BCE

De meesters in digitale transformatie

HEFBOMEN OM HET
succes van low/no-code te vergroten

Klaas-Jan Molendijk
Partner & adviseur van BCE

Met dank aan BCE-collega's Joost van Bilsen, Maxine Hilhorst, Inge de Laat, Lisa Rijksen en Eric van der Wolk.

Oktober 2019

Benodigde voorkennis:

- Je weet wat low/no-code software-ontwikkeling is
- Je weet wat heavy-code software-ontwikkeling is
- Je hebt basiskennis van Scrum
- Je hebt basiskennis van Kanban
- Je hebt bij voorkeur meerdere softwareontwikkel-projecten van dichtbij meegemaakt

Inhoudsopgave

INLEIDING

INZICHT 1	Low/no-code kan een katalysator van cultuurverandering zijn
INZICHT 2	Veel aandacht voor het besturingsmodel creëert mogelijk een illusie van succes & verandering
INZICHT 3	Een high-performing low/no-code-team is altijd de nummer één van alle hefbomen
INZICHT 4	Gezond boerenverstand is belangrijker dan Scrum
INZICHT 5	Met Kanban-principes kun je de overhead binnen het project reduceren
INZICHT 6	Het loont om lak te hebben aan rollen
INZICHT 7	In low/no-code-projecten vraagt de backlog om veel meer tijd en energie
INZICHT 8	Een ijzersterk project begint met een glashelder architectuur-ontwerp
INZICHT 9	Veel klant-/gebruikersorganisaties hebben aanvankelijk veel positieve 'weerstand' nodig
INZICHT 10	Met de low/no-code-store kun je ontwikkeling versnellen
INZICHT 11	Kanariereleases zorgen voor rust bij softwareopleveringen met veel impact
INZICHT 12	Testautomatisering is toegankelijker te maken met Robotic Process Automation (RPA)
INZICHT 13	Met Robotic Process Automation (RPA) zijn diverse low/no-code-applicaties nog krachtiger te maken
INZICHT 14	Een low/no-code-platform is geen oplossing voor alles

INLEIDING

Hefbomen om het succes van low/no-code te vergroten

Low/no-code-gereedschap biedt geen garantie voor *building applications many times faster/better*, zoals de belofte luidt van diverse platformleveranciers. Aan de techniek ligt het wat ons betreft niet meer, maar de vraag is wel: hoe maak je van jouw low/no-code-initiatieven het beloofde enorme succes? Dat lijkt misschien een technologisch vraagstuk, maar is vooral een organisatorisch, menselijk vraagstuk.

Vind je dit een interessante hypothese? Lees dan onze 14 inzichten. Inzichten die je als hefboom kunt zien om het maximale uit low/no-code te halen. Geschreven voor organisaties die nu met low/no-code gaan beginnen. En natuurlijk ook voor organisaties die onderweg zijn en die de inzichten willen gebruiken als checklist om vast te stellen waar kansen liggen om low/no-code-succes te vergroten.

Met dit artikel proberen we een prikkelend tegenwicht te bieden aan de, naar onze mening, veelal technocratische, *'IT-minded'* benaderingen van het onderwerp low/no-code.

INZICHT 1

Low/no-code kan een katalysator van cultuurverandering zijn

Zo nu en dan komen wij professionals en organisaties tegen die een pittige veranderkundige stelling innemen: “Wij zijn er niet klaar voor. Dat wat jij roept over low/no-code, dat kan zo maar niet, dat past écht niet bij onze organisatie. Dat kunnen ‘ze’ hier niet.” De stellingname zou zo maar eens een veranderkundig feit kunnen zijn. De crux zit echter niet in deze analyse, maar in de onverstandige vervolgactie die wordt geadviseerd: “en dus moeten we eerst zorgen voor een cultuurverandering”. Nu ben ik zelf ook opgeleid aan de hand van de klassiekers van professor en veranderkundegoeroe John Kotter, maar ik hoop maar (met terugwerkende kracht) dat hij zijn fameuze acht stappen van succesvolle verandering niet als een zuiver sequentiële waterval ziet/zag…

Onze ervaring is namelijk dat cultuur- en gedragsverandering alleen effectief te beïnvloeden zijn als ze direct gepaard gaan met ‘doen’. Je verandert geen gedrag met een cursus of intervisiesessie van een paar uur of een aantal dagdelen. Je verandert wél gedrag als je de kennis en inzichten vanuit de genoemde ‘klassieke’ leer- & ontwikkelinstrumenten direct aanvult en inkleurt door iedere dag en iedere week in te zetten op die verandering door het te laten zien en door het (voor) te doen.

Dat leidt tot verrassende resultaten. Denk aan vermeende “digibeten” die razendsnel de principes van agile-ontwikkeling doorgronden. Soms zelfs beter dan MT’s, directies of besturen, getuige het feit dat deze laatste groep ons nog steeds vragen stelt als “maar wat heb ik dan precies eind 2021”.

Zodra je gaat ‘doen’ – een woord dat van nature uitstekend past bij low/no-code – wordt duidelijk dat agile-principes eigenlijk heel natuurlijk zijn, maar dat we dat natuurlijke gedrag in de kantooromgeving af zijn geleerd. Laat woorden zoals cultuur, bewustwording, urgentiebesef of een ander woord uit het veranderkundig woordenboek nooit een reden zijn om het daadwerkelijk aan de slag gaan met low/no-code uit te stellen. Technologie is namelijk een katalysator van cultuurverandering.

De betekenis van het woord *disruptie* (namelijk: ontwrichting of uiteenrukking) onderstreept dit inzicht. Want als low/no-code écht disruptief is – en dat is het! – dan horen wrijving en lastige gesprekken er per definitie bij. Dit ‘knetteren’ is geen teken dat het niet goed is voorbereid. Het is ook geen teken dat er een probleem is met *governance*. En het is zeker geen teken dat je er als organisatie niet klaar voor bent. Wat het wel betekent is dat je bliijkbaar noemenswaardig, ‘wrijvingswaardig’ aan het transformeren bent. Laat het gebeuren!

INZICHT 2

Veel aandacht voor het besturingsmodel creëert mogelijk een illusie van succes & verandering

Als de organisatie professioneel is omgegaan met de selectie van het low/no-code-platform (waarover wij als BCE overigens ook al een artikel schreven), dan is men vaak vastbesloten om die professionele lijn vast te houden als de organisatie zich voorbereidt op wat komen gaat. Die voorbereiding omvat natuurlijk ook het besturingsmodel van het project of programma en in het verlengde daarvan beheer & onderhoud. Klinkt goed. Maar… "iets heel goed voorbereiden" activeert in het brein van veel professionals de fascinerende laten-we-heel-veel-regelen-en-vastleggen-knop. Documenten schrijven, checklists maken, overleg- & verantwoordingsstructuren vastleggen, roldefinities, matrices met allerlei weldoordachte zaken… en ongetwijfeld nog veel meer. Sommigen gaan zo ver dat de introductie van low/no-code ook de uitrol van *Scaled Agile Framework* (SAFe) impliceert, zelfs al komen ze niet eens in de buurt van de omvang die nodig is om ook maar een seconde na te denken over zo'n raamwerk.

Een kritische noot over deze 'zware' insteek leidt dan tot lastige gesprekken en verontwaardigde reacties in de lijn van "binnen onze organisatie is het echt belangrijk om X of Y goed te organiseren hoor" en "we sorteren voor op de toekomst". Oké, prima, maar ons enige punt is: voordat je het weet is het low/no-code-team vooral bezig met het spelen van het spel. Meer met het proces en betrokken poppetjes dan met het resultaat en de beoogde effecten. De energie is er dan al uit voordat ze überhaupt begonnen zijn. De bedoeling is uit het oog verloren en de kans op een enorm succes is daarmee direct gehalveerd.

Het alternatief is om te beginnen met een minimaal besturingsmodel – *'lightweight governance'* – om vervolgens alle ruimte te geven aan een sterk low/no-code-team. Zo'n team is prima in staat om het besturingsmodel 'agile' door te ontwikkelen. Denk heel praktisch aan Scrum's *retrospective* waarin ook het besturingsmodel een terugkerend onderwerp kan zijn en waar anderen – zoals de opdrachtgever(s) – op uitnodiging kunnen aanschuiven. Want hoe dan ook: het is voor de menselijke psyche veel makkelijker iets toe te voegen aan een 'te licht' model waarin nog wat ontbreekt, dan om later iets af te breken wat onnodig is in een 'te zwaar' model. Wouter Hart stelt in zijn boek *Anders vasthouden* (wat overigens over organiseren in algemeenheid gaat en niet over software-ontwikkeling) precies de juiste vraag: hoe zou de besturing eruitzien als het nemen van (professionele) verantwoordelijkheid de enige mogelijke uitkomst is?

