

Vem ser till att AI-koden är säker?

AI-verktyg kan skriva kod på sekunder, men i säkerhetskritiska system är det inte själva kodningen som är den stora utmaningen. Det avgörande är att kunna visa att koden uppfyller kraven, följer rätt standarder och kan spåras hela vägen från krav till test.

AI-assistenter för kodning har förändrat vad en ensam utvecklare kan producera på kort tid. Kod som förut tog timmar att skriva genereras nu på några sekunder och produktivitetsvinsterna för utforskande arbete är betydande.

Men för säkerhetskritiska applikationer med krav på certifiering enligt ISO 26262 (fordon), IEC 62304 (medicinteknik) eller IEC 61508 (industri) var det aldrig tiden det tog att skriva koden som var problemet. Det kritiska momentet ligger i att bevisa att koden uppfyller alla krav. Koden ska ha producerats enligt definierade metoder, kontrollerats mot rätt standarder och vara spårbar från kravspecifikation till test. Den del av kodskrivandet som AI-assistenter vanligen accelererar var inte utmaningen. Samtidigt ökar användandet trycket på den del som redan var dyr: verifiering och validering.

AI kan generera kod, men kan inte bevisa regelefterlevnad. Det betyder inte att vi inte ska använda AI i dessa applikationer, men det betyder att utvecklarens roll förändras.



Av Rafael Taubinger, IAR

Rafael Taubinger är Global Product Marketing Manager på IAR, baserad på Sverigekontoret i Uppsala. Med över 20 års erfarenhet inom embedded-branschen har han tidigare varit FAE Manager och Senior FAE på IAR, samt test- och utvecklingsingenjör på Siemens. Han är specialiserad på C/C++-programmering för 8-, 16- och 32-bitars mikrokontroller. Sedan oktober 2025 är IAR ett helägt dotterbolag till Qt Group.



MODERNA AI-VERKTYG kan mycket. De kan föreslå implementeringar, komplettera funktioner och generera tester. Om du ber om en varvtalsberäkning i C får du något som är syntaktiskt korrekt och ofta logiskt sunt. Men den genererade koden garanterar inte regel efterlevnad.

En enkel AI-genererad funktion kan se helt okomplicerad ut vid en första genomläsning. Men en statisk analys som tillämpar MISRA C:2012, regel 10.3, kommer att flagga den implicita typkonverteringen som en avvikelse, en som måste åtgärdas innan koden kan spåras till ett verifierat krav.

Det här mönstret upprepas för varje standard som utvecklingsteam för inbyggda system arbetar mot:

- **MISRA C/C++:** Ett etablerat regelverk för C- och C++-kod i säkerhetskritiska system. Genom att begränsa språket till en väldefinierad delmängd minimeras risken för odefinierat beteende. I takt med att AI-verktyg genererar mer C- och C++-kod blir MISRA-efterlevnad en viktig kontrollpunkt för allt som förs in i kodbasen.
- **CERT C/C++:** Fokuserar på att förhindra kodmönster som angripare utnyttjar. Där MISRA riktar in sig på driftsäkerhet, riktar CERT C/C++ in sig på skydd mot intrång.

Det här är två områden som i allt högre grad överlappar varandra i uppkopplade inbyggda system.

- **CWE:** En katalog över återkommande svagheter i mjukvara. För team som granskar AI-genererad kod ger CWE ett strukturerat begreppssystem för att identifiera sårbarheter som en modell kan introducera eller reproducera i genererad kod.

Ingen av dessa standarder kan tillämpas tillförlitligt av AI-modeller. Att se till att de uppfylls är utvecklarens ansvar, med stöd av rätt verktyg.

VERIFIERING OCH VALIDERING utgör redan 40 procent eller mer av FoU-budgeten i säkerhetskritiska program. Det beror inte på ineffektivitet, utan på vad det kostar att bygga den beviskedja som tillsynsmyndigheter och certifieringsorgan kräver.

Vad händer när AI accelererar kodskrivandet men lämnar beviskedjan oförändrad? Mer kod flödar in i verifieringspipelinen. Flaskhalsen fylls. Säkerhetsansvarig får mer att granska, men tröskeln för att godkänna koden sänks inte.

