Whitepaper

Low-code software opent de weg naar Smart Maintenance

Jan Peter Meeuwse
Benno Beuting
Dick Lorier
Paul Cobben

EUROPESE UNIE
Europees Fonds voor Regionale Ontwikkeling.
Mede gefinancierd in het kader van de respons van de Unie op de COVID-19-pandemie.

Provincie Noord-Brabant

OPZuid

CORDIS SUITE

WORLD CLASS MAINTENANCE

Kennisproduct
Fieldlab VIA APPIA
Whitepaper
30-9-2024
Versie 1.0

Voorbeelden van wanneer je toe bent aan een nieuwe werkwijze in softwareontwikkeling:

- **Gebrekkige documentatie door een externe ontwikkelaar:** Een inhuurkracht heeft de software ontwikkeld, maar na zijn vertrek blijkt de documentatie minder volledig dan gedacht. Dit leidt ertoe dat problemen niet snel genoeg worden opgelost, omdat je niet precies weet hoe alles in elkaar zit.
- **Service engineers in een reactieve modus:** Je service engineers ontvangen veel 'calls' en moeten vaak naar de klant reizen om problemen op te lossen. Hierdoor werken ze constant reactief en blijven ze achter met andere werkzaamheden. Daarnaast zijn de kosten hoger dan je kunt doorbelasten aan de klant.
- **Ambitieus bedrijf met focus op service en onderhoud:** Je wilt een nieuw businessmodel opbouwen rond service en onderhoud voor je machines. Hoewel je een urenteller in de machine hebt ingebouwd, ontbreekt er data over slijtage of vroege signalen van onderhoudsbehoefte. Zonder toegang tot machinegegevens is het lastig om proactief te handelen.
- **Hololenzen benutten voor geavanceerde ondersteuning:** Je hebt hololenzen aangeschaft voor 'over-de-schouder' ondersteuning van service monteurs. Hoewel de hololens nuttig is als camera, kan je verder gaan door in de hololens live machinegegevens te tonen. Zo krijg je sneller inzicht in problemen, terwijl je twee handen vrij hebt voor het monteren of afstellen van onderdelen.
- **Code overnemen van een collega:** Een collega heeft de software voor een machine geschreven, maar hij is nu drie weken op vakantie in Bali. Hoe navigeer je door 200.000 regels code van een ander zonder heldere documentatie of begeleiding?
- **Omzetten van software voor een grote prospect:** Een grote klant heeft interesse in je machines, maar jij werkt uitsluitend met een specifiek merk besturingshardware. Het omzetten van je software naar dit merk is tijdrovend, riskant en betekent dat je uiteindelijk twee versies van de software moet onderhouden.

Standaardisatie met low-code modellen is het antwoord

Mechanische ontwerpers hebben CAD-tekeningen, elektrische ontwerpers werken met CAD-bedradingsschema's, en software engineers... die hebben vaak een beknopte schets van de software en beginnen te programmeren totdat de machine goed werkt. Maar hoe zou de wereld eruitzien als software engineers een soort CAD-tool hadden, waarbij de code automatisch en foutloos gegenereerd wordt?

Met behulp van **low-code** modelleer je grafisch het gewenste besturingsgedrag van een machine. Vanuit dit grafische model kan vervolgens de besturingssoftware voor de gekozen hardware automatisch gegenereerd worden. Dit grafische model speelt echter ook een centrale rol in de systeemdata, documentatie, gebruikersinterface, testen, en meer.

Traditioneel maken software engineers handgeschreven code die moeilijk toegankelijk is voor anderen. Dit betekent dat veel bedrijfskennis in die code ligt opgeslagen. Low-code modellen zijn te vergelijken met een mechanische of elektrische tekening van een machine: iedereen kan ze bekijken, begrijpen, specifieke punten aanwijzen en indien nodig aanpassingen doen. De code wordt automatisch gegenereerd en hoeft zelden handmatig bekeken te worden, omdat de vertaling van het low-code model foutloos is. Je bedrijfskennis wordt dus vastgelegd in een model, vergelijkbaar met een tekening, in plaats van in ondoorzichtige softwarecode.

Wanneer pas je low-code toe?

