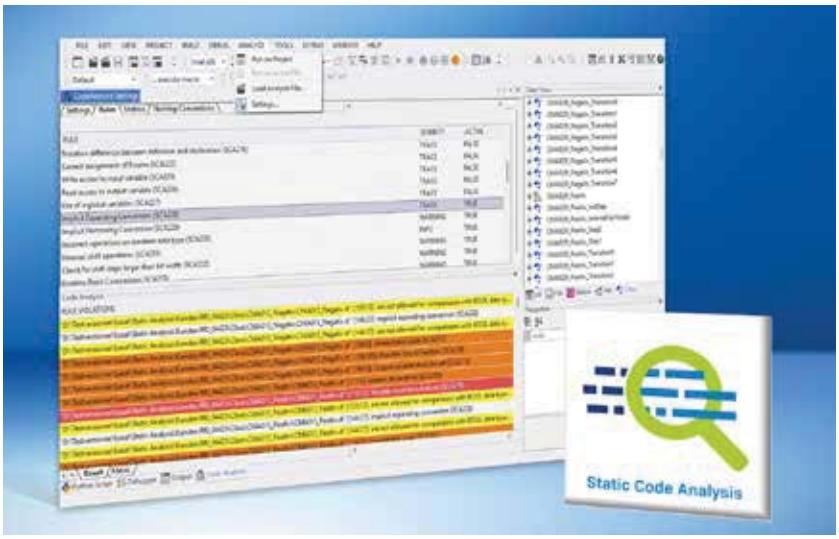
Steuerungs-Software wächst – und mit ihr die Wahrscheinlichkeit, dass sich kleine Unsauberkeiten als große Stillstandsursachen entpuppen. Statische Code-Analyse prüft Structured-Text-Programme, bevor sie laufen, und macht Risiken wie undefiniertes Verhalten sichtbar. Warum das für OEMs und Betreiber zunehmend zum Effizienzhebel wird – und wie sich die Methode pragmatisch in Engineering-Prozesse integrieren lässt.

TEXT: Christian Vilsbeck, A&D **BILDER:** Sigmatek; iStock, ATHVisions

Software ist in der Automatisierung längst kein „Add-on" mehr, sondern Teil der Maschinenfunktion selbst. Gleichzeitig wird der Codeumfang größer: mehr Varianten, mehr Optionen, mehr Schnittstellen, mehr Safety-Logik, mehr Diagnose. Unter diesem Druck entstehen Fehler oft nicht spektakulär, sondern leise – als nicht initialisierte Variable, als implizite Typkonvertierung, als ungünstige Lebensdauer eines Zeigers oder als Randfall, der erst beim Kunden auftritt. Genau dort sind Bugs am teuersten, weil sie Inbetriebnahme, Verfügbarkeit und Servicefähigkeit beeinflussen.

Früher prüfen statt später suchen

Statische Code-Analyse (Static Code Analysis, SCA) verschiebt die Qualitätsprüfung nach vorn. Anders als dynamische Tests bewertet sie den Quellcode ohne Ausführung und sucht nach Mustern, die auf mögliche Laufzeitfehler, undefiniertes Verhalten oder Verstöße gegen definierte Regeln hinweisen. Für Teams, die IEC 61131 3-Programme in Structured Text entwickeln, ist das besonders interessant: Viele Probleme sind formal erkennbar, aber im laufenden Prozess nur schwer



Die SCA prüft den Code auf Laufzeitfehler und undefiniertes Verhalten, etwa nicht initialisierte Variablen, ungültige Pointer-Rückgaben oder implizite Konvertierungen.

zu reproduzieren. SCA wirkt hier wie ein Frühwarnsystem, das Standardfehler konsequent abräumt, bevor sie sich in einer Anlage „einbrennen".

Wie sich so ein Ansatz in der SPS-Praxis anfühlt, lässt sich gut an einer konkreten Implementierung ablesen: Im Engineering Tool Lasal Class von Sigmatek gibt es eine optionale Erweiterung „Statische Code Analyse", die Structured Text Steuerungscode untersucht, ohne dass der Code ausgeführt werden muss. Der Anbieter beschreibt typische Prüffälle wie nicht initialisierte Variablen, die Retournierung eines Pointers auf eine lokale Variable oder implizite Konvertierungen. Entscheidend ist dabei, dass die Analyse nicht erst am Ende eines Projekts ansetzt, sondern als wiederholbarer Schritt im Engineering verfügbar ist.

Die Methodik ist aus der IT-Welt bekannt, passt aber zunehmend in die OT-Realität. Denn Steuerungssoftware ist heute oft modular, objektorientiert und über Jahre im Einsatz. Genau diese Langfristigkeit macht Konsistenz und Wartbarkeit zum Produktionsfaktor: Wer bei Updates, Retrofits oder Variantenwechseln ständig an unübersichtlichem Code nacharbeiten muss, zahlt mit Zeit, Risiko und am Ende auch mit OEE. In integrierten Umgebungen wie der Lasal-Suite, in der Control, Visualisierung, Motion und Safety zusammengeführt sind, wird Code zudem häufiger über Team- und Disziplingrenzen hinweg geteilt. Das erhöht den Druck, Standards nicht nur zu definieren, sondern sie auch verlässlich zu prüfen.