Det som faktiskt tar tid har alltid varit statisk analys, dynamisk testning, täcknings-

mätning, spårbarhet och godkännande. Om kodskrivandet går snabbare men resten av processen förblir densamma, blir det ingen produktivitetsvinst för ett team som levererar certifierad programvara. I stället ökar trycket på en process som redan är hårt belastad.

En lösning på det här problemet är att göra statisk analys, dynamisk analys och täckningsmätning till en del av det dagliga utvecklingsarbetet, inte en granskning som kommer efteråt utan ett kontinuerligt och integrerat moment. Ett bra arbetsflöde skiljer inte på kodgenerering och efterlevnadskontroll, utan låter dem gå hand i hand.

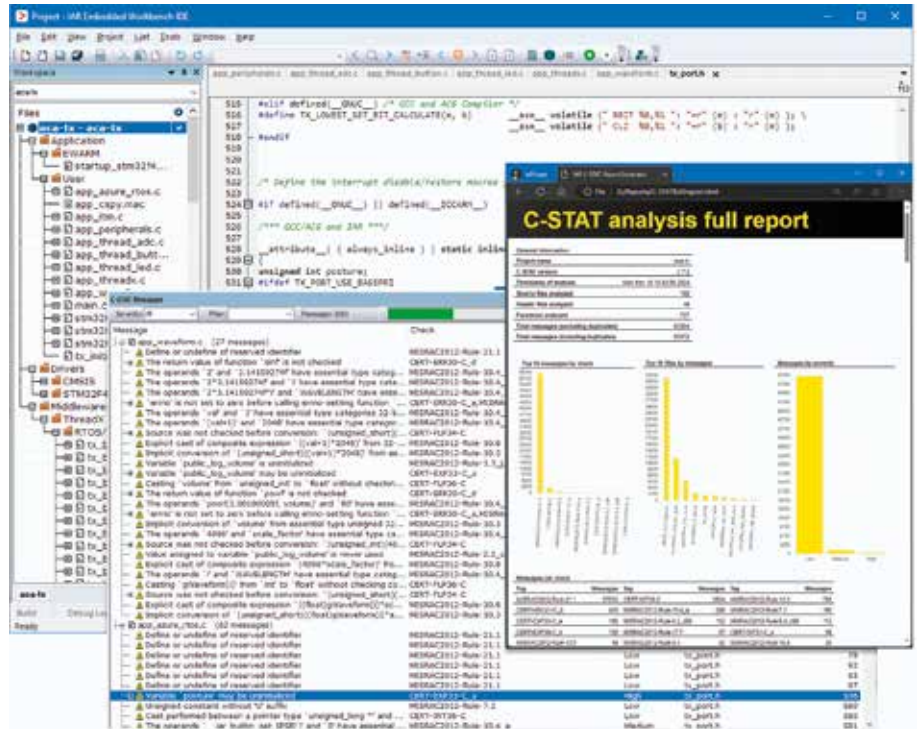
Ett analysverktyg som körs när koden skrivs kan flagga avvikelser från MISRA C, MISRA C++, CERT C och CWE innan koden når en granskningsgrupp eller en certifieringsrevision. AI föreslår, utvecklaren granskar, analysverktyget verifierar.

STATISK ANALYS kan upptäcka många fel utan att programmet körs, men inte alla problem kan avgöras enbart från koden. Vissa fel visar sig först under specifika körförhållanden, exempelvis minnesfel eller gränsöverskridanden som beror på programmets faktiska tillstånd. Där kompletterar dynamisk analys den statiska analysen genom att övervaka programmet medan det körs.

AI-genererad kod kan se strukturellt korrekt ut men ändå bete sig på oväntade sätt när den körs. Därför behövs den här typen av dynamisk analys.

ARGUMENT MOT AI? Nej, inget av detta är det. Produktivitetsvinsterna är verkliga. Inom inbyggd mjukvara, där kompetenta utvecklare är en bristvara och projektkomplexiteten växer, är varje verktyg som accelererar kodskrivandet värdefullt.

