
Automatisierung in der Embedded-Software-Entwicklung:

# Continuous Integration

**Der Zielvorstellung einer weitgehend automatisierten Software-Entwicklung kommt man mit Continuous Integration ein ganzes Stück näher. Richtig konfiguriert, kann sie zudem viele der ungeliebten Testaufgaben übernehmen. Um Continuous Integration in Embedded-Projekten einzuführen, sind aber viele Schritte nötig.**

Von Dr. Frank Ziesel

Gestiegene Qualitätsanforderungen für die Embedded-Software-Entwicklung stellen viele Unternehmen vor große Herausforderungen. Vor allem in Projekten, in denen die funktionale Sicherheit des fertigen Steuergeräts nachgewiesen werden muss, sind die Bedingungen nicht mehr mit konventionellen Entwicklungsprojekten zu vergleichen. Die Nachweispflicht über durchgeführte Qualitätsmaßnahmen, Nachvollziehbarkeit von Anforderungen bis in den Code und häufigere Release-Zyklen bringt nicht nur mehr Aufwand, sondern auch einige Stolpersteine in Embedded-Software-Projekten mit sich. Eine Automatisierung von Teilen des Software-Entwicklungsprozesses kann das Arbeiten in solchen Projekten enorm erleichtern, sowie das Ausgangsprodukt qualitativ auf ein einheitliches Level anheben.

Der sich seit Jahren fortsetzende Trend, immer mehr Steuergerätefunktionen von der Hardware auf die Software zu verlagern, führt generell dazu, dass der Fokus bei Entwicklungsprojekten immer mehr auf der Software-Entwicklung liegt. Dies erhöht nicht nur die Komplexität der Software, sondern auch die Anzahl der Software-Auslieferungen in den Projekten.

## Hohes Automatisierungspotenzial in der Software-Entwicklung

Die klassische Entwicklung nach dem V-Modell ist unter anderem die Grundlage für die Entwicklung von Steuergeräten nach sicherheitsgerichteten Normen (z.B. ISO 26262 oder ISO 13849). In der Software-Entwicklung muss in solchen Projekten vermehrtes Augenmerk auf die Durchführung eines dokumentierten Ablaufs des Software-Entwicklungsprozesses gelegt werden. Bei der Betrachtung des Automatisierungspotenzials kann der Bogen jedoch noch etwas weiter gespannt werden. So ist es möglich, prozessbegleitende Aufgaben zu unterstützen oder sogar vollumfänglich zu automatisieren. Hierzu zählen zum Beispiel Software-Qualitätsanalysen oder die Ausführung von

Modultests. Des Weiteren sind von der Hardware-Software-Integration bis hin zu HIL-Tests (Hardware in the Loop) für bestimmte Systeme automatisierbar. **Bild 1** zeigt einen Überblick über das Potenzial der Automatisierung im Software-Entwicklungsprozess und darüber hinaus.

## Contineous Integration und Contineous Delivery

Die Zielvorstellung für eine weitestgehend automatisierte Software-Entwicklung ist, dass der Software-Entwickler bereits mit dem Implementieren einer neuen Funktion eine Maschinerie in Gang setzt, die alle folgenden Schritte automatisiert durchführt. In **Bild 2** ist ein möglicher Durchlauf für ein Embedded Continous Delivery System dargestellt, dessen Herzstück der Continuous Integration Server (CI-Server) bildet.

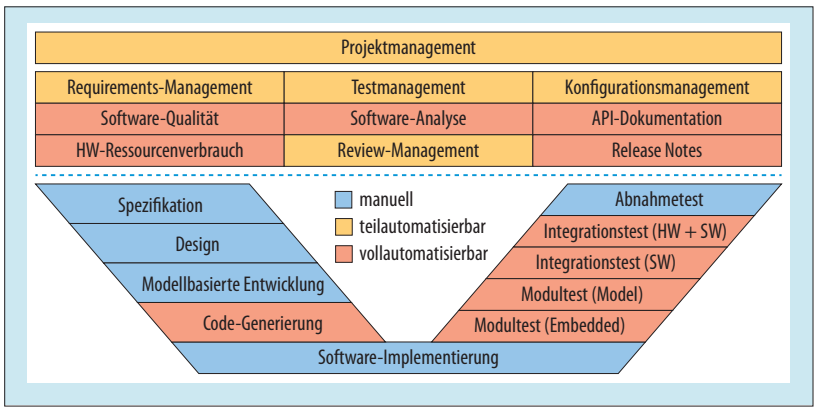


**Bild 1. Bei der Software-Entwicklung können viele Aufgaben automatisiert werden, vor allem im Bereich des Testens.** (Quelle: Gigatronik)