1. **Eenvoudig gedrag in minder dan 15 stappen**
   Als je het gedrag van je systeem in minder dan 15 stappen kunt beschrijven, is handmatige codering waarschijnlijk de betere optie. Je kunt dit wellicht afhandelen met minder dan 200 regels code.
2. **Software voor slechts één machine of veel variaties**
   Als je de software maar voor één systeem of machine maakt, is een traditionele ontwikkelmethodiek vaak voldoende. Neem bijvoorbeeld een stoplicht: dit is een relatief eenvoudig systeem dat prima te programmeren is met 200 regels code. **Maar** als jouw bedrijf zich richt op het ontwikkelen van **veel variaties** van stoplichten, wordt het al snel complex. In dat geval is een overstap naar low-code aan te raden, omdat het je de flexibiliteit en aanpasbaarheid biedt die je nodig hebt gedurende de levenscyclus van de software.
3. **Complexe softwareontwikkeling met meerdere ontwikkelaars**
   Als je bedrijf zich onderscheidt met software en je afdeling uit meer dan 2-3 ontwikkelaars bestaat die gezamenlijk aan projecten werken, is low-code een uitstekende oplossing. Bij traditionele softwareontwikkeling wordt kennis vastgelegd in moeilijk leesbare code, die vaak varieert per ontwikkelaar. Dit maakt het wisselen van taken lastig en kan leiden tot duizenden regels code die moeilijk te begrijpen en te onderhouden zijn. Low-code daarentegen maakt het softwareontwerp visueel, vergelijkbaar met elektrische of mechanische tekeningen. Dit maakt het eenvoudiger voor andere engineers om inzicht te krijgen in het ontwerp en sneller problemen op te lossen, zelfs tijdens jouw afwezigheid.
4. **Brownfield/Greenfield softwareprojecten**
   a. **Greenfield**: Bij een greenfield-project begin je met een blanco vel en ontwikkel je een volledig nieuw systeem. In dit geval is low-code de beste keuze om vanaf de basis een flexibel en schaalbaar systeem op te bouwen.
   b. **Brownfield**: Een brownfield-project houdt in dat je een bestaand systeem of machine van nieuwe software voorziet. De mechanische onderdelen blijven behouden, maar de automatisering wordt vernieuwd. Dit kan noodzakelijk zijn door verouderde hardware, ontbrekende documentatie, of componenten die niet langer leverbaar zijn. Low-code biedt hier een gecontroleerde manier om nieuwe, betrouwbare software te ontwikkelen, met de bijkomende voordelen van flexibiliteit en eenvoudig onderhoud in de toekomst.

Vrijwel elk bedrijf heeft reeds bestaande bibliotheken van software en wil deze blijven gebruiken. De low-code aanpak is zeer open en daardoor kunnen hierin reeds bestaande software componenten eenvoudig worden geintegreerd. Dit kunnen software componenten zijn die in-house gebouwd zijn of commercieel beschikbare componenten (bijv. vision, motion, algorithmes).

Low-code en onderhoud

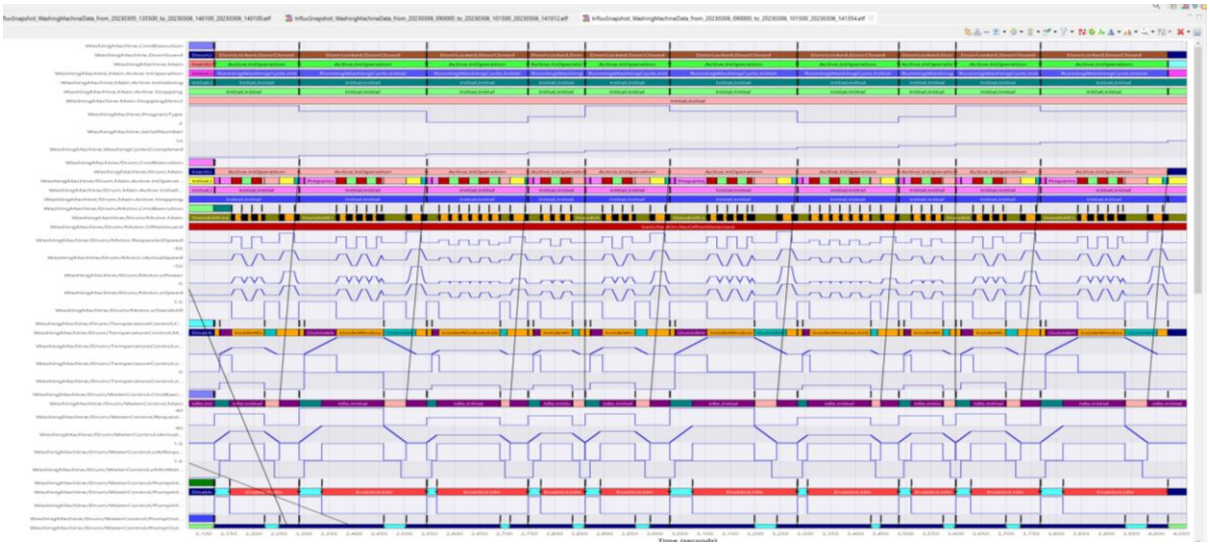
Vandaag de dag wordt machine-onderhoud voornamelijk gestuurd door het aantal bedrijfsuren en sensorwaarden die bepaalde drempels overschrijden. Met low-code kun je onderhoud veel inzichtelijker en voorspellender maken, doordat je toegang krijgt tot meer gedetailleerde en gestructureerde data uit een machine.

