

Onderhoudbare detectie- en vergelijkingsketen voor in-house software in EPC en WSIT/CMDB

Reflectie

Quinten De Meyer
Student Bachelor in de Elektronica-ICT — Cloud & Cyber Security

Stagebedrijf: Pfizer Puurs — ICS Infrastructure

Stagecoach Thomas More: Cas Magnus
Stagementor Pfizer: Nikki Lissens

Inhoudsopgave

1.	Inleiding	3
2.	Inhoudelijke reflectie	4
3.	Persoonlijke reflectie	8

1. Inleiding

Dit reflectiedocument blikt terug op het uitgevoerde stageproject bij Pfizer in Puurs-Sint-Amands. Tijdens deze stage werd gewerkt aan de ontwikkeling van een onderhoudbare en overdraagbare workflow voor het detecteren van in-house software op endpoints en het vergelijken van die software met de administratieve registraties in de CMDB binnen WSIT. De aanleiding voor het project (zoals beschreven in het projectplan; De Meyer, 2026a) was het gebrek aan betrouwbare zichtbaarheid op intern ontwikkelde of niet-standaardsoftware via de standaardinventarisatie van Endpoint Central.

Het stageproject resulteerde (zie realisatiedocument; De Meyer, 2026b) in een volledige toolketen die softwaredetectie, dataverzameling, vergelijking en semiautomatische CMDB-update combineert. Naast de technische realisatie werd ook aandacht besteed aan onderhoudbaarheid, overdraagbaarheid en praktische bruikbaarheid binnen de werkomgeving van Pfizer.

In hoofdstuk 2 wordt inhoudelijk gereflecteerd op het gerealiseerde project, de behaalde resultaten en de meerwaarde voor het bedrijf. Hoofdstuk 3 bevat een persoonlijke reflectie over de opgedane ervaringen, de competenties die tijdens de stage aan bod kwamen en de manier waarop met technische en organisatorische uitdagingen werd omgegaan.

2. Inhoudelijke reflectie

Tijdens deze stage werd een volledige EPC-CMDB-toolketen gerealiseerd waarmee in-house software op endpoints kan worden gedetecteerd, vergeleken met de CMDB en semiautomatisch kan worden verwerkt in Jira Assets binnen WSIT. De oplossing bestaat uit meerdere samenwerkende componenten, waaronder een centraal YAML-bronbestand, een profielgenerator, een Python-gebaseerde detection engine, scripts voor dataverzameling en normalisatie, een comparison engine, een launcher-tool en ondersteunende beheer- en buildtools.

De gerealiseerde oplossing levert concrete meerwaarde op voor de organisatie, aangezien een correcte en actuele software-inventaris cruciaal is voor security-audits, betrouwbare infrastructuur-upgrades en licentiebeheer. Voorheen was er geen reproduceerbare methode om de feitelijke toestand van in-house software op endpoints te vergelijken met de administratieve registratie in de CMDB. De standaardinventarisatie van Endpoint Central detecteerde een deel van deze software niet, omdat de installaties geen gebruikmaken van klassieke Windows Installer-mechanismen. Door de nieuwe detectieworkflow ontstaat nu een beter zicht op actieve in-house software binnen de afgesproken scope. Hierdoor kunnen discrepanties sneller worden opgespoord en kan de kwaliteit van de CMDB verbeterd worden.

Daarnaast levert de oplossing ook operationele voordelen op voor toekomstige beheerders. De workflow is opgebouwd rond onderhoudbaarheid en overdraagbaarheid. Nieuwe software kan worden toegevoegd via het YAML-bronbestand zonder wijzigingen aan de detectie-engine zelf. Dankzij de Software Source Maintainer kunnen collega's het bronbestand onderhouden zonder kennis van YAML-syntax. Ook de vaste mappenstructuur, automatische input- en outputrouting, centrale launcher en uitgebreide Confluence-documentatie dragen bij aan een lagere beheerdrempel.