De kans dat je een *high-performing* low/no-code-team creëert in een 'zware' omgeving – d.w.z. veel bureaucratie, veel overlegvormen, veel verantwoording, veel hierarchie – waarop het team géén invloed uit kan oefenen, is echt heel klein. Laat het starten met low/no-code zeker niet de aanleiding zijn om dan ook maar gelijk 'eventjes' iets als *Scaled Agile Framework* (SAFe) uit te rollen. Trap niet in de valkuil dat iedereen ontzettend druk is, alles perfect & prachtig is uitgewerkt, maar er feitelijk weinig werkende oplossingen zijn. *Less is more;* loslaten zorgt gegarandeerd voor meer échte resultaten.

INZICHT 3

Een *high-performing* team is altijd de nummer één van alle hefbomen

Het begrip *high-performing* team kan wat kriebels geven, maar toch herken je dergelijke teams wel als je ze meemaakt:

- Ze zijn super gemotiveerd en hebben er echt zin in. Een tegenslag leidt ze beslist niet af; ze blijven positief en constructief.
- Zo'n team heeft als geheel alle benodigde competenties en bij voorkeur blinken de afzonderlijke teamleden ergens in uit.
- Ze hebben als team het mandaat om besluiten te nemen. Zowel wat betreft het eindproduct, het interne ontwikkelproces als ook de externe processen (richting opdrachtgevers en gebruikers).
- Het team werkt intensief en op z'n minst het leeuwendeel van de week samen. Zo niet, dan ontbreekt focus en is er geen basis om een hecht team te worden.
- Het team is zo klein mogelijk en dat vereist dat de teamleden geen *one trick ponies* maar zogenaamde *M-shaped professionals* zijn.
- De teamleden zitten fysiek dicht bij elkaar. Wij kunnen vanuit BCE zeker niet voor alle softwareprojecten van de wereld spreken – waarin iedereen fantastisch lijkt te kunnen near/off-shoren – maar onze ervaring is dat dicht bij elkaar zitten (op z'n minst) een enorme pre is.
- Het team zit dicht bij de klant (als in: op max. twee minuten loopafstand).
- Het team heeft nauw contact met de *business owner*. De *business owner* is proactief, investeert veel tijd en interesse in het initiatief, heeft een stevige positie en is een invloedrijke ambassadeur/sponsor.
- Het team voelt de vrijheid om op een positieve en constructieve manier weerstand te bieden aan de gebruikersorganisatie als zij 'onverstandige' dingen vraagt.

Over de kenmerken in dit lijstje kunnen we het vermoedelijk snel eens zijn, maar als MT, stuurgroep of *business owner* heb je dan nog wel een paar cruciale vragen te beantwoorden:

1. *Hoe krijg je zo'n team bij elkaar?* Dat is ongelofelijk lastig, zeker in een zeer krappe IT-markt en misschien een nog wel krappere low/no-code-markt waarin de toevoer ook veel in de vorm van zij-instromers (uit non-IT) komt. Complicerende factor is dat door voorgaande ontwikkeling het aantal jaren ervaring van een ontwikkelaar bijna non-informatie is geworden.
2. *Hoe zet je dat team in zijn kracht en geef je het al het vertrouwen?* Als je jezelf als opdrachtgever betrapt op controle-, overleg-, rapportage- en stuurdrang, dan vertrouw je blijkbaar niet op het team. En als je niet op het team kan vertrouwen dan heb je echt maar twee keuzes: (1) het team vervangen of (2) in de spiegel kijken en jezelf streng toespreken vanwege deze traditionele, non-agile/non-lean managementstijl.
3. *Ben je bereid om hard in te grijpen wanneer nodig?* Het is een onaardige boodschap, maar er zijn maar bijster weinig *high-performing* teams, ook al hoor je altijd iets anders op eindejaarborrels. Dit vraagt zeker in de vroege fase – waarin het team zich nog vormt – om extra alertheid. Als je het allemaal maar laat gebeuren en *low-performance* accepteert, dan sla je het leven al vroeg uit het team en is de verkeerde weg ingeslagen. En dat terwijl je nog jaren vooruit wilt met low/no-code!
4. *Hoe borg je dat (1) ontwikkelteams uitstekend "gevoed" worden (2) de resultaten optimaal worden ingezet?* In de praktijk lijkt het moeilijk voor 'de business' om te zorgen dat een team soepeltjes kan werken. Consistent keuzes maken (op epicniveau) blijkt soms lastig. De voortdurende stroom van wensen & feedback komt mogelijk moeilijk op gang of is van onvoldoende kwaliteit. Effectief implementeren valt tegen. Demotiverend voor het team is dat de gebruikersorganisatie vaak onvoldoende beseft dat een ontwikkelteam niets liever wil dan een kritische gebruikersorganisatie die aanklopt en zegt "dit is nog niet goed genoeg, ik moet doel X halen en daar heb ik nog voor nodig dat A en B worden ontwikkeld".

INZICHT 4

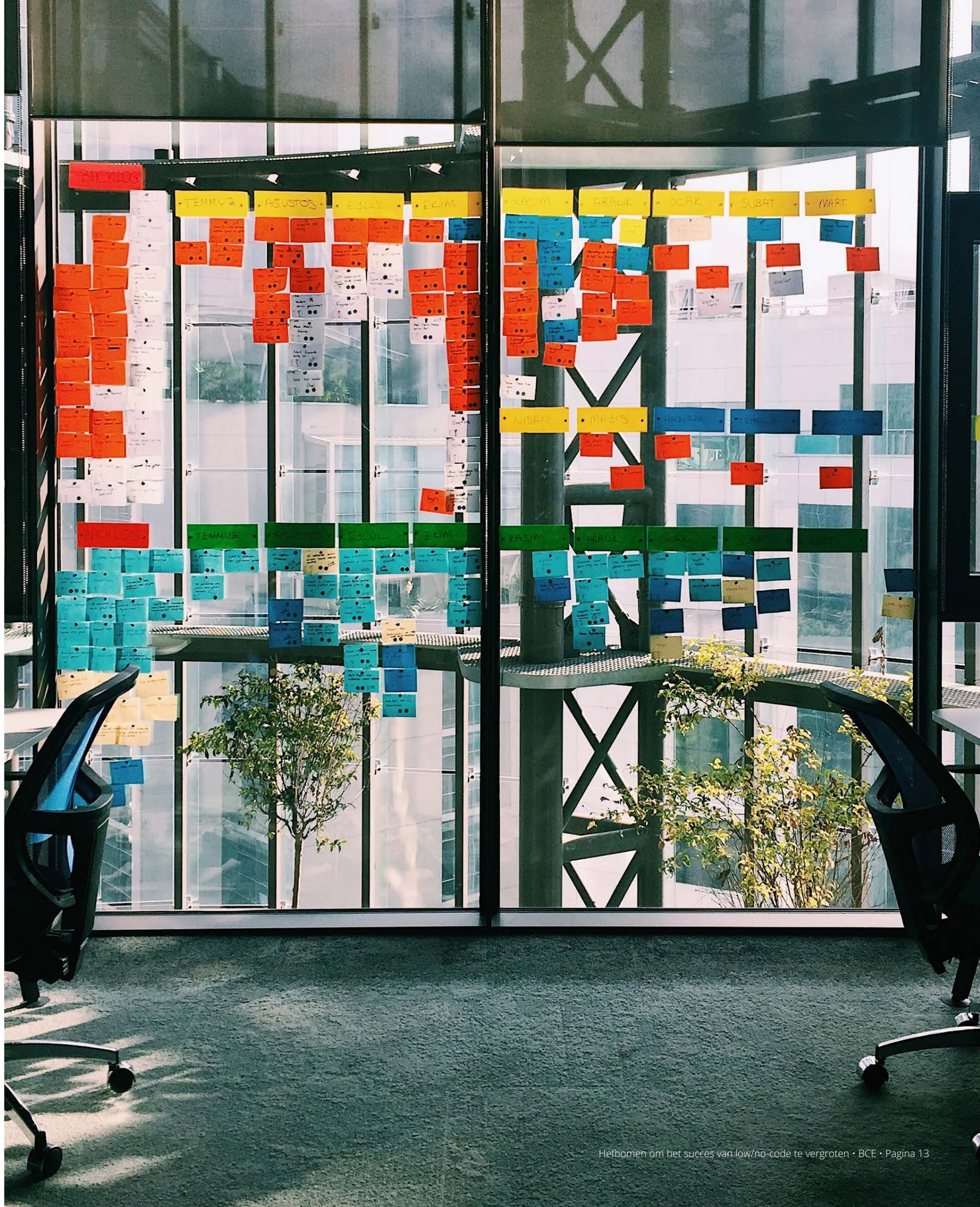
Gezond boerenverstand is belangrijker dan Scrum

Rondom de implementatie van *agile practices* – zoals Scrum – lees je *lessons learned* in de trant van: “het makkelijkste wat je kunt doen is om Scrum op de letter te implementeren”. We snappen en begrijpen de strekking. Immers, voordat je het weet staat er helemaal geen fundament meer en noemt men het desondanks toch Scrum… en heeft men er vervolgens ook nog eens een mening over. Toch is het ook een gevaarlijke boodschap. Je legt dan namelijk dingen buiten jezelf, want “de methodiek zegt het”. Begint het denkproces voor de oplossing bij de externe norm of bij de vraag wat er werkelijk nodig is, is de terechte vraag die Wouter Hart stelt in zijn eerder genoemde boek (dat helemaal niet over Scrum gaat overigens).

Verstandiger is om uitgebreid tijd te investeren in het ‘waarom’ achter alle *practices* en je te laten begeleiden door iemand die dat al vaker gedaan heeft. Tegelijkertijd: Scrum is anno 2019 helemaal niet meer nieuw. Misschien komen we als collectief nu wel in de fase waarin we het advies van de allereerste zin van dit inzicht ontgroeid zijn en is nu de tijd rijp om veel adaptiever naar raamwerken en methodieken zoals Scrum te kijken.

Hoe dan ook: Scrum heeft genoeg rituelen waarin je jezelf compleet kunt verliezen: de sprint, allerlei vormen van afstemming (zoals de *daily stand-up*), *story points*, *velocity*, enzovoort. Achter ieder van die concepten zit een sterk verhaal, maar de les die wij hebben geleerd is: durf ook dingen niet of anders te doen. Een heel simpel voorbeeld: er is niet iedere dag een *daily stand-up* nodig (je zit als team letterlijk naast elkaar dus communicatie is - als het goed is - een veel organischer iets).

VERVOLG ››



›› VERVOLG INZICHT 4

Los van overlegrituelen zoals de *daily stand-up*: laten we – ter illustratie van onze stelling dat je dingen niet moet doen – ook eens naar het begrip *velocity* kijken. Je maakt daarmee vooraf een inschatting van het aantal *story points* – als relatieve omvangsmaat – van een specifieke functionaliteit. Gedurende de sprints ontstaat er inzicht in hoeveel werk (uitgedrukt in *story points*) een team aan kan. Puur ter voorspelling van komende sprints. Dat introduceert een hele lading kritische vragen; vragen waarbij wij nog maar een enkeling zien stilstaan. Juist in de context van low/no-code stellen we die vragen nadrukkelijk wél:

1. Staat de productiviteit überhaupt ter discussie dan? Onze ervaring is dat je met low/no-code zo veel kunt produceren dat er helemaal niemand naar *velocity* vraagt.
2. Wie wil er van jou zekerheid over de komende sprints? Wie 'piept' als iets langer duurt? Kortom, voor wie doe je dit eigenlijk? Voor je eigen team? Als je het voor je eigen team doet, denk je dan oprecht dat metrieken meer losmaken dan uitstekende ontwikkelaars die reflecteren op geleverde prestaties?
3. Wie kent er ontwikkelaars die dit (1) leuk vinden en (2) goed kunnen? Op de een of andere manier zit ik zelf altijd in teams die hier niet zo sterk in zijn (zoek de causale verbanden! ;-)). En dus creëer je een schaduwwereld waarvan iedereen eigenlijk al weet: "we hebben er niet veel aan, maar we doen het toch, want het is een prima manier om de omgeving 'stil' te krijgen/houden".
4. Denk je dat *high-performance* en *high-velocity* hetzelfde zijn? Zo ja, denk eens aan wat er gebeurt als er door de loop van de tijd steeds hoger geschat wordt…
5. Wat doet dit met psychosociale factoren zoals teammoraal? Software ontwikkelen is (meestal) op geen enkele manier te vergelijken met muren metselen of kozijnen schilderen… waarom benaderen we het dan toch zo?
6. Stel dat je wél perfect kan schatten; wat heb je er eigenlijk aan? De dynamische praktijk van softwareontwikkeling is nu eenmaal dat gebruikers nieuwe inzichten krijgen terwijl ze het gebouwd zien worden. Scrum wil tijdens een sprint mutaties in specificaties voorkomen, wat leidt tot operatie-geslaagd-patiënt-overleden-situaties; alles is dan conform afspraak opgeleverd, maar het is niet wat men nodig heeft.

Kortom, vaak is de prijs die je betaalt voor 'zekerheid'/planbaarheid veel hoger dan de prijs van soms een keer een weekje later krijgen wat je wilt hebben als organisatie. En dan hebben we in dit inzicht alleen nog maar gekeken naar *velocity* en overlegvormen. Ten slotte: het enige doel van deze 2 voorbeelden is om organisaties te inspireren om te kiezen voor een andere, 'onzekere' vorm van sturing die professionals uitdaagt om op te staan.

"Maar ik heb een lastige opdrachtgever…"

Daar hebben wij als BCE maar één reactie op: als je opdrachtgever gevoeliger is voor grafiekjes en pseudowetenschap dan voor een goed gesprek en het echte verhaal, dan heb je een heel ander probleem op te lossen…

"Dus ik moet me maar niets aantrekken van methodieken…"

Nee, ons betoog is beslist niet om alle concepten maar weg te gooien en een cowboy-tijdperk te herintroduceren. *Daily stand-up*: prima. *Velocity:* prima. Enzovoort. Maar… is dat ook echt wat nodig is in jouw situatie?! Dergelijke vragen moeten altijd gesteld (kunnen) worden.

"Maar ik moet me verantwoorden richting de CFO…"

Natuurlijk heb je vooraf business cases nodig. Anders kom je niet op het punt dat je low/no-code-projecten kunt gaan doen. Echter, daarna zou verantwoording richting de CFO meer op basis van wat je opleverde (in termen van functionaliteiten & waarde) met een *MVP* en het vertrouwen dat dit creëert, dan op basis van wat je nog exact gaat maken in de komende periode.

“Als je tijd centraal stelt middels sprints, dan zie je allerlei kwaliteits- en (ontwikkel)procesellende ontstaan die op geen enkele manier business value creëert.”

INZICHT 5

Met Kanban-principes kun je de overhead binnen het project reduceren

Dit artikel is niet bedoeld om een uitgebreide uitleg over Kanban te geven. Mocht je dit begrip niet kennen, Google er dan op in combinatie met s*oftware development*. Tip: *https://www.atlassian.com/agile/kanban/kanban-vs-scrum*. Let op: Scrum en agile worden soms als synoniemen gebruikt. De meest simpele manier is echter om agile te zien als een denkkader/paradigma en Scrum te zien als een manier/aanpak die dat denkkader concretiseert. Maar daar zijn er dus meer van, waaronder ook Kanban.

Met Kanban is iets af wanneer het af is. Inkoppertje? Niet bepaald, ga maar eens in sprints werken. Dan kom je namelijk (vaker dan je wilt) op het punt dat de tijd op raakt, maar bepaalde functionaliteiten niet (geheel) af zijn. Vervolgens ontstaat er een uiterst voorspelbare keten van acties. Het begint met overleg met de *product owner*, meestal met concessies als gevolg. En dat heeft vaak weer tot gevolg dat je in een halffabricaat hebt geïnvesteerd die in een latere sprint weer gerefactord moet worden. *User stories* worden vervolgens gesplitst. *Story points* moeten uiteraard worden gecorrigeerd. Aan de gebruikersorganisatie is iets uit te leggen. En zo kan ik nog even doorgaan. Conclusie: als je tijd centraal stelt middels sprints, dan zie je allerlei kwaliteits- en (ontwikkel)procesellende ontstaan die op geen enkele manier waarde creëert.

Het vervelende is dat je – aangezien perfect schatten en dus voorspelbaarheid een illusie is (zie ook inzicht #4) – frequent tegen de momenten aanloopt dat de sprint te ruim of te krap is voor de ingeschatte *user stories*. En het spelen van dat spel wat dan plaatsvindt, dat vergt veel te veel energie. Iets wat wij gewoon hartstikke zonde van de tijd noemen. Niemand wordt er blij van. Niemand heeft er iets aan. Het heeft geen waarde. Sterker nog: het is verspilling van tijd waarin je wel waarde had kunnen creëren.

VERVOLG »

›› VERVOLG INZICHT 5

Bij Kanban zit het woord ‘focus’ nog wat meer ingebakken in de benadering dan bij Scrum, wat wil zeggen dat de ontwikkelaars aan de slag gaan met de zaken die de hoogste prioriteit hebben en dat zij die afronden. Punt. En nee, dit betekent niet dat ze geen oog meer hebben voor *goldplating* en oneindig lang aan één functionaliteit bouwen (want ontwikkelaars zijn niet gek). Maar zodra het dan af is, dan *kan* het beschikbaar worden gesteld aan de gebruikers.

Direct beschikbaar stellen (dat is de meest zuivere vorm van *continuous delivery)* is niet altijd verstandig, want dat vraagt veel van de (gebruikers)organisatie. Wij hebben zelf goede ervaringen met het werken in *timeboxes* die staan voor een vaste hoeveelheid tijd (meestal 1 week, maximaal 2 weken). *Timeboxes* worden natuurlijk ook wel ‘sprints’ genoemd, hoewel dat direct de Scrum-connotatie heeft, terwijl wij er zelf toch een iets andere draai aan geven. *Timeboxes* bieden een prachtig implementatieritme voor opgeleverde zaken. Kortom, wat je kunt doen is wel in *timeboxes/sprints* beschikbaar stellen, maar niet in *timeboxes/sprints* ontwikkelen, zoals je dat met Scrum wel probeert te doen. In de praktijk zien wij overigens vaak het omgekeerde: er wordt wel in *sprints* ontwikkeld, maar niet in sprints beschikbaar gesteld. Aangezien onze suggestie Kanban en Scrum in de mixer gooit, is een ‘Kanban vs. Scrum’-discussie compleet irrelevant, maar raakt dit in de eerste plaats het gezonde boerenverstand waarin er een maatwerkaanpak ontstaat die perfect is voor de specifieke context.

Ten slotte: het ontwikkelteam is niet altijd klaar voor een hoger Kanban-gehalte; zeker niet als als het geen *high-performing* team is zoals dat eerder werd geschetst. Soms kunnen teamleden de vrijere insteek van Kanban simpelweg niet aan en hebben ze keiharde deadlines nodig om te kunnen presteren (veroorzaakt door onder andere een studentensyndroom waar overigens de meeste professionals in meer of mindere mate nog steeds mee kampen). In dat geval past Scrum beter. Echter, als je een topteam hebt zitten, toont Kanban wezenlijk meer respect aan de professie van de softwareontwikkelaar: de mix van de kunstenaar, architect, puzzelaar en analyticus. Daarnaast verspil je ook nog eens minder tijd. Een grote win dus waarmee je aantoonbaar sneller software – en toegevoegde waarde – realiseert!

INZICHT 6

Het loont om lak te hebben aan rollen

De mooiste projecten die wij als BCE doen, zijn projecten waarin je de bijdrage van een individu niet simpelweg middels het noemen van 1 of meerdere rollen kan uitleggen. Een rol is voor ons niet meer dan een accent, maar competenties wil je juist roloverschrijdend binnen je team terugzien (zie ook de eerdere opmerkingen over de *M-shaped professional*). Een *product owner* die techniek goed snapt. Een ontwikkelaar die met ideeën komt om een functionaliteit krachtiger te maken. Een ontwerper die in no-time heeft getest (immers, hij/zij heeft de *stories* geschreven). Een ontwikkelaar die twijfels uit over een wens. Een *product owner* die tegelijkertijd ook een uitstekende *scrum master* is. Enzovoort. Wij zijn voorstander van teams waarin één persoon meerdere rollen vertegenwoordigt en waar roloverschrijdend werken en roloverschrijdend feedback geven de orde van de dag is.

Kanban schrijft geen rollen voor, maar voorgaande stellingname is zeker wel een contradictie met Scrum-informatie/literatuur waarin juist het belang van rolafbakening en functiescheiding worden benadrukt. Sterker nog: op bijvoorbeeld scrumalliance.org staan teksten als *when the same person attempts to fill both roles disaster almost always ensues*. Pittige tekst, hoewel ze gelukkig zelf ook aangeven dat *there are exceptions to every rule*.

We snappen het risico – namelijk: geen van de rollen die iemand vervult wordt op het juiste kwaliteitsniveau ingevuld – maar blijkbaar geldt er ook zoiets als *high risk, high reward*. Dus wij blijven refereren aan de eerste zin van dit inzicht waarin wij het over onze mooiste projecten hebben. Ook blijven we ambitieus streven naar het zijn van *the exception to a rule* en hopen we daarmee anderen te inspireren.

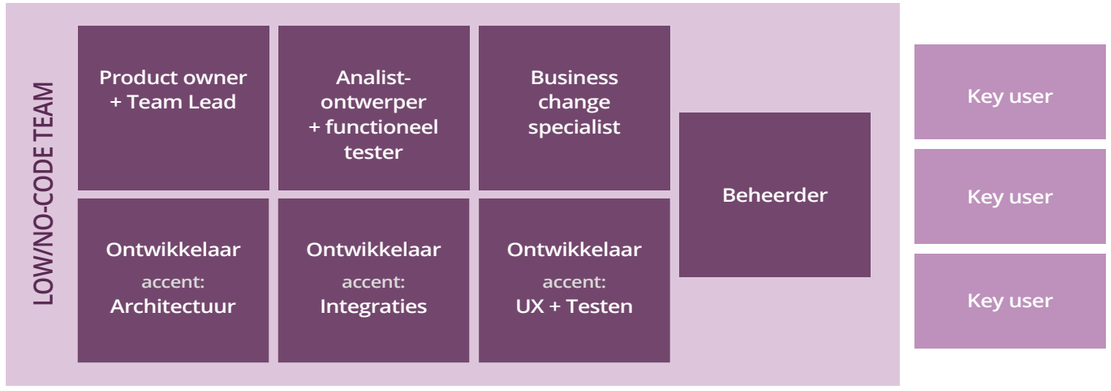
Feit is dat door Scrum-overhead te reduceren en te sturen op een ijzersterk team (waarin enkele personen met elkaar het hele rollenpalet invullen) ongelofelijk veel communicatie/'synchronisatie' te besparen is. Dan heeft bijvoorbeeld die *product owner* ineens wel tijd om ook de *scrum master* te zijn. Dan heeft de analist-ontwerper wel tijd om functioneel te testen.

Als iemand vanuit een *heavy-Scrum-context* hierop reflecteert dan zal hij of zij concluderen: "onmogelijk, ik zou hier absoluut geen tijd voor hebben, dit is een slecht idee". Maar – let op – dat is echt alleen maar waar in die *heavy-Scrum-context* en bovendien een zichzelf vervullende profetie (want maak van iedere rol een apart poppetje en je bent inderdaad erg druk met o.a. communiceren).

VERVOLG ››

›› VERVOLG INZICHT 6

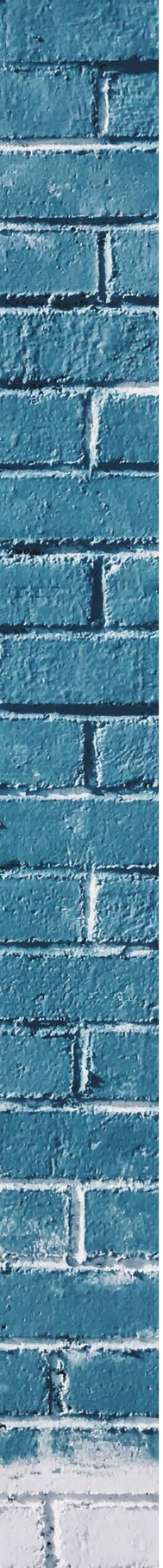
In de onderstaande afbeelding zie je een indicatieve teamopbouw waarover wij in de content van typische bedrijfsapplicaties erg enthousiast zijn. Een paar dingen vallen op:



- Een gecombineerde *product owner* / teamleider (lees *scrum master*). Primaire reden: effectieve en efficiënte sturing van het team, zowel binnen het team als richting de omgeving.
- Meer aandacht voor de backlog in de vorm van een analist-ontwerper die nauw samenwerkt met de *product owner.* Primaire reden: low/no-code gaat zo snel dat je anders te snel 'droog loopt'. De meest bizarre verspilling in een low/no-code-project is natuurlijk een ontwikkelaar die niet verder kan en daardoor zelf spoorzoekertje moet spelen.
- Een *business change specialist* die in nauwe samenwerking met *key-users* zorgt dat opgeleverde zaken bekend zijn, goed worden gebruikt en maximaal worden benut. Immers, mooie dingen opleveren is nog niet hetzelfde als duurzaam veranderen en business cases verzilveren.

- Ontwikkelaars die allemaal een extra accent in het team hebben, waarbij in geval van typische bedrijfsapplicaties Architectuur, UX, Test(automatisering) en Integraties relevante accenten zijn.
- Nauwe betrokkenheid van *key-users* (lees 'degenen die er mee moeten werken'). Er zijn *key-users* die continu intensief betrokken zijn en er zijn er die alleen worden bevraagd als er iets specifieks aan de orde is.
- Op termijn (niet letterlijk op dag één) een beheerder die applicatiebeheer op zich neemt en nauw samenwerkt met de *business change specialist*.

Laten we ter afsluiting nog benadrukken dat het inzicht over boerenverstand ook van toepassing is op wat wij hier zelf beschrijven; neem onze enthousiaste suggesties dan ook niet blind over. Wij hebben het hier namelijk over een typische bedrijfsapplicatie; er zijn genoeg applicaties te bedenken waarin het wel degelijk loont om bijvoorbeeld een aparte UX-ontwerper aan te stellen, enzovoort.