Men i ett AI-assisterat arbetsflöde förändras utvecklarens roll från kodförfattare till kvalitetsgranskare. Hantverket försvinner inte, men det ändrar fokus. I stället för att skriva varje funktion från grunden utvärderar utvecklaren AI-förslag mot domänkunskap, kör koden genom analysverktygen och gör de



Analysrapport som visar avvikelser mot MISRA C 2012 och CERT C i ett verkligt utvecklingsprojekt.

bedömningar som fortfarande kräver utvecklarens domänkunskap och ansvar: om logiken stämmer med avsikten, om testsviten täcker rätt scenarier och om arkitekturen håller ihop.

Det är det lagret av bedömning som förvandlar genererad kod till den granskade och godkända kod som reglerade branscher kräver.

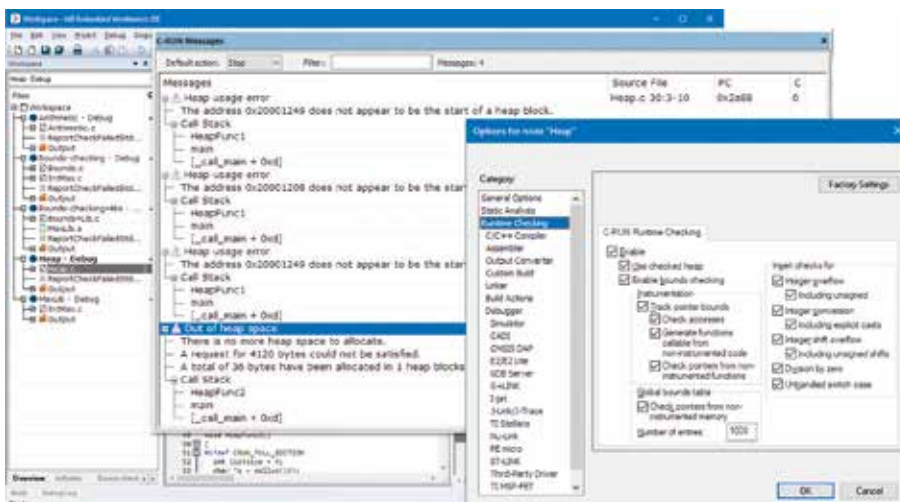
VERKTYG SOM KÖRS DIREKT vid arbetsstationen fångar upp många problem tidigt. Men i ett team räcker det inte att varje utvecklare kontrollerar sin egen kod för sig. Det behövs en gemensam kontrollpunkt: pipelinen, där varje commit automatiskt testas mot organisationens standarder oavsett vem som skrivit koden eller hur den togs fram.

En enda AI-assisterad kodsession kan i dag producera mer kod än vad som tidigare skapades under en hel arbetsvecka. Att då

förlita sig på manuella kvalitetskontroller fungerar helt enkelt inte längre. Kontrollerna måste köras automatiskt vid varje commit.

Samma kompilator, länkare och verktygskedja som används lokalt kan också paketeras för fristående körning i kontinuerlig integration (CI). Oavsett om pipelinen körs på Jenkins, GitHub Actions eller Azure DevOps blir bygget i CI identiskt med det som byggs på utvecklarens dator. För utveckling enligt ISO 26262 och IEC 62304 behöver verktyg och konfigurationer dokumenteras så att byggen kan reproduceras och granskas. Att köra samma verktygskedja i CI som lokalt gör reproducerbarheten påvisbar, samtidigt som CI-processen ger ett naturligt underlag för dokumentation.

Den statiska analysen kan köras fristående i CI och avvikelser rapporteras direkt som en del av byggresultatet. Över tid går det att följa hur testtäckningen utvecklas och om kodkvaliteten håller jämna steg. Bevisen för regelefterlevnad behöver alltså inte samlas ihop i sista stund inför en release utan byggs upp kontinuerligt genom hela projektet.



Dynamisk analys som upptäcker heap-fel och gränsöverskridanden under körning.

STATISK OCH DYNAMISK ANALYS räcker inte var för sig. De behöver köras i en och samma plattform, där byggen går att reproducera, verktygen är godkända och det går att följa hela vägen från kod till certifikat.

IAR har byggt sin utvecklingsplattform kring den här modellen. Byggverktygen är certifierade av TÜV SÜD för användning i utveckling enligt ISO 26262, IEC 61508 och IEC 62304. Analysverktygen C-STAT och C-RUN kan användas i samma verktygskedja lokalt och i CI, så att kontroller och dokumentation följer programvaran genom utvecklingsprocessen.