Het project is functioneel afgerond en de volledige workflow werkt in productie voor duizenden Windows systemen. De grote productiescan werd succesvol uitgevoerd en leverde

bruikbare detectieresultaten op. De comparison engine genereert correct discrepantierapporten en importeerbare CSV-bestanden voor WSIT. Ook het importproces in Jira Assets werd gevalideerd in de testomgeving.

Toch zijn er nog enkele beperkingen en mogelijke uitbreidingen. Oudere systemen, zoals bepaalde Windows 7 Enterprise/Ultimate-endpoints, blijven problematisch door compatibiliteitsproblemen in de execution context of patch-baseline. Daarnaast vallen 32-bit systemen buiten de actuele uitvoerbare scope door beperkingen in Endpoint Central zelf. Hoewel deze systemen momenteel weinig impact hebben op de globale bruikbaarheid van de tool, kan het in de toekomst nuttig zijn om alternatieve uitvoeringsmethodes te onderzoeken voor deze endpoints.

Een tweede mogelijke uitbreiding betreft gedecommissioneerde software. Momenteel detecteert de tool enkel actieve, gemodelleerde software binnen scope. Technisch zou het mogelijk zijn om ook gedecommissioneerde software op te nemen, maar dat zou de onderhoudslast van het bronbestand aanzienlijk verhogen. Indien de organisatie in de toekomst meer nadruk wil leggen op historische softwaredetectie of cleanup-controles, kan deze keuze herbekeken worden.

De oplossing werd tijdens de stage door mij in gebruik genomen als eerste volledige uitvoering van de workflow. De detectie-engine werd succesvol uitgevoerd op productie-endpoints en de volledige workflow werd meerdere keren end-to-end getest. Bovendien werden ook infrastructuurmaatregelen genomen om het project duurzaam beschikbaar te houden, zoals het aanmaken van een configuratiebestand, aangezien de netwerkshare kan veranderen. Er werd ook overdrachtsdocumentatie in Confluence opgesteld.

Voor de toekomst lijkt het belangrijk dat de organisatie blijft investeren in het onderhoud van het YAML-bronbestand. De kwaliteit van de detectie blijft immers afhankelijk van correcte en actuele softwaredefinities. Daarnaast is het aanbevolen om wijzigingen in installatieprocedures of nieuwe softwareversies systematisch te verwerken in het bronbestand, zodat de detectie betrouwbaar blijft naarmate nieuwe software uitgerold wordt op endpoints.

Een weloverwogen scopekeuze betrof de omgang met gedecommissioneerde in-house software. De tool detecteert geldige, actieve in-house software, maar niet software die gedecommissioneerd is en mogelijk nog op endpoints staat. Geldige software detecteren is daarmee niet hetzelfde als alle aanwezige in-house software detecteren. Die afbakening verdient een uitgebreide toelichting, omdat ze raakt aan de volledigheid van de oplossing.

Technisch gezien is het toevoegen van gedecommissioneerde software betrekkelijk eenvoudig. Het YAML-bronbestand [CM8] (het centrale bestand waarin softwarekennis gestructureerd wordt beschreven) zou kunnen worden uitgebreid met een statusveld per product, bijvoorbeeld status: valid of status: decommissioned, en de detectie-engine zou die status kunnen meegeven in de output. De moeilijkheid zit niet in het datamodel of in de code, maar in het onderhoud. Gedecommissioneerde software die men wil blijven detecteren, mag niet uit het bronbestand worden verwijderd maar moet juist worden bewaard. Dat maakt het bronbestand aanzienlijk langer en lastiger te onderhouden, zowel voor de huidige ontwikkelaar als voor toekomstige beheerders. Het handmatig toevoegen van alle historische gedecommissioneerde software zou bovendien een grote tijdsinvestering vergen die niet in verhouding staat tot de meerwaarde.

De strikte aanpak is op zichzelf een bewuste keuze die voortkomt uit de werkomgeving. Binnen Pfizer wordt ervan uitgegaan dat software correct volgens IID of formeel installatieplan wordt geïnstalleerd. Software-installaties verlopen via een ticketproces waarbij meerdere personen betrokken zijn: een installatie die niet volgens procedure verloopt, wordt gerapporteerd aan de verantwoordelijke, en een actie zonder formeel proces wordt als zware fout beschouwd. In elke IID staat ook een deïnstallatieprocedure, waardoor uitfasering in dezelfde procedurele context valt als installatie. De primaire controle op correcte uitfasering is dus het ticket- en procedureproces zelf, niet een technisch detectiemiddel. De stagementor bevestigde dit uitgangspunt: gedecommissioneerde

endpoints mogen genegeerd worden, en software wordt correct geïnstalleerd via procedure en ticketing.

Toch is het eerlijk om te erkennen dat dit een blinde vlek oplevert als de opdracht breed wordt gelezen als het detecteren van alle in-house software die EPC niet ziet. Gedecommissioneerde in-house software die nog op een endpoint aanwezig is, is nog altijd in-house software. De tool is in dat opzicht geen volledige in-house software detector, maar een gerichte detector voor actieve, gemodelleerde software. Argumenten als "die software zal wel juist gedeïnstalleerd zijn" zijn niet overtuigend, juist omdat het doel van een detectietool is om zichtbaar te maken wat anders onzichtbaar blijft. Het sterkste en eerlijkste argument voor de gemaakte afbakening is tweeledig: enerzijds de disproportionele onderhoudskost ten opzichte van de meerwaarde, en anderzijds het feit dat de primaire controle voor uitfasering in deze omgeving de procedureflow is en niet de detectietool.

3. Persoonlijke reflectie

Deze stage was voor mij een zeer leerrijke en intensieve ervaring waarin ik niet alleen technisch ben gegroeid, maar ook sterker ben geworden in zelfstandig werken, probleemoplossend denken en communiceren binnen een professionele IT-omgeving. Het project gaf mij de kans om een complexe en realistische endpoint- en serverbeheeromgeving van dichtbij mee te maken en om een oplossing uit te werken die effectief gebruikt kan worden binnen een grote organisatie.

Op technisch vlak heb ik bijzonder veel bijgeleerd. Tijdens de stage werkte ik met verschillende technologieën en platformen, waaronder Python, PowerShell 7, C#/WPF, Jira Assets, REST API's, YAML, JSON en Endpoint Central. Vooral het programmatisch integreren van de Jira REST API en het opzetten van dynamische Custom Groups in Endpoint Central waren volledig nieuw voor mij. Hierbij leerde ik concreet hoe bedrijfsapplicaties veilig via API's met elkaar communiceren en hoe grootschalige scriptdistributie werkt. Ik heb geleerd hoe belangrijk onderhoudbaarheid en overdraagbaarheid zijn binnen enterprise-omgevingen. In schoolprojecten ligt de focus vaak vooral op functionaliteit, terwijl tijdens deze stage duidelijk werd dat gebruiksvriendelijkheid, documentatie, foutbestendigheid en beheerbaarheid minstens even belangrijk zijn.

Ik ben ook sterk gegroeid in analytisch en probleemoplossend denken. Veel problemen tijdens de stage hadden geen directe of eenvoudige oplossing. Een voorbeeld hiervan was het importproces in WSIT, waarbij objecttypes, AQL-filters en object keys onverwachte problemen veroorzaakten. De oplossing kwam niet uit één grote wijziging, maar uit het stap voor stap analyseren van gedrag, testen van hypothesen en aanpassen van de configuratie tot het volledige proces correct werkte. Ook performanceproblemen in de detection engine vereisten een analytische aanpak. Door grondige analyse ontdekte ik dat één ontbrekende detectie-indicator leidde tot zware bestandsvergelijkingen, waarna de runtime drastisch kon worden teruggebracht van minuten naar seconden.

Daarnaast heb ik geleerd om pragmatische keuzes te maken. Niet elk probleem moet volledig technisch opgelost worden als de organisatorische context daar geen meerwaarde voor biedt. De keuze om gedecommissioneerde software buiten scope te houden is daar een voorbeeld van. Technisch was dit mogelijk, maar binnen de context van Pfizer woog de onderhoudslast niet op tegen de praktische meerwaarde. Ik heb geleerd om technische beslissingen niet alleen vanuit technologie, maar ook vanuit beheerbaarheid, tijdsinvestering en organisatorische realiteit te evalueren.

Een ander voorbeeld hiervan is het publiceren van de Python-tools als uitvoerbare bestanden (.exe). Technisch gezien was dit niet noodzakelijk; de launcher kon de scripts ook rechtstreeks uitvoeren. Ik heb deze keuze voornamelijk gemaakt om een duidelijk visueel en operationeel onderscheid te creëren tussen ontwikkelartefacten (_dev) en releaseartefacten (release). Dit voorkomt dat toekomstige beheerders per ongeluk broncode wijzigen in de operationele omgeving. Deze beslissing leerde mij dat onderhoudbaarheid, overdraagbaarheid en duidelijkheid soms minstens even belangrijk zijn als pure technische optimalisatie.

Daarnaast is een zeer relevante skill een oplossing niet alleen technisch laten werken, maar vooral het maken met in gedachte dat de oplossing voor een lange tijd op regelmatige basis moet worden gebruikt. Ik hoorde van het team dat de tools van vorige stagiaires technisch wel werken, maar dat ze praktisch nooit gebruikt worden. Dit kwam als een oogopener voor mij om mijn prioriteiten in het gebruiksgemak en onderhoudbaarheid te verhogen.

Ook communicatief ben ik gegroeid. Tijdens de stage moest ik regelmatig overleggen met mijn stagementor en collega's over technische keuzes, beperkingen en risico's. Ik leerde technische problemen duidelijk uitleggen aan mensen met verschillende perspectieven en rekening houden met hun feedback en verwachtingen. Daarnaast heb ik ervaren hoe belangrijk documentatie is in professionele omgevingen. Het schrijven van Confluence-handleidingen en het uitwerken van overdraagbare workflows maakte duidelijk dat een oplossing pas echt bruikbaar is wanneer anderen ermee verder kunnen werken.

Niet alle uitdagingen waren puur technisch. Een belangrijke moeilijkheid was omgaan met onzekerheid en veranderende vereisten. Tijdens de stage evolueerde het project voortdurend. Verschillende tools, zoals de Software Source Maintainer, de EXE Reviewer, de launcher-tool en de Result Merger / Normalizer, waren oorspronkelijk niet voorzien maar bleken noodzakelijk om de workflow praktisch bruikbaar te maken. Dat zorgde ervoor dat ik mijn planning regelmatig moest aanpassen en prioriteiten moest herbekijken. Ik heb geleerd om flexibel om te gaan met zulke veranderingen zonder het overzicht over het geheel te verliezen.

Een andere uitdaging was het werken binnen beperkingen van bestaande infrastructuur en systemen waar ik zelf geen controle over had. Problemen met bepaalde Endpoint Central-servers, rechtenstructuren en compatibiliteit van oudere besturingssystemen konden niet altijd volledig opgelost worden. Ik heb geleerd dat professioneel IT-werk vaak betekent dat je oplossingen moet bouwen binnen bestaande beperkingen en afhankelijkheden.

Ik kijk zeer positief terug op deze stage. Het project gaf mij de kans om een complexe technische oplossing uit te werken die niet enkel theoretisch werkt, maar ook praktisch inzetbaar is binnen een grote organisatie. Ik heb niet alleen mijn technische kennis uitgebreid, maar ook geleerd hoe belangrijk analyse, documentatie, onderhoudbaarheid en samenwerking zijn binnen professionele IT-projecten. Deze ervaring heeft mij meer vertrouwen gegeven in mijn eigen capaciteiten en vormt een sterke voorbereiding op een toekomstige professionele carrière binnen IT en softwarebeheer.