Navigeer naar: #BLOG 1 #BLOG 2 #BLOG 3 #BLOG 4 #BLOG 5 #BLOG 6

Blog 5: architectuur en hoe dat zijn effect heeft binnen het team en over teams heen.

Je ziet ze vaak: de waarschuwingsbordjes om draden niet aan te raken of een ruimte niet te betreden omdat het levensgevaarlijk is. Ook in code doet dit verschijnsel zich voor. Vaak zien we dit verschijnsel bij aanwezigheid van 'legacy'-systemen c.q. veelal in oude programmeertalen. Complexe onderdelen met heel veel belangrijke functionaliteit. De maker is vertrokken, en/of er is weinig of geen documentatie, dus niemand weet precies hoe het onderdeel werkt. Zolang we maar niets wijzigen aan het onderdeel, gaat het vaak wel goed.

Maar in de huidige tijd met de wens tot Agile voortbrenging is weinig meer stabiel. Grote kans dus dat de wens ontstaat om ook zo'n monolithisch onderdeel aan te passen, met als gevolg dat er een gevaarlijke situatie ontstaat. Want wie durft zich te branden aan het aanpassen van de code? De risico's dat er wat belangrijks omvalt zijn simpelweg te groot.

Helaas komt dit probleem ook in Agile voortbrenging voor. Niet direct bij nieuwbouw, maar het probleem ontstaat geleidelijk in de vervolgiteraties. Bijvoorbeeld tijdens de bouw van nieuwe functionaliteit en/of nieuwe koppelingen naar andere systemen waardoor er nieuwe mogelijkheden ontstaan. Langzaam vormt zich zo een kluwen van onderdelen (bijvoorbeeld van applicaties, interfaces en webservices), waarbij het steeds lastiger wordt om de impact van wijzigingen te voorspellen en te testen. Deze onzekerheid over de impact van aanpassingen uit zich vaak in productieproblemen, sprint estimates die niet blijken te kloppen en DoD's die niet worden gehaald. Daardoor schuiven zaken door naar volgende sprints of ontstaat er 'technical debt' (Ontwikkelwerk dat ontstaat doordat gekozen wordt voor een kortetermijnoplossing die voor de lange termijn slecht onderhoudbaar is en dus nog een keer moet worden aangepast). Dit zorgt voor een opeenstapeling van problemen die langzaam maar zeker de velocity (het vermogen om nieuwe functionaliteit op te leveren) van de teams aantast.

Wat we in dit soort situaties vaak constateren, is dat teams (te) weinig tijd hebben besteed aan refactoren van hun bestaande code, ofwel het servicen van hun technical debt. Deze activiteit is steeds ondergeschikt gemaakt aan het bouwen van nieuwe functionaliteit. Vaak speelt er nog een ander probleem, namelijk dat het agile mechanisme voor continue verbeteren (i.e. de retrospective) niet goed werkt of totaal niet geïmplementeerd is. De retrospective is bij uitstek het moment om terug te kijken en je af te vragen waarom er bevindingen zijn in productie of waarom het sprintdoel niet gehaald is.

Naast bovengenoemde problemen kan er ook nog een fundamenteel ander probleem spelen. Dit is met name het geval als er sprake is van koppelingen tussen applicaties die door verschillende teams worden gebouwd. Als er vanuit de architectuur onvoldoende gestuurd

wordt op simpele koppelingen, die bij voorkeur ‘loosely coupled’ (elke afnemer zoekt alleen de strikt noodzakelijke data in het leverende eindpunt en is daarmee zo ongevoelig mogelijk voor aanpassingen op dat eindpunt) worden geïmplementeerd, dan is het risico groot dat een kleine aanpassing binnen een team zich als een olievlek verspreid. Hierdoor moeten meerdere teams ook een aanpassing doen met afstemmings- en testproblematiek tot gevolg. Dit fenomeen hebben we tevens behandeld in Blog 3: End-to-end, ketens en non-functionals.

> Op donderdag 21 september behandelden we tijdens het Polteq webinar het toenemende belang van kwaliteit als onderscheidende factor bij softwareontwikkeling. Tijdens deze online lunch-presentatