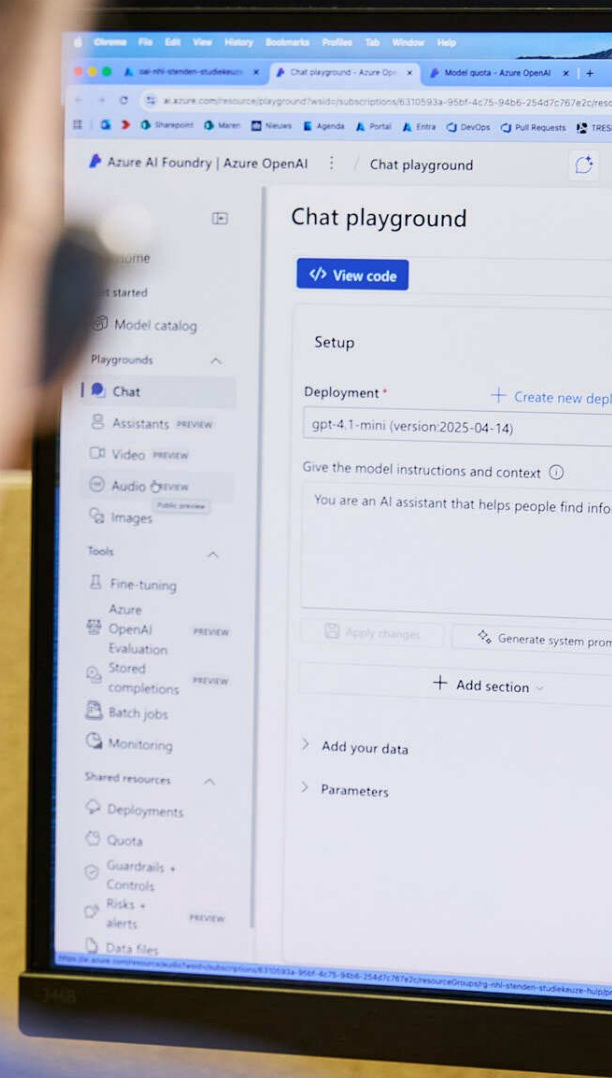




Snel waar het kan. Beheerst waar het moet.

Zo houd je grip als medewerkers zelf software
bouwen met AI



Software bouwen wordt toegankelijker. Met AI-codingtools kunnen medewerkers in natuurlijke taal uitleggen wat ze nodig hebben en in korte tijd een werkende toepassing maken.

Een planner bouwt een tool om productiecapaciteit inzichtelijk te maken. Een controller automatiseert een terugkerende berekening. Een servicemedewerker maakt een toepassing waarmee collega's sneller de juiste documentatie vinden.

Dat biedt kansen. Ideeën die eerder bleven liggen omdat er geen tijd, budget of ontwikkelcapaciteit voor was, kunnen ineens binnen een paar uur of dagen worden getest.

En dat gebeurt steeds vaker. In 2025 gebruikte 29,8% van de Nederlandse bedrijven met 10 tot 250 medewerkers minimaal één vorm van AI. Bij bedrijven met 50 tot 250 medewerkers was dat 45%.¹ AI maakt de stap van idee naar software kleiner. Daarmee ontstaat ook een nieuwe vraag:

Hoe zorg je dat een handige zelfgebouwde tool verantwoord meegroeit wanneer je organisatie er steeds afhankelijker van wordt?

Daarvoor ontwikkelde TRES het **Software Responsibility Model**. Het uitgangspunt is eenvoudig: de manier waarop je software ontwikkelt en beheert, moet passen bij de verantwoordelijkheid die je de software geeft.

Een tijdelijk experiment vraagt iets anders dan een applicatie die dagelijks orders verwerkt. En een interne rekentool vraagt iets anders dan software die persoonsgegevens verwerkt of rechtstreeks informatie naar je ERP schrijft.

Snel waar het kan. Beheerst waar het moet.