Die Software-Änderung wird in einem Tool-gestützten Review freigegeben und leitet so den nachfolgenden Schritt ein. Mit synchron aktualisierten Modultests werden die geänderten Module gegen die Anforderungen getestet. Die hieraus entstehenden Ergebnisse werden für die spätere Auslieferung archiviert. Ein erfolgreiches Durchlaufen der Modultests macht den Weg frei für die Software-Integration, bei der die Firmware für das Steuergerät erstellt wird. Diese Firmware ist ein Endprodukt der Software-Entwicklung und kann nun an die Testabteilung für manuelle Tests ausgeliefert oder durch weitere Automatisierungssysteme verarbeitet werden. Ein automatisierter Flash-Vorgang ist hierzu der Beginn eines HiL-Tests. In diesem werden die Gesamtfunktionen des Steuergeräts unter simulierten Umwelteinflüssen getestet. Diese Tests können bei Bedarf als Abnahmekriterium für ein Projekt definiert werden, wodurch ein erfolgreiches Durchlaufen eine Auslieferung der Software ermöglicht.

In der Realität wird jedoch kaum ein Projekt im Embedded-Umfeld mit mehreren Software-Ständen täglich in dieser Automatisierungsstufe ausgeliefert. Der Vorteil besteht in solchen Systemen darin, dass das abgelegte Firmware-Paket immer alle nötigen Liefergegenstände, sowie alle Abnahmekriterien und Qualitätsmerkmale enthält.

Was in der IT-Welt als „Continuous Delivery" bezeichnet wird, ist in der Embedded-Welt nicht ganz analog zu gewährleisten. Die Integration und Testdurchführung auf einer realen Hardware ist mit einigen Fallstricken verse-

hen, die es im Test-Set-up zu bewältigen gibt. Eine reproduzierbare Initialisierung des Gesamtsystems ist z.B. meistens mit einem kompletten Neustart des Steuergeräts verbunden, was die Testausführung deutlich verlängern kann. Des Weiteren sind die zentralisierte Anbindung der benötigten Tools und die Behandlung von Tool-Fehlern nicht immer einfach.

Auch bei komplett automatisierten Systemen müssen sich Experten um die Wartung sowie Instandhaltung der Testplattformen und der Automatisierungsserver kümmern. Je nach Komplexität der Build-Prozesse oder Aufbauten kann dieser Betreuungsaufwand stark variieren. Bei einfachen Systemen kann dies jedoch durchaus von einer Person als Teilzeitaufgabe bewältigt werden.

## Abläufe automatisieren

In den meisten Fällen kommen im Embedded-Bereich kommerzielle Compiler zum Einsatz, die es erfordern, dass eine zusätzliche Lizenz für den CI-Server angeschafft werden muss. Dieser zusätzliche Aufwand wird jedoch schnell durch die Ersparnis an Entwickler-Arbeitszeit kompensiert. Dies gilt nicht nur für Compiler, sondern auch für Code-Generatoren, Test-Tools und Diagnose-Tools die in einem üblichen Testablauf verwendet werden.

Die Toolchains für den Embedded-Bereich sind nicht immer für eine automatisierte Ausführung optimiert. Während bei Java zum Beispiel eine sehr breite Unterstützung für das gängigste Build-Management-Tool (Maven) existiert, sind im Embedded-Bereich meistens Skript-basierte Ausführungen der jeweiligen Build-Konfiguration (Makefiles) die Regel. Diese etwas aufwendigere Konfiguration stellt jedoch keinerlei Hindernis dar, benötigt aber einen höheren Konfigurations- und Pflegeaufwand.

Gleiches gilt für die Ausführung und Konfiguration von Testaufträgen, die oft spezifische Test-Tools für die Embedded-Anwendung verwenden (z.B. Tessy für Modultests auf dem Zielsystem). Tools dieser Art sind vor allem für Projekte mit hohen Qualitätsanforderungen notwendig, um die erreichte Testabdeckung auf der Zielplattform zu erreichen. Andererseits können auch allgemeine Test-Frameworks (Cunit, CPPunit oder Google Test in der PC-Umgebung) konfiguriert und verwendet werden. Diese bieten sich vor allem für einfache Projekte an, bei denen keine Cross-Compilierung gefordert wird.

Die Möglichkeit des CI-Systems Testergebnisse und Reports darzustellen, lässt sich ebenfalls auf Embedded-Projekte anwenden. Manchmal erfordert dies, eine Konvertierung des Tool-spezifischen Reports in bestimmte gängige Formate (XUnit) um eine Darstellung, Nachverfolgung oder sogar E-Mail-Benachrichtig abhängig von Testergebnissen zu ermöglichen.

Speziell für Embedded-Projekte ist die genaue Betrachtung des Ressourcenverbrauchs ein sinnvolles Feature für ein CI-System. Da verfügbarer Arbeits- und Flash-Speicher immer limitiert ist, muss eine Überwachung des Ressourcenverbrauchs durch das

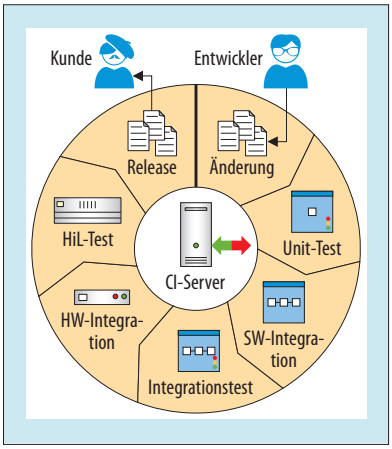


**Bild 2. Embedded Contineous Delivery: Eine Freigabe oder ein erfolgreich durchlaufener Test lösen den nächsten Schritt im Entwicklungsprozess aus.** (Quelle: Gigatronik)

CI-System erfolgen sowie ein Ressourcenmanager auf die plangemäße Verteilung der Ressourcen achten.

Die Ausführung von Tools zur Software-Analyse (MISRA-Checker oder Bug-Finder) ist ebenfalls ein sinnvoller Anwendungsfall für ein CI-System. Eine kontinuierliche Überwachung der Software-Qualität ist notwendig, um parallel zum Entwicklungsfortschritt eine hohe Software-Qualität zu gewährleisten. Auftretende Probleme können schnell erkannt und noch während der Entwicklungsphase eines Features behandelt werden. Dies reduziert nicht nur den Beseitigungsaufwand, sondern führt langfristig auch zu einer Verbesserung der abgelieferten Software. Die Entwickler verinnerlichen die Codierungsrichtlinien beim direkten Bezug auf ihre tägliche Arbeit deutlich besser.

Durch die Zentralisierung zahlreicher kostenpflichtiger Tools auf das CI-System können in manchen Fällen auch die gesamten Entwicklungskosten reduziert werden. Es kann eine Lizenz für das CI-System angeschafft werden, die durch ständige Ausführung dem gesamten Entwicklungsteam aktuelle Daten zur Verfügung stellt. Dieser Effekt ist vor allem bei Code-Generatoren und Analyse-Tools signifikant.

## Versionsverwaltung unverzichtbar

Der erste Build-Job in einem Embedded-Projekt ist meist ein automatisierter Compiliervorgang. Um diesen kontinuierlich auf einem Automatisierungssystem auszuführen, muss der Quellcode in einem Versionsverwaltungssystem abgelegt sein. Dies ermöglicht nicht nur die Verfolgung der Änderungen, sondern auch eine Zuordnung dieser zum verantwortlichen Entwickler. Diese Technologie sollte jedoch für alle Software-Projekte bereits Standard sein. Als nächstes muss eine einheitliche Build-Umgebung geschaffen werden. Dies ermöglicht die Skript-gesteuerte Ausführung des Compilier- und Link-Vorgangs. Mit einer einfachen Konfiguration eines Build-Jobs, welcher auf das Quellcode-Repository zugreift und den Compiliervorgang startet sowie dessen erfolgreiche Ausführung überprüft, ist bereits der wichtigste Schritt für ein Embedded Continuous Integration System geschaffen. Dieser erste Schritt stellt sicher, dass der aktuelle Quellcode im Repository konsistent ist und für die folgenden Schritte verwendet werden kann.