Machines voeren voortdurend repeterende taken uit, wat we ook wel **stateflow** noemen. Een voorbeeld van een stateflow in een wasmachine kan er als volgt uitzien:

1. Controleer of de deur gesloten is: 0,1 seconde
2. Vul waterreservoir: 180 seconden
3. Verwarm water: 600 seconden
4. Voeg zeeppoeder toe: 10 seconden
5. Centrifugeren: 120 seconden
6. ... etc.

Dit daadwerkelijk uitgevoerde gedrag kun je over een langere periode opslaan. Vervolgens kun je analyses uitvoeren, bijvoorbeeld om te vergelijken hoe lang de wasmachine nodig had om te centrifugeren toen deze nieuw was, en hoe lang dit duurt na vijf jaar gebruik. Mogelijk zie je verschillen die inzicht geven in het slijtagegedrag van de machine.

Met low-code wordt elk onderdeel van het machinegedrag, de instellingen en de sensor- en actuatorwaarden opgeslagen en eenvoudig toegankelijk gemaakt. Hierdoor krijg je inzicht in componenten, modules en het hele systeem. Je kunt met 'gezond boerenverstand' naar de data kijken, maar ook geavanceerde AI-algoritmes erop loslaten. Aan gestructureerde en gedetailleerde data zal het je in elk geval niet ontbreken.



Deze data is relatief eenvoudig op te roepen in een tablet, telefoon, hololens om op locatie direct inzicht te krijgen in het functioneren van het systeem.

Voorbeelden uit VIA APPIA

- **Robuustheid test van een component:** een machinebouwer heeft een kritische component in zijn machines, de met enige regelmaat faalt met kostbare stilstanden en materiaalverlies tot gevolg. Er is een testopstelling gebouwd om het faalgedrag van deze component te onderzoeken, op het moment van falen is werkelijk alle data met betrekking tot deze component beschikbaar en heeft men het faalmechanisme kunnen achterhalen.
- **HoloLens met augument reality:** een productiebedrijf heeft een lange productielijn (lengte ongeveer 120 meter). Een service engineer loopt diverse malen per dag zijn ronde voor inspectie van de lijn. Door gebruik van de HoloLens, ziet hij de meetwaardes direct

geprojecteerd op de machine. De inspectieronde kan hier mee sneller en diepgaander worden uitgevoerd.

- **Data om 'fingerspitzengefuhl' met de machine te verkrijgen:**

Hoe start ik met een low-code aanpak?

Een low-code aanpak vergt enige voorbereiding en leidt als snel (~1 maand) tot resultaten.

- **Training**: zoals elk vak is het van belang de basis goed te kennen en dit doe je middels een training in 'design skills' en het uitwerken & bediscussieren van cases met je medecursisten.
- **Project One**: elke bedrijf heeft zijn eigen specifieke werkwijze en wil deze grotendeels behouden. 'Project One' dient om de bestaande werkwijze samen te voegen met de low-code aanpak. De uitkomst is een stevige basis voor het projecten (en een blauwdruk voor alle projecten die volgen).
- **Coaching**: zoals elke engineer zijn ontwerp capaciteiten verbeterd over de jaren, is coaching een beproefde methode om feedback te krijgen en in korte cycli te verbeteren.