Vibe coding verlaagt de drempel om software te maken

Bij vibe coding beschrijf je wat je wilt maken en genereert AI een groot deel van de code. Je test het resultaat, geeft nieuwe instructies en past de toepassing stap voor stap aan.

Dat betekent niet dat programmeerkennis overbodig wordt. Onderzoek naar vibe coding laat juist zien dat kennis verschuift naar het beoordelen van resultaten, het bewaken van context, debugging en het herkennen van situaties waarin menselijke interventie nodig is.² Maar de drempel om iets werkends te maken wordt wel lager. Voor het MKB is dat interessant. Medewerkers kennen hun eigen werk vaak heel goed. Ze weten waar informatie steeds opnieuw moet worden ingevoerd, welke Excelbestanden iedere week gecombineerd moeten worden en welke terugkerende handelingen onnodig veel tijd kosten. Met AI kunnen zij daar steeds vaker zelf iets voor maken.

Dat kan bijvoorbeeld beginnen bij een planner in een productiebedrijf. Iedere week exporteert zij gegevens uit het ERP en combineert die handmatig met een Excelbestand waarin de beschikbare machinecapaciteit staat. Met een AI-codingtool bouwt ze een kleine toepassing die beide bestanden automatisch combineert.

Ze gebruikt de tool zelf. De officiële planning blijft leidend en als de toepassing een keer niet werkt, kan ze terugvallen op haar oude werkwijze. Op dat moment is het risico beperkt. Hier kan de snelheid van vibe coding dus erg waardevol zijn.

Een handige tool kan ongemerkt belangrijk worden

Een paar maanden later ziet de situatie er anders uit. Twee collega's gebruiken de tool inmiddels ook. Er zijn nieuwe functies toegevoegd en het Excelbestand met machinecapaciteit is vervangen door een rechtstreekse koppeling met het ERP. Nog een paar maanden later worden capaciteitsbeslissingen standaard op basis van de toepassing genomen.

De tool is succesvol en daardoor is er tegelijkertijd iets veranderd. De oorspronkelijke code kan grotendeels hetzelfde zijn. De verantwoordelijkheid die de organisatie aan de software geeft, is dat alleen niet.

Als de toepassing nu uitvalt of verkeerde informatie geeft, heeft dat gevolgen voor de dagelijkse planning. Misschien zelfs voor productie en levering aan klanten. Dat is het moment waarop vragen ontstaan die bij de eerste versie nauwelijks relevant waren.

Wie heeft toegang? Wie beheert de toepassing? Waar staan wachtwoorden en API-keys? Kunnen we een vorige versie herstellen? Wie merkt dat een koppeling niet goed werkt? Wat gebeurt er wanneer de medewerker die de tool heeft gemaakt vertrekt? Een werkende applicatie geeft daar geen antwoord op.

Onderzoek naar AI-generated code laat zien dat functioneel werkende code niet automatisch betekent dat de kwaliteit en security op orde zijn.³ Dat geldt overigens ook voor software die volledig door mensen wordt geschreven. Het verschil zit vooral in de snelheid waarmee tegenwoordig een werkende eerste versie kan ontstaan.

“Hoe sneller software ontstaat, hoe belangrijker het wordt om te herkennen wanneer een experiment verandert in iets waar je organisatie op vertrouwt.”

Hoeveel verantwoordelijkheid krijgt de software?

Het TRES Software Responsibility Model helpt om dat gesprek te voeren. Daarvoor kijken we naar twee vragen.

1. Wat zijn de gevolgen als het misgaat?

Denk aan de gevolgen voor klanten, medewerkers, financiën, privacy, security, continuïteit en belangrijke bedrijfsdata. Een interne tool waarmee één medewerker openbare gegevens ordent heeft een ander profiel dan een toepassing die automatisch verkoopprijzen naar klanten stuurt.