Das fertige Compilat kann nun an die virtuelle Testabteilung weitergeleitet werden. Dieser separate Build-Job beginnt einen Skript-gesteuerten Flash-Vorgang der Ziel-Hardware. Dies kann über implementierte Diagnoseprotokolle (UDS, XCP) erfolgen oder direkt über ein Debug-Interface, je nachdem welches Testlevel der Test abdecken soll. Für White-Box-Software-Tests können die Möglichkeiten eines Debug-Interfaces notwendig sein, wobei HiL-Tests eine Serienschnittstelle benötigen, um das Steuergerät während der Testausführung nicht zu beeinflussen. Ein solcher Black-Box-Test unter simulierten Umwelteinflüssen ist die höchste Teststufe, die mit einem CI-System abgebildet werden kann. Hierzu sind sämtliche Schritte der Testautomatisierung nötig, sowie automatisierte Initialisierungs- und Rücksetz-Mechanismen. Diese ermöglichen es, das Steuergerät auch bei Fehlern oder Testabbrüchen wieder in einen definierten Ausgangszustand zu versetzen.

## Automatische Qualitätssicherung

Sofern Software-Qualität oder Modultests als Evaluierungskriterium verwendet werden sollen, kann diese Aufgabe dem ersten Integrations-Job vorgelagert werden. Dies ermöglicht es, qualitativ nicht ausreichende Software-Änderungen umgehend zu erkennen und eine entsprechende Nacharbeit in Gang zu setzten. Dieser Qualitäts-Job kann statische Analyse-Tools ausführen und die gefunden Abweichungen mit einer definierten Schwelle vergleichen. Zum Beispiel können hierdurch alle Software-Änderungen, bei denen kritische MISRA Verletzungen gefunden werden, zurückgewiesen werden. Somit können projektspezifische Qualitätskriterien definiert werden, welche zu einer kontinuierlichen Verbesserung der Code-Qualität führen.

Bei modellbasierter Entwicklung kommen meist Code-Generatoren zum Einsatz. Ein solches Projekt kann diesen Schritt ebenfalls in einem eigenen CI-Job abarbeiten lassen. Dieser Job kann auch so ausgelegt werden, dass die Modelle vom einem Entwicklungsteam entwickelt werden und das CI-System die generierten Quellcode-Dateien an einer anderen Stelle ins Versionsverwaltungssystem einpflegt. Diese Dateien stehen nun dem gesamten Entwicklungsteam zur Verfügung und das CI-System kann im Folgenden die gesamte Software-Integration durchführen.

## Ausblick für zukünftigen Einsatz

Ein automatisierter Entwicklungsprozess heißt nicht, dass Entwickler ersetzt werden. Vielmehr gibt es den an der Entwicklung beteiligten Personen die Möglichkeit, sich auf ihre Kernaufgabe zu konzentrieren. Die Automatisierung kann die Entwicklung auch nur zu einem bestimmten Teil unterstützen. Die finale Validierung muss natürlich den jeweiligen Vorschriften oder Normen entsprechend gestaltet werden. Ein CI-System hilft auf diesem Weg, das Ziel immer im Auge zu behalten. Es bewirkt, dass die sekundären Entwicklungsziele (Qualität, Testabdeckung, Ressourcenverbrauch etc.) nicht vergessen werden und nicht hinter die primären Entwicklungsziele (Features) gestellt werden können. Des Weiteren werden Abweichungen, Probleme und Missverständnisse frühestmöglich erkannt und kommuniziert. Der digitale Integrator sorgt somit für ein besseres Ergebnis und ist an sämtliche Bedürfnisse anpassbar. Diese Vorteile sind längst auch in Embedded-Projekten erkannt worden. *jk*

**Dr. Frank Ziesel**
leitet bei Akka Gigatronik verschiedene Projekte vom OEM bis zum Mittelständler mit dem Schwerpunkt Embedded Software in unterschiedlichen Brachen.
Frank.Ziesel@akka.eu