INZICHT 7

In low/no-code-projecten vraagt de backlog om veel meer tijd en energie

De belofte van low/no-code is dat je sneller ontwikkelt. En aangezien wij zowel veel heavy-code- als low/no-code-projecten doen, kunnen wij een behoorlijk onpartijdige, onafhankelijke conclusie delen, namelijk: ja, de low/no-code-belofte is waar. Ikzelf (met een achtergrond in heavy-code) moest een aantal jaar geleden overigens oprecht niets hebben van low/no-code, maar zeker in het licht van de ontwikkeling die low/no-code doorgemaakt heeft en de komende jaren nog zal doormaken, is low/no-code wat mij betreft het nieuwe normaal. Er zijn echt nog wel *use cases* te bedenken voor heavy-code, maar dat lijstje wordt nu wel korter.

Een neveneffect van de snelheidswinst van low/no-code is een veelgehoorde klacht vanuit low/no-code-teams: “wij ontwikkelen veel sneller dan de organisatie wensen kan formuleren”. Want als er *N* x sneller ontwikkeld wordt, moet je ook *N* x sneller je backlog op hoge kwaliteit hebben en kost dit ook *N* x meer tijd. De *product owner* en de analist-ontwerper hebben hier een serieuze kluif aan, waarbij professionele tooling (zoals Atlassian’s Jira) onmisbaar is. De *product owner* is meer op scenario-/epic-/featureniveau bezig en de analist-ontwerper levert aanvullende denk- en doekracht op de details (*user stories*), hoewel dat prima kan overlappen met de rol van de *product owner*. Ten slotte zijn de *business change specialist* en beheerder natuurlijk ook ‘backlogvoeders’ aangezien zij het dichtst op de gebruikersorganisatie staan en ongelofelijk veel waarde kunnen toevoegen als zij met hun voelsprieten continu de meest relevante dingen signaleren.

Een goede user story geeft positieve energie, is afgestemd, heeft direct toegevoegde waarde (nee, dat is niet vanzelfsprekend!) en na realisatie bewijst de praktijk dat de *cost of avoidable rework* van deze user story laag is. Dat laatste subfacet is belangrijk. *Avoidable rework* gaat over slecht analyse- & ontwerpwerk met onnodige herstelcycli als gevolg. We doelen nadrukkelijk niet op voortschrijdend inzicht/verfijningen, want dat is nu eenmaal de praktijk van productontwikkeling die we met veel plezier omarmen.

Uitgaande van een typische bedrijfsapplicatie, is de uitwerking van een user story zo visueel mogelijk (in de vorm van *mock-ups)*, maar wel aangevuld met duidelijke acceptatiecriteria. Leg daarin geen dingen vast die iedereen weet, maar je desondanks toch (bijna dwangmatig) vastlegt omdat je van jezelf (of een methodiek) ‘volledig’ moet zijn. Wij maken teams mee waarin men op een gegeven moment zo goed op elkaar ingespeeld is, dat je letterlijk aan een paar regels genoeg hebt. Andersom zijn er ook teams waarin niets vanzelfsprekend gaat en werkelijk ieder acceptatiecriterium uitgeschreven dient te worden, hoe vanzelfsprekend ook. Bijvoorbeeld dat er op een veld met de naam ‘e-mailadres’ toch echt wel een e-mailadresvalidatie moet zitten. Of dat de vormgeving van de nieuwe functionaliteit moet aansluiten op de rest van de applicatie. Als dat laatste gebeurt: zie dan inzicht #3 over het *high-performance* team, want dat is een teken aan de wand. Ook al geeft een ‘methodist’ je misschien wel de correcte doch onjuiste (ja, het staat er goed!) feedback “dan moet je het maar in de *definition of done* zetten”.

INZICHT 8

Een ijzersterk project begint met een glashelder architectuurontwerp

Met low/no-code ontwikkel je makkelijk en snel. Wil je X? Zo, daar heb je X. Wil je...? Enzovoort. Vertaald naar een klusser: als ook in & rondom het huis klussen nog makkelijker zou zijn, dan zou het bijvoorbeeld in mijn tuin een spaghetti van hutjes worden. Niet goed doordacht en met alle consequenties van dien. Zo zou ik mijn afvalcontainers niet meer aan de weg kunnen zetten en geen BBQ's meer kunnen organiseren. Zo is het eigenlijk ook met software. Creëren is te gemakkelijk en te toegankelijk geworden. Veel low/no-code-initiatieven gaan te snel aan de slag om gebruikers blij te maken. En nee, dit is beslist geen betoog om maandenlang te studeren, maar zorg op z'n minst wel dat er vooraf een duidelijk beeld is van de contouren/grenzen van de te bouwen applicatie(s). Een domeinmodel (ook wel klassemodel of ERD genoemd), een scherminteractiemodel en een lijst met benodigde integraties zijn in veel gevallen al een uitstekend vertrekpunt.

Het begrip 'architectuur' hoort natuurlijk thuis in deze discussie, maar: wat is architectuur eigenlijk? In de praktijk zien wij dat architectuur vaak betekent dat er een techneut is die zijn vocabulaire & gereedschapskist opdringt aan mensen die zijn taal helemaal niet spreken en zich ook niet herkennen in wat uiteindelijk dan 'de architectuur' heet.

De definities vanuit TOGAF en de talloze architectuurrichtlijnen van specifieke low/no-code platforms laten we om die reden graag even voor wat het is. Er is maar één ding om nu mee te nemen: een architectuur is in de eerste plaats 'iets' (zoals een visueel model) waardoor relevante betrokkenen een *aha-erlebnis* hebben. Is het daarmee voor iedereen duidelijk wat de richting is? En is dat ook te merken tijdens de uitvoering van projecten? Er zijn dus geen momenten waarop je denkt "die heeft de basis toch niet begrepen"? Dan heb je een uitstekend architectuurontwerp; laat niemand je iets anders wijs maken.

VERVOLG ››

"Creëren is te gemakkelijk en te toegankelijk geworden. Te veel low/no-code-initiatieven gaan te snel aan de slag om gebruikers blij te maken."

›› VERVOLG INZICHT 8

Dat er vervolgens ook nog eens een technische architectuur is in het low/no-code-platform, dat is vooral het werk van de architect & ontwikkelaars en is grotendeels onzichtbaar voor de *product owner* en gebruikersorganisatie. Iedere zichzelf respecterende ontwikkelclub die met low/no-code werkt (of jullie organisatie op gang helpt) heeft *best practices* & *standards/guidelines* op dit vlak. Maak je daar als non-techneut niet te druk om en zorg dat je er niet te veel energie in stopt om het te begrijpen. Als je te vaak uitspraken hoort in de trant van "in platform X werkt het als volgt" of "in platform X heet dit zo", dan is de te stellen gewetensvraag of wel de juiste gesprekken gevoerd worden.

"Hoe verhouden de backlog en architectuur zich tot elkaar", is ook een vraag die we regelmatig krijgen. Een goede architectuur geeft structuur en een vocabulaire aan de backlog. En andersom: het werken in een backlog – schrijven van stories, maken van schetsen – zorgt dat je patronen ontdekt die in de architectuur thuishoren. In sommige backlogs is samenhang echter lastig te zien, o.a. door simpele dingen als woordgebruik en formuleringen. Het lijkt één grote verzamelbak met allemaal wensen zonder een duidelijk patroon.

Het loont om tijd en energie te investeren om die patronen er uit te destilleren (van *epic* naar *feature* naar *user story*). De naam van een *user story* zou bijvoorbeeld kunnen zijn: *CRM-applicatie* > *360-graden klantbeeld* > *Weergave klantdetails*. Zo'n extreem simpele afspraak/naamgevingsconventie geeft rust en duidelijkheid. Alleen al met de naam is het dan mogelijk om *stories* veel beter te duiden. Er is dan nog geen letter inhoud, nog geen acceptatiecriterium geschreven, maar voor het team is toch al veel duidelijk.

Interessant is dat je gemakkelijk kunt doorslaan met architectuur als onderwerp. Systeemdenken kent het fascinerende verschijnsel 'compenserende feedback': het verschijnsel dat goed bedoelde ingrepen in het systeem een reactie opwekken die – hoe paradoxaal het ook voelt – de voordelen van de ingrepen teniet doen (en soms nog erger). Vertaald naar architectuur zijn er 2 van dergelijke valkuilen:

1. Hoe meer modellen je maakt, hoe completer en 'beter' ze zijn, hoe minder (!) de kern over lijkt te komen bij de betrokkenen en hoe minder ze in de praktijk ingezet worden en hoe meer blijkt dat de denkbeelden niet synchroon lopen.
2. Er wordt in de technische architectuur voorgesorteerd op allerlei hypothetische en/of uitzonderlijke toekomstige wensen en situaties.