2. Hoe structureel wordt de software gebruikt?

Denk aan de gevolgen voor klanten, medewerkers, financiën, privacy, security, continuïteit en belangrijke bedrijfsdata. Een interne tool waarmee één medewerker openbare gegevens ordent heeft een ander profiel dan een toepassing die automatisch verkoopprijzen naar klanten stuurt.

Vier situaties, vier manieren van werken

1. Vibe coding sweet spot

Tijdelijk gebruik + beperkte gevolgen

Hier kun je de snelheid van vibe coding volop benutten. Denk aan prototypes, proof-of-concepts, tijdelijke rekentools of een kleine toepassing waarmee een team onderzoekt of een idee werkt.

De planner uit ons voorbeeld begon hier. Ze gebruikte haar tool zelf, kon altijd terugvallen op de bestaande planning en er waren geen kritieke processen afhankelijk van.

De basis moet natuurlijk op orde zijn. Bedrijfswachtwoorden horen bijvoorbeeld niet in openbare code en gevoelige klantinformatie hoort niet zomaar in een willekeurige AI-dienst. Maar uitgebreide softwareprocessen zouden hier vooral snelheid wegnemen.

Ons advies: experimenteer, test en leer.

2. Controlled vibe coding

Tijdelijk gebruik + grote gevolgen

Sommige software wordt maar kort gebruikt, terwijl een fout wel direct grote gevolgen kan hebben. Denk aan een tijdelijke migratietool die duizenden gegevens naar het ERP schrijft, een script dat verkoopprijzen aanpast of een toepassing die tijdelijk persoonsgegevens verwerkt.

De levensduur is kort. De impact kan groot zijn. Dan zijn bijvoorbeeld beheerde accounts, versiebeheer, toegangsrechten, veilige opslag van wachtwoorden en API-keys, logging, rollback en een technische review belangrijk. Welke maatregelen nodig zijn, hangt af van wat de toepassing doet.

Ons advies: benut de snelheid, met passende technische vangrails.

3. Managed development

Duurzaam gebruik + beperkte gevolgen

Een andere toepassing kan relatief weinig schade veroorzaken als hij een dag uitvalt, maar wel jarenlang onderdeel blijven van het dagelijkse werk. Denk aan een intern kwaliteitsdashboard, een kennisapplicatie of een servicetool.

Dan verschuift de vraag. Wie onderhoudt de software over twee jaar? Wie houdt gebruikte componenten actueel? Kan iemand anders het beheer overnemen? Kunnen nieuwe versies betrouwbaar worden uitgerold?

Hier worden versiebeheer, documentatie, dependency management, reproduceerbare deployments, eigenaarschap en passende monitoring belangrijk. Ons advies: gebruik AI volop, maar behandel het resultaat als een softwareproduct.