SCA ist dabei kein Ersatz für Simulation, Integrationstests oder die Inbetriebnahme an der Maschine. Ihre Stärke liegt in der systematischen Breite: Sie prüft immer gleich, vergisst nichts und wird nicht müde. Damit entlastet sie Reviews, weil sich menschliche Prüfer stärker auf Logik, Architektur und Grenzfälle konzentrieren können. In Lasal Class ist das explizit als Ziel formuliert: Automatisierte Basisprüfungen sollen Reviewer von Routinefunden entlasten. Gleichzeitig können Metriken Hinweise darauf liefern, wo Komplexität wächst und Refactoring sinnvoll ist – nicht als „Gefühl", sondern als messbarer Trend.

Vom Regelwerk zur täglichen Engineering-Routine

Entscheidend ist weniger, dass „eine Analyse existiert", sondern wie sie konfiguriert und genutzt wird. In der Lasal-Class-Erweiterung laufen viele Prüfregeln parallel und markieren Qualitätsmängel, Richtlinienverstöße oder Abweichungen von internen Coding-Standards früh. Praktisch relevant sind dabei die Stellschrauben: Laut Sigmatek stehen über 50 vordefinierte Regeln für Variablen, Typen, Funktionen und Methoden bereit, die einzeln parametriert werden können. Hinzu kommen konfigurierbare Naming Conventions sowie die Möglichkeit, Metriken zu berechnen. Regeln, Metriken und Namenskonventionen lassen sich individuell oder gruppenweise ein- und ausschalten – ein Detail, das die Einführung in gewachsenen Codebasen erleichtert.

Gerade in der Automatisierung kommt es selten zu „Greenfield"-Software. Wer eine Bestandsanlage modernisiert oder Funktionsbibliotheken über Jahre pflegt, braucht einen pragmatischen Einstieg: erst die Regeln, die wirklich Ausfallrisiken adressieren, dann schrittweise strengere Stil- und Strukturvorgaben. Eine SCA, die wie in Lasal Class flexibel parametrierbar ist, kann diese Reifegrade abbilden. Gleichzeitig bleibt das Thema Signal-Rausch-Verhältnis: Wenn Warnungen bewusst toleriert werden, muss das nachvollziehbar dokumentierbar sein. In Lasal Class ist dafür vorgesehen, Meldungen zeilenweise per Code-Kommentar zu unterdrücken – also dort, wo die Entscheidung entstanden ist.

Auch die Bedienlogik zählt. Die Lasal-Class-Analyse wird in einem eigenen Ausgabefenster angezeigt; per Doppelklick springt der Entwickler zur betroffenen Codezeile. Prioritäten sind farblich unterscheidbar, um kritische

Mit SCA lassen sich triviale, schwer auffindbare Fehler vermeiden – das entlastet Teams und steigert Produktivität sowie Softwarequalität in Entwicklung und Wartung.

Funde schneller zu triagieren. Solche Details entscheiden in der Praxis darüber, ob Analyseergebnisse tatsächlich konsequent abgearbeitet werden oder im Projektalltag untergehen.

Integrierte Toolketten und der „Shift Left"-Gedanke

SCA entfaltet den größten Effekt, wenn sie nicht als Einmal-Aktion vor der Auslieferung läuft, sondern als wiederkehrender Schritt im Engineering-Prozess. Das gelingt besonders gut in integrierten Toolketten, weil Ergebnisse dort bleiben, wo Entwickler ohnehin arbeiten. Die Lasal-Suite von Sigmatek ist als durchgängige Engineering-Umgebung konzipiert, die Control, Visualisierung, Motion und Safety kombiniert und objektorientiertes Programmieren gemäß IEC 61131 3 unterstützt. In so einem Setup kann SCA als Qualitätsstufe in Builds, Bibliotheksfreigaben oder Variantenableitungen mitlaufen – und nicht als separate „Abnahmeprüfung" am Ende. Für Betreiber und Instandhalter zahlt sich das indirekt aus. Je sauberer die Codebasis, desto schneller lassen sich Ursachen eingrenzen, Updates durchführen und Änderungen dokumentieren. Wenn eine SCA wie die Lasal-Class-Erweiterung zusätzlich Metriken bereitstellt, können Teams Komplexitätsanstiege früh erkennen, bevor Wartbarkeit kippt. Und wenn Regeln und Naming Conventions konsistent angewendet werden, sinkt das Risiko, dass sich bei Personalwechseln oder über mehrere Maschinenlinien hinweg stillschweigend unterschiedliche „Dialekte" im Code etablieren.

Fazit: SCA wird zur Basishygiene industrieller Software

Die zentrale Entwicklung ist klar: Maschinen werden softwarelastiger, Varianten werden zahlreicher, und der Zeitdruck bleibt hoch. Statische Code-Analyse adressiert dabei keinen „Trend um des Trends willen", sondern ein dauerhaftes Problem der Industrie: die Kosten von Fehlern, die zu spät gefunden werden. Als Ergänzung zu Tests und Reviews bringt SCA Wiederholbarkeit und Messbarkeit in die Codequalität. Der Nutzen entsteht vor allem dann, wenn SCA in der täglichen Engineering-Umgebung verankert ist – so wie es bei integrierten Erweiterungen à la Lasal Class der Fall ist.

Technologisch passt das zum breiteren Shift Left Gedanken in der Automatisierung: Qualitäts-, Safety- und zunehmend auch Security-Anforderungen wandern in frühere Phasen, weil spätere Korrekturen zu teuer sind. In diesem Sinne dürfte SCA in vielen Organisationen zur Basishygiene werden – als pragmatischer Baustein, der die wachsende Softwarekomplexität in der Produktion beherrschbarer macht.