INZICHT 9

Veel klant-/gebruikersorganisaties hebben aanvankelijk veel positieve ’weerstand’ nodig

Of je nu een externe of interne medewerker bent: als low/no-code nieuw in jouw (klant)organisatie is, dan heb je in allerlei opzichten een mooie uitdaging voor de boeg, waaronder het effectief managen van de omgeving. De omgeving – waaronder jouw leidinggevende/opdrachtgever/stuurgroep – kent namelijk alleen ‘de oude wereld’ en zal je aanvankelijk ook met de denkkaders vanuit die oude wereld aansturen. Als je dan oprecht gelooft in ‘klant is koning’, dan loopt je daarmee direct het risico dat je je eigen succes ernstig ondermijnt.

Een praktisch, veelvoorkomend voorbeeld is een ervaren opdrachtgever die gewend is dat IT-projecten (1) duur zijn, (2) steevast langer duren dan verwacht en (3) dat het effect/rendement altijd tegenvalt. Met alle goede bedoelingen blijft hij/zij daarom maar drukken op ‘beter uitzoeken’, ‘meer afstemmen’, ‘goed doorrekenen’, enzovoort. En zo kom je te lang niet aan ‘doen’ toe, terwijl jij weet dat experimenteren er juist ook bij hoort en dat low/no-code dat uitstekend en snel kan faciliteren. Sterker nog: vermoedelijk is dit goedkoper dan al die analyses en afstemmingen als je ze zou omrekenen naar geld.

In dat soort situaties is het verleidelijk om wrijving uit de weg te gaan. Ons appel is: doe dat niet! Want vergeet niet dat – als het allemaal tegenvalt – diezelfde gebruikersorganisatie niet zal aarzelen om stickers te plakken als “low/no-code valt hartstikke tegen”. Zo zijn er nog vele andere contraproductieve dynamieken (zie kader rechts) waarmee je als team iets moet als je productief wilt blijven en low/no-code nu en in de toekomst als strategisch instrument wilt kunnen inzetten.

Andere voorbeelden van negatieve dynamieken om positief te beïnvloeden

- Organisatie heeft een irrationele aversie tegen alles wat ‘maatwerk’ is.
- Organisatie wil één-op-één de oude wereld nabouwen met nieuw gereedschap en mist daardoor allerlei verbeterkansen.
- Organisatie maakt onvoldoende gebruik van bestaande IT-middelen en vraagt jou toch om nieuwe (overbodige) software te ontwikkelen.
- Organisatie weet helemaal nog niet wat ze wil, maar wil wel iets en vindt dat vage ‘iets’ zelf al heel concreet.
- Organisatie weet intern geen consensus te bereiken maar geeft toch de opdracht om aan de slag te gaan.
- Organisatie krijgt het concept low/no-code niet tussen de oren; als je *use case* A hebt gebouwd denken zij dat het dus blijkbaar technologie voor *use case* A is.
- Organisatie draaft door en ineens moet ‘alles’ in het low/no-code-platform.
- Organisatie kiest enthousiast voor het low/no-code-platform, maar tot je verbazing worden er door diezelfde enthousiastelingen ook tegenstrijdige zaken in gang gezet (zoals een nieuwe app die er ineens is).
- Organisatie heeft voortdurend wensen in de categorie ‘nice-to-have’; veel energie gaat verloren aan dingen waar weinig mee gewonnen wordt.
- Organisatie heeft heavy-code-ontwikkelaars en die vinden iets van low/no-code en deze krijgen alle ruimte om zich er ook niet in te verdiepen.
- Organisatie gaat van de hak op de tak en pakt op geen enkele applicatie die je met het low/no-code-platform bouwt echt goed door.
- Organisatie investeert vooral tijd in het besturingsmodel; de resultaten vallen tegen, maar de illussie van succes strooit verrassend lange tijd zand in het ogen van de betrokkenen. Zie ook inzicht #2.
- Organisatie vraagt het ontwikkelteam om hun business cases te maken.

“Voor alles waar je tegen jezelf zegt *hier moeten toch anderen ook tegenaan zijn gelopen*, daar ga je iets voor vinden. ”

INZICHT 10

Met de low/no-code-store kun je ontwikkeling versnellen

Het delen van software – of je het nu componenten, *packages*, *libraries* of complete applicaties noemt – kan met zowel low- als heavy-code. Gelukkig maar, want hergebruik (‘*reusability*’) is een van de krachtigste, meest bewezen concepten van softwareontwikkeling in algemeenheid. Daarom hebben low/no-code-platforms allemaal een ‘store’-achtig iets; een groeiende bibliotheek van handigheidjes tot complete functionele componenten en complete applicaties aan toe die helpen om de ontwikkeling te versnellen.

Voor alles waar je tegen jezelf zegt “hier moeten toch anderen ook tegenaan zijn gelopen”, daar ga je iets voor vinden. Rapportages, chat, validatie, bestandsconversie, integraties met bekende systemen (van Salesforce tot en met Google Maps, Office 365 en IBM Watson), basisfuncties zoals FAQ’s en kennisbanken, … enzovoort. Maak daar gebruik van! De *product owner* en analist-ontwerper hebben hierin de belangrijkste rol, omdat zij als beste de wensen kennen en ook in staat zijn een doordachte *make vs. use from store* afweging te maken.

Maar… de kans dat je voor jouw kernfuncties iets vindt dat naadloos past is klein. Wij zien de store vooral als nuttig voor de meer generieke zaken (*utility/commodity*). Voorkom dat je in de valkuil trapt dat je meer tijd kwijt bent met configureren/aanpassen van een bestaand component dan het je had gekost als je het zelf had gebouwd.

Ook zijn er vaak zo veel componenten dat je door de bomen het bos niet meer ziet, zelfs niet met alle zoek- en filtermogelijkheden die de *stores* bieden. Het toewensen van veel zoek- en probeerplezier is op zijn plaats… ;-)

INZICHT 11

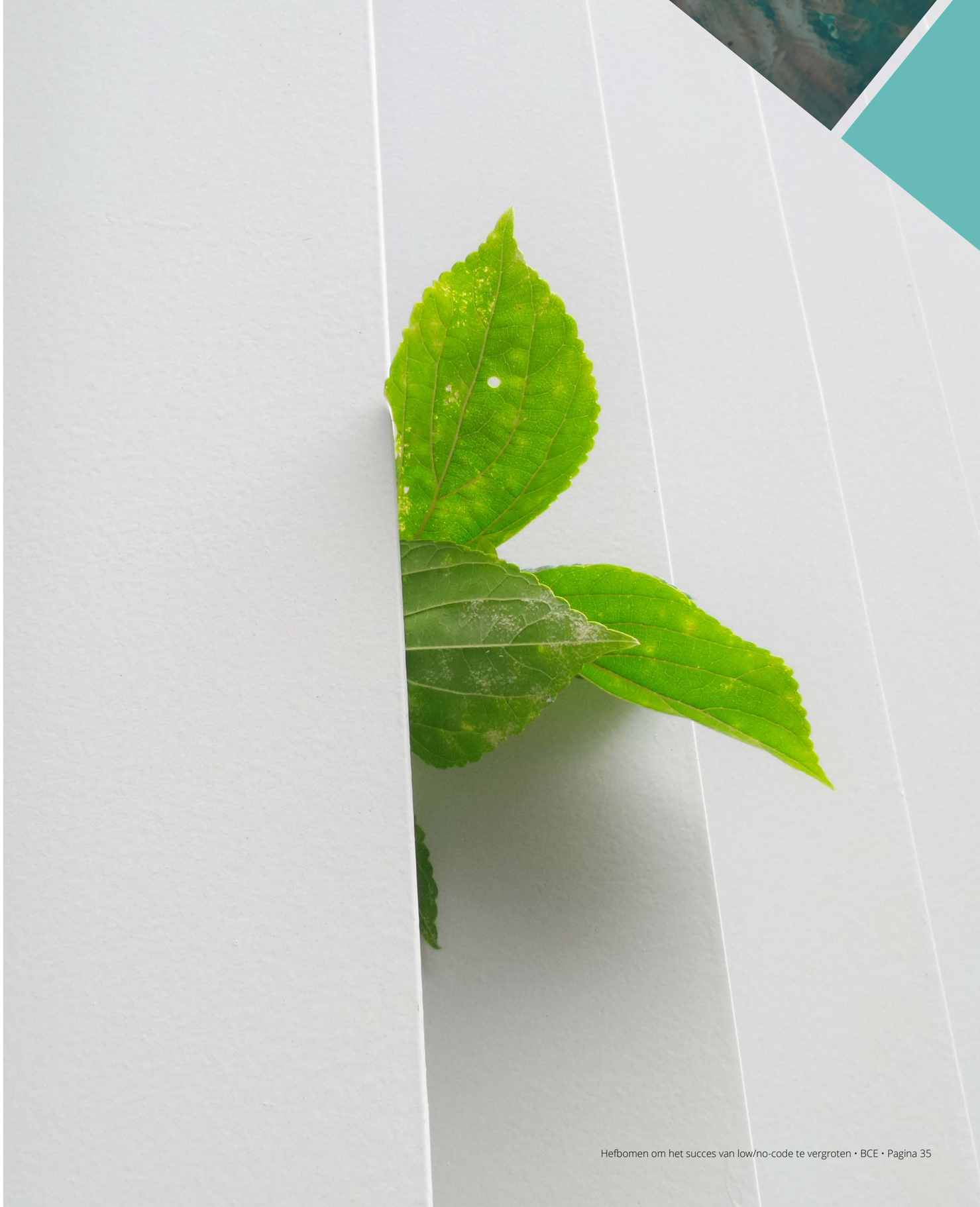
Kanariereleases zorgen voor rust bij softwareopleveringen met veel impact