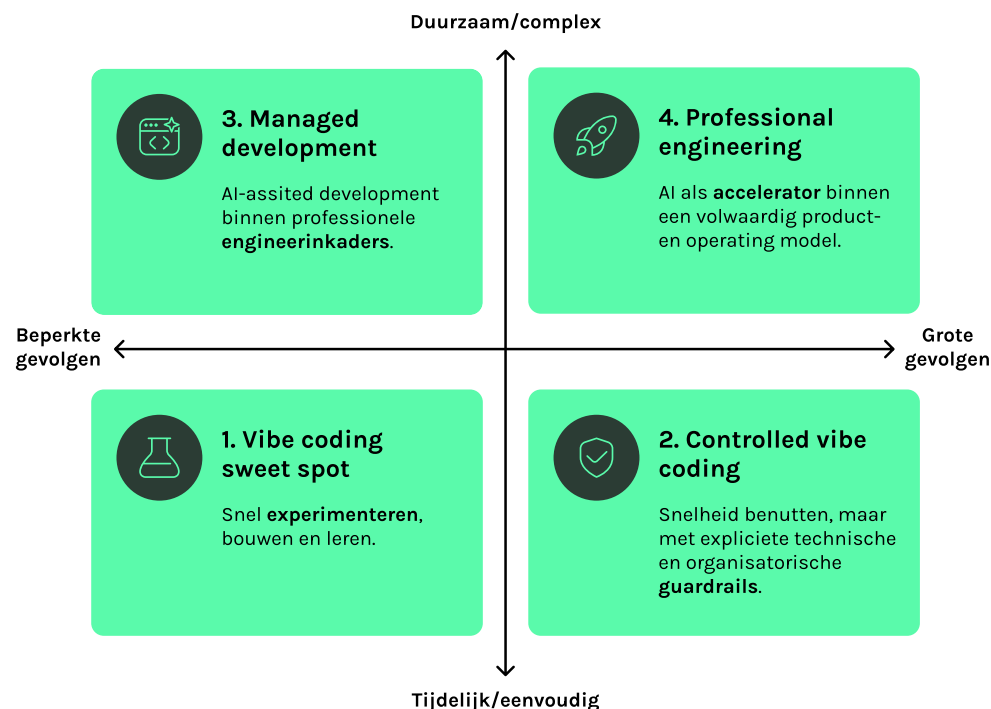
4. Professional engineering

Duurzaam gebruik + grote gevolgen

Hier wordt de organisatie structureel afhankelijk van de software en kunnen fouten grote gevolgen hebben. Bijvoorbeeld bij een klantportaal dat rechtstreeks gekoppeld is aan ERP en orderdata, software die productieplanning aanstuurt of een toepassing waarin belangrijke financiële of contractuele logica zit. Architectuur, security, testen, toegangsbeheer, monitoring, herstelbaarheid, documentatie en lifecyclemanagement horen dan bij het product.

AI kan ook hier een grote rol spelen. Professionele developers gebruiken AI bijvoorbeeld voor code, tests, analyse, documentatie en debugging. Het verschil zit dus niet in het wel of niet gebruiken van AI. De verantwoordelijkheid van de software bepaalt hoeveel engineering en beheer nodig zijn.

Ons advies: zet AI in als versneller, binnen professionele software-engineering.



Wanneer moet je opnieuw kijken?

Een tool hoeft niet voor altijd in hetzelfde vak van het model te blijven. Sterker nog: succesvolle software beweegt vaak. De planner uit ons voorbeeld begon met een tijdelijke persoonlijke tool. Toen collega's hem gingen gebruiken, werd continuïteit belangrijker. Met de ERP-koppeling nam de impact verder toe. En zodra de productieplanning structureel afhankelijk wordt van de toepassing, hoort daar een andere manier van ontwikkelen en beheren bij.

Daarom is het belangrijk om niet alleen bij de start naar software te kijken. Een herbeoordeling is bijvoorbeeld verstandig zodra:

- » meer mensen de toepassing gaan gebruiken;
- » de toepassing langer blijft bestaan dan oorspronkelijk verwacht;
- » persoonsgegevens worden verwerkt;
- » er een koppeling met een kernsysteem bijkomt;
- » de toepassing gegevens mag wijzigen of schrijven;
- » klanten de toepassing rechtstreeks gaan gebruiken;
- » financiële of andere belangrijke bedrijfslogica wordt toegevoegd;
- » een bestaande handmatige terugvalmogelijkheid verdwijnt.

De software groeit. En de manier waarop je haar beheert groeit op deze manier mee.

Grip hoeft innovatie niet af te remmen

De reflex kan zijn om alle zelfgebouwde software centraal te laten goedkeuren. Voor een MKB-organisatie levert dat al snel een zwaar proces op. En juist de snelheid waarmee medewerkers een idee kunnen testen is een van de voordelen van AI-assisted development.

Een werkbaarder uitgangspunt is daarom: maak duidelijk wat medewerkers zelf kunnen doen en wanneer extra expertise nodig is.

Een eenvoudig register van zelfgebouwde toepassingen kan daarbij al helpen. Leg bijvoorbeeld vast wat de toepassing doet, wie eigenaar is, wie haar gebruikt, welke gegevens worden verwerkt en met welke systemen zij verbonden is. Daarna draait het vooral om duidelijke momenten waarop opnieuw wordt gekeken.