Razendsnel ontwikkelen & implementeren – laten we zeggen: wekelijks – is natuurlijk top, maar introduceert ook risico's in termen van regressie en implementatie. Ook al heb je zaken perfect afgestemd, ook al is het prima getest, het kan zijn dat de nieuwe functionaliteiten het toch niet goed doen bij de gebruikersgroep. Soms is de stap te groot voor *key-users* om de volledige impact tijdens ontwikkelen & testen te doorzien, zelfs al heb je allerlei praktijkcasussen met ze doorgenomen. Een *big bang release* – in één keer alles voor iedereen – is dan vragen om problemen.

De zogenaamde *canary release* biedt in dergelijke situaties een mooi alternatief, omdat je daarmee feitelijk mini-go-lives hebt voor een kleine groep gebruikers. Alleen zij krijgen dan de nieuwe software. En pas als het naar tevredenheid werkt – wat meestal betekent dat er ondertussen weer een mooi lijstje van verbeterpunten is doorgevoerd – wordt het breder uitgerold. Op deze manier bescherm je het low/no-code-team voor situaties waarin 'de wereld ontploft' bij releases met veel impact… wat onnodig veel energie vraagt van iedereen (van het low/no-code-team én de gebruikersorganisatie).

Het grappige concept van de kanariere release

Voor veel lezers zal een kanariere release vermoedelijk een nieuw begrip zijn. Overigens een begrip dat wij niet hebben verzonnen, maar dat in de wereld van *software deployment strategies* vaker wordt gebruikt. Waarom heet het zo? In de kolenmijnen werden vroeger kanaries ingezet om vroegtijdig dodelijke gassen te detecteren (de kanaries vielen dan namelijk al heel snel letterlijk dood neer). De macabere parallel is dat bij een kanariere release de subset van gebruikers ook wordt gebruikt om vroegtijdig 'gedoe' vast te stellen en zo de rest te beschermen.



INZICHT 12

Testautomatisering is toegankelijker te maken met Robotic Process Automation (RPA)

Het regressierisico groeit recht evenredig met de hoeveelheid software. En die hoeveelheid kan razendsnel groeien met low/no-code. Testautomatisering wordt daarom steeds belangrijker naarmate je ook nog eens steeds sneller wilt releasen. Testautomatisering is zeker niet nieuw en low/no-code-platforms hebben daar allerlei eigen oplossingen en voorkeuren voor. Selenium, Jenkins, Cucumber, JEST, Tosca... enzovoort zijn begrippen die je mogelijk eerder in een testautomatiseringscontext hoorde.

Eerlijkheidshalve moet ik melden dat wij veel organisaties zien die hier niet klaar voor zijn. De stap naar die professionele tooling lijkt te groot, de tooling is te duur of het kost te veel tijd van het ontwikkelteam (en dat betekent dus minder tijd voor softwareontwikkeling). Het effect is dat er verontrustend veel projecten zijn die het daarom maar laten voor wat het is en dat er veel releases zijn waarin of niet of enorm slecht is getest op regressie.

Voor die organisaties kan *Robotic Process Automation* (RPA) een oplossing zijn. Met deze nieuwe ontwikkeling kun je een zogenaamde softwarerobot de belangrijkste functionaliteiten laten testen. Alles wat een mens op een computer kan – vensters openen, klikken, zoeken, kopiëren, inloggen, bestanden aanmaken, mails sturen, enzovoort – kan zo'n robot ook. De robot logt vervolgens zijn testbevindingen in bijvoorbeeld Excel, et voilà, je hebt al een grote stap gezet wat betreft professionaliseren van het testproces. Wat RPA bijzonder maakt is dat de licentiekosten van zo'n robot (denk aan UiPath, AutomationAnywhere of Blue Prism) laag zijn en je het bovendien ook nog eens snel ingericht kunt hebben (kwestie van enkele dagen tot weken).

INZICHT 13

Met Robotic Process Automation (RPA) zijn diverse low/no-code-applicaties nog krachtiger te maken

Voorgaande inzicht introduceerde RPA al even als praktische, haalbare oplossing voor testautomatisering. Het gaat echter veel verder, want RPA kan in bepaalde situaties het succes van low/no-code applicaties noemenswaardig vergroten door een API te ‘faken’.

Om die stelling te kunnen duiden, is begrip van het verschil tussen RPA en low/no-code cruciaal. Het verschil tussen *low-code application development* en *low-code robot development* is dat de eerste leidt tot nieuwe applicaties – schermen, buttons, invoervelden, enzovoort – terwijl de laatste leidt tot het verrichten van handelingen in applicaties die er al zijn (al dan niet met low/no-code gemaakt). De oplettende lezer ziet dat er ook ‘low-code’ voor *robot development* staat. Dat is correct, want net zoals bij app(licatie)s is de bouw van robots inmiddels te versnellen & vereenvoudigen door een verschuiving van heavy-code naar low-code.

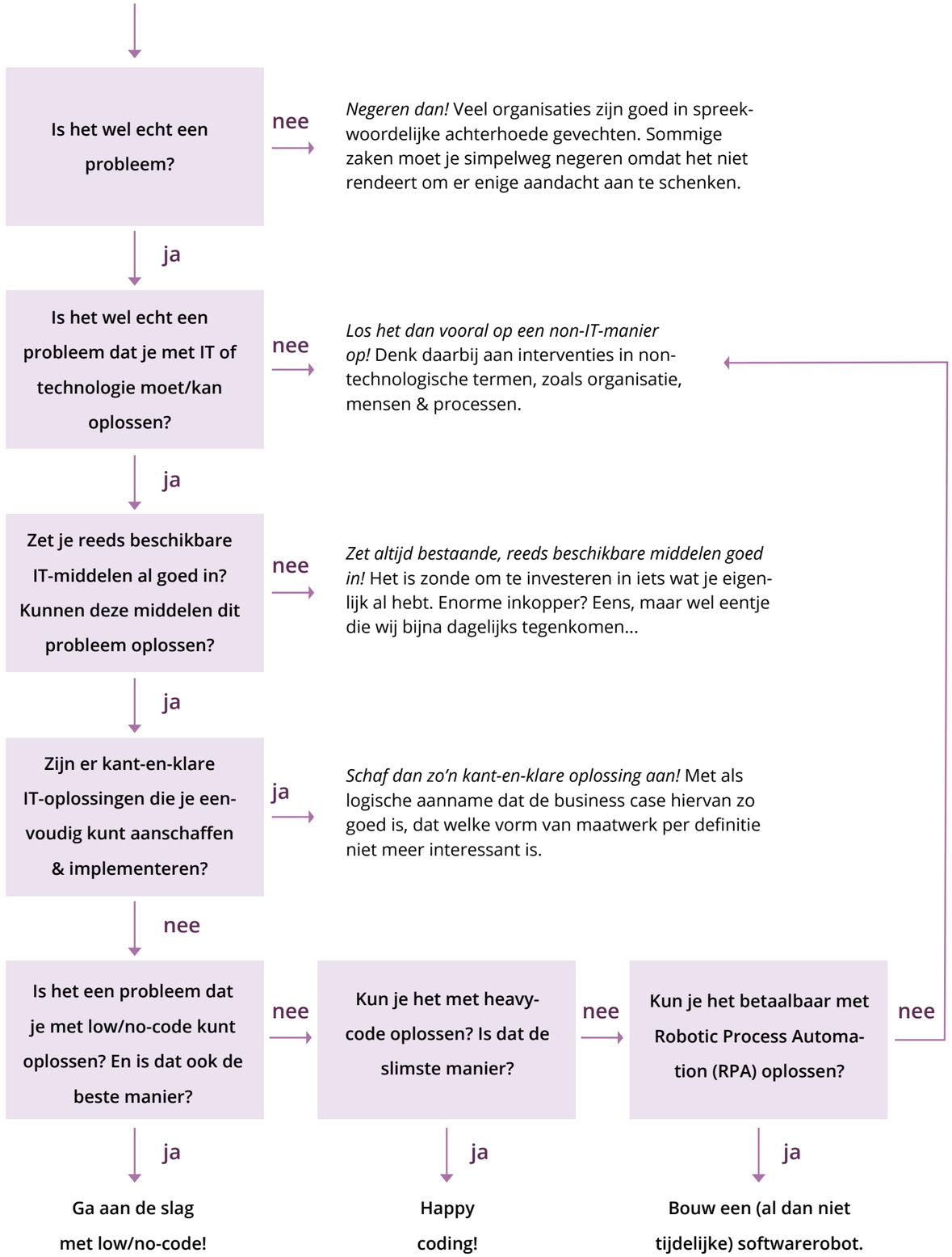
Maar wat betekent dit voor een low-code-project? Het meest klassieke voorbeeld van softwarerobots in combinatie met een low/no-code-applicatie is de integratie van systemen waarvoor de gebruikelijke koppelmogelijkheden (zoals REST of SOAP) ontbreken. Er van uitgaand dat je wel toegang tot deze systemen hebt (via de gebruikersinterface), is het alternatief dat via de gebruikersinterface een koppeling wordt gesimuleerd. Dat betekent concreet dat, terwijl de gebruiker in de low/no-code applicatie werkt, op de achtergrond door robots in opdracht van de low/no-code-applicatie (1) gegevens worden opgehaald uit andere systemen en/of (2) op de achtergrond gegevens worden verwerkt in die ‘onkoppelbare’ systemen. Vanuit gebruikersperspectief is er feitelijk nu wel een koppeling. Een enorme *usability win*, zeker voor gebruikersgroepen die al in veel te veel systemen moeten werken!

INZICHT 14

Een low/no-code-platform is geen oplossing voor alles

Voor de klant-/gebruikersorganisatie is al die variatie in technologie (lees: alle buzzwords) best lastig: de ene keer wordt er geroepen “maar dit moet echt met een low/no-code-platform gemaakt worden”, dan moet het ineens weer met AI of met RPA en weer een andere keer dan is “nee, dit moet je helemaal niet willen”. Hoe dan ook: low/no-code is geen *magic bullet* voor alle vragen. Soms zijn andere oplossingen nodig.

In de dagelijkse praktijk zien we dat er veel te snel wordt getunneld (in de richting van bepaalde oplossingstypes) en dagen weinig professionals elkaar op dit vlak gezond-kritisch uit. Het effect is dat dit tijd, energie en ontzettend veel geld kost. Bovendien zijn de echte gebruikers uiteindelijk altijd de dupe. Kortom, doorloop altijd een eenvoudige checklist (zie het beslismodel rechts) die als denkkader kan fungeren. Dat voelt als low/no-code-team soms misschien raar – want in zekere zin geef je soms werk terug/weg – maar uiteindelijk win je er op alle fronten mee, waaronder vertrouwen van je opdrachtgever. En nog belangrijker: je wint er energie mee in het *high-performing* team, want niets is zo schadelijk voor het team als dingen bouwen die niet nodig zijn… Want die feedback komt vroeg of laat.



VERVOLG ››

Technisch zijstapje A

Heavy-code zal altijd net iets voorlopen op low/no-code

Het technische eindresultaat van low/no-code is heavy-code. Als jij in een low/no-code-platform bijvoorbeeld een scherm bouwt met daarop een aantal velden met daarachter een eenvoudige *workflow*, dan is er uiteindelijk ook gewoon sprake van allerlei heavy-code, zoals HTML, Javascript en CSS. Er zit in een low/no-code-platform namelijk een vertaler (‘compiler’) die de low/no-code-modellen omzet naar iets wat bijvoorbeeld browsers en andere systemen snappen. Het gevolg van deze werking is dat heavy-code per definitie één stap voor loopt op low/no-code. Immers, als er vandaag een geweldige nieuwe heavy-code mogelijkheid beschikbaar komt, dan moeten low/no-code platforms die nieuwe functie eerst wel een ‘smoeltje’ geven in hun producten. Een praktisch voorbeeld op dit moment is het *Single Page Application (SPA)-principe*. In heavy-code-omgevingen (bijv. Vue, React, Angular) kun je daar meer *usability* mee creëren dan low/no-code-platforms momenteel kunnen. Ik twijfel er niet aan dat low/no-code-platforms dit zullen oplossen, maar tegen die tijd is er wel een nieuw heavy-code-element waar ze moeite mee zullen hebben.

Technisch zijstapje B

Heavy-code is een eis in situaties waarin meer controle nodig is

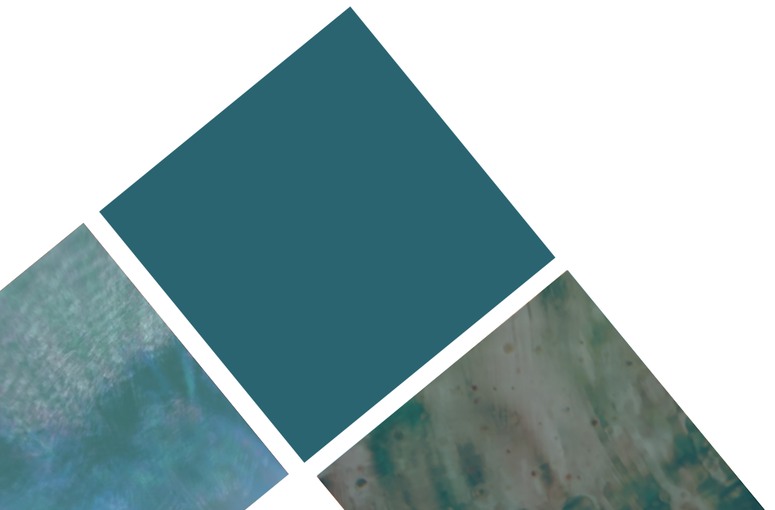
Gezien de vertaalslag vanuit low/no-code naar heavy-code (zie technisch zijstapje A) geef je iets van controle weg. Immers, jij hebt geen grip op de heavy-code die het low/no-code-platform heeft gecreëerd. Die grip heb je in een beperkt aantal situaties echter wel nodig. Denk aan softwaresystemen waarin *performance* en iets als geheugengebruik om een bepaalde reden perfect moeten zijn. Dan biedt heavy-code je meer mogelijkheden om precies het optimum te realiseren dat nodig is.

“Doorloop altijd een eenvoudige checklist die als denkkader kan fungeren. Uiteindelijk win je er op alle fronten mee, waaronder vertrouwen van je opdrachtgever.”

Ten slotte

De variëteit van de inzichten in dit artikel bewijzen wel dat er ongelofelijk veel facetten zijn die low/no-code-succes beïnvloeden. Bijna allemaal facetten die niets met de techniek van het ‘low/no-code-gereedschap’ te maken hebben. Dit artikel is onze eerste versie van inzichten, maar we weten zeker dat we dit artikel zullen blijven bijwerken zodat er een bibliotheek van *good practices* ontstaat. Want net zoals low/no-code zich constant ontwikkelt, zo leren ook wij iedere keer weer wat bij of worden we ons bewust van voorheen onbewuste inzichten.

Met de inzichten in dit artikel probeerden we doelbewust wat tegenwicht te bieden aan de, naar onze mening, veelal technocratische, *IT-minded* benaderingen van low/no-code. We hopen dat dit gelukt is en dat ons verhaal op een constructieve en positieve manier prikkelt!



EEN KORTE INTRODUCTIE

BCE helpt middelgrote organisaties succesvol digitaal te transformeren

Onze missie: middelgrote organisaties helpen om echt het verschil te maken. Met behulp van moderne technologie, bedrijfskundige kennis en gewoon gezond boerenverstand. Digitale transformatie die medewerkers efficiënter en effectiever laat werken. Waar de klant op zit te wachten. En waardoor de prestaties van de organisatie significant verbeteren: niet ‘10% beter’ maar ‘50% anders’.

Onze mensen: gedreven en breed geïnteresseerde professionals met een achtergrond in bedrijfskunde, innovatiemanagement en informatica.

Onze rol: al sinds 2002 zorgen wij dat digitale innovatie bij onze klanten echt werkt. Van conceptontwikkeling tot en met realisatie. Als architect, product owner, programmamanager, projectleider, analist, consultant, strategisch sparringpartner, interim CIO… we zijn niet makkelijk in een ‘hokje’ te stoppen. Altijd in nauwe verbinding met organisatie en medewerkers, want succesvolle innovatie is mensenwerk. Zeker is: jouw digitale transformatie is bij ons in goede handen.

BCE
De meesters in digitale transformatie

WE HOREN GRAAG VAN JE

Meer informatie over low/no-code?

Klaas-Jan Molendijk

Partner & adviseur van BCE

klaas-jan@bce.consulting

06 81 14 13 72

www.bce.consulting