Dat past ook bij bestaande richtlijnen voor veilige softwareontwikkeling. NIST adviseert organisaties maatregelen af te stemmen op hun businessbehoefte, risico's en beschikbare middelen.⁴ OWASP benadrukt daarnaast dat AI-generated code een menselijke eigenaar nodig heeft en dat AI-assisted wijzigingen moeten worden beoordeeld.⁵

Voor een organisatie betekent dat bijvoorbeeld: een prototype hoeft geen uitgebreid securitytraject te doorlopen. Een toepassing die persoonsgegevens gaat verwerken verdient wel extra aandacht. Een koppeling die gegevens naar het ERP schrijft vraagt om technische controle. En software waarvan de dagelijkse operatie afhankelijk wordt, vraagt om professioneel beheer.

Zo ontstaat ruimte om te experimenteren en duidelijkheid over het moment waarop de verantwoordelijkheid verandert.

Wie houdt overzicht als steeds meer medewerkers zelf bouwen?

Naarmate medewerkers meer zelf kunnen ontwikkelen, ontstaat een nieuwe beheeropgave.

Wie weet welke toepassingen er inmiddels bestaan? Wie ziet wanneer een experiment steeds belangrijker wordt? Wie beoordeelt of toegangsbeheer en security nog passen? En wie zorgt ervoor dat software overdraagbaar blijft wanneer de oorspronkelijke maker iets anders gaat doen? Dat vraagt om overzicht en eigenaarschap.

TRES kan organisaties helpen om zelfgebouwde software te inventariseren en te beoordelen met het Software Responsibility Model. Vervolgens bepalen we welke technische en organisatorische maatregelen passen bij de verantwoordelijkheid van iedere toepassing.

Soms kan een team gewoon verder experimenteren. Soms zijn een paar extra vangrails voldoende. En soms is software inmiddels zo belangrijk geworden dat professionele engineering nodig is. Zo blijft de snelheid van AI behouden, terwijl de organisatie grip houdt op de software waar zij steeds meer op vertrouwt.

En hoe zit het met wetgeving?

De ontwikkelmethode bepaalt niet automatisch welke wetgeving van toepassing is. Software die met behulp van AI is ontwikkeld, is bijvoorbeeld niet uitsluitend daarom zelf een AI-systeem onder de Europese AI Act. Daarvoor is relevant wat het uiteindelijke systeem doet.⁶

Verwerkt een toepassing persoonsgegevens, dan blijven de verplichtingen uit de AVG gelden. Organisaties zijn zelf verantwoordelijk voor de verwerking van persoonsgegevens en moeten hun systemen en processen daarop inrichten.⁷

Voor bedrijven die digitale producten of softwarecomponenten op de Europese markt brengen, kan daarnaast de Cyber Resilience Act relevant zijn. De rapportageverplichtingen voor fabrikanten gelden sinds 11 september 2026; de verordening wordt vanaf 11 december 2027 volledig van toepassing.⁸

Welke regels precies gelden, hangt af van de toepassing, de gegevens die worden verwerkt en de rol van de organisatie. Daarom is wetgeving één van de onderdelen die je meeweegt zodra software meer verantwoordelijkheid krijgt.

Deze whitepaper biedt een bestuurlijk en technisch denkkader en is geen juridisch advies. Voor concrete toepassingen waarbij persoonsgegevens, gereguleerde besluitvorming, veiligheidsfuncties of softwareproducten voor de markt betrokken zijn, moet de toepasselijke wet- en regelgeving afzonderlijk worden beoordeeld.

Twee vragen voor je volgende MT-overleg

Software bouwen met AI wordt steeds toegankelijker. Dat biedt organisaties de kans om sneller te experimenteren en verbeteringen uit te proberen. De uitdaging zit vooral in wat er daarna gebeurt.

Maak daarom eens een overzicht van de tools, scripts, spreadsheets en applicaties die medewerkers de afgelopen twaalf maanden zelf hebben gemaakt. Kies er drie uit en stel per toepassing twee vragen:

- » Wat gebeurt er als dit morgen verkeerd gaat?
- » Wat gebeurt er als we dit over twee jaar nog steeds gebruiken?

Je hoeft daar niet direct een uitgebreid softwarebeleid voor te schrijven. De antwoorden laten meestal al zien waar extra aandacht nodig is.

Snel waar het kan. Beheerst waar het moet.

Meer weten?

Neem contact met ons op.

Harmen Visser

☎ 06 54 24 19 90

✉ harmenvisser@tres.nl

🌐 www.tres.nl



Bronnen

1. CBS

[Gebruik van AI-technologie door Nederlandse microbedrijven en Bedrijven gebruiken AI vaakst voor marketing of verkoop](#), 2025/2026. CBS rapporteert voor 2025 AI-gebruik bij 29,8% van het Nederlandse MKB met 10-249 werkzame personen en 45% bij bedrijven met 50-249 werkzame personen.

2. Microsoft Research

Sarkar, A. & Drosos, I. [Vibe coding: programming through conversation with artificial intelligence](#), 2025. Onderzoek naar de manier waarop developers met generatieve AI programmeren en de verschuiving van programmeerwerk naar onder andere contextmanagement, evaluatie en debugging.

3. Sabra, A., Schmitt, O. & Tyler, J. (2025)

[Assessing the Quality and Security of AI-Generated Code: A Quantitative Analysis](#). Onderzoek naar 4.442 Java-opgaven gegenereerd door vijf verschillende AI-modellen. De resultaten laten zien dat functioneel werkende AI-code nog bugs, securitykwetsbaarheden en code smells kan bevatten en dat functionele prestaties geen betrouwbare indicator zijn voor algemene codekwaliteit en security.

4. NIST

[Secure Software Development Framework \(SSDF\)](#). Het framework beschrijft secure software development als een lifecyclevraagstuk en adviseert organisaties maatregelen af te stemmen op businessbehoeften, risicotolerantie en beschikbare middelen.

5. OWASP

[Secure Coding with AI Cheat Sheet](#). OWASP adviseert onder andere menselijk eigenaarschap van AI-generated code, expliciete review en herleidbaarheid van AI-assisted wijzigingen.

6. Europese Commissie

AI Act / Regulatory framework for AI. De AI Act is sinds 2 augustus 2026 grotendeels van toepassing. Voor bepaalde high-risk toepassingen gelden latere termijnen.

7. Autoriteit Persoonsgegevens

AVG en verantwoordingsplicht. Organisaties blijven verantwoordelijk voor de verwerking en bescherming van persoonsgegevens, onafhankelijk van de manier waarop software is ontwikkeld.

8. Europese Commissie

[Cyber Resilience Act](#). Rapportageverplichtingen voor fabrikanten rond actief uitgebuite kwetsbaarheden en ernstige security-incidenten gelden sinds augustus 2026. De CRA wordt vanaf 11 december 2027 volledig van toepassing.

Aanvullende onderzoeksbasis

Voor de bredere onderbouwing van AI-assisted development kan ook het onderzoek van Cui, Demirer, Jaffe, Musolff, Peng en Salz worden opgenomen. In drie veldexperimenten met in totaal 4.867 developers werden gemiddeld 26,08% meer taken afgerond met een AI-codingassistant. Een METR-experiment uit begin 2025 onder zestien zeer ervaren open-sourceontwikkelaars vond in die specifieke onderzoekssetting juist een vertraging van 19%. METR meldde in 2026 aanwijzingen dat nieuwere tools inmiddels meer versnelling opleveren, maar noemt de nieuwe data zelf onvoldoende betrouwbaar om daar een algemeen percentage aan te verbinden.

Juist die uiteenlopende resultaten ondersteunen een belangrijk uitgangspunt van deze whitepaper: de waarde van AI-assisted development hangt af van de taak, context, gebruiker en manier waarop de technologie wordt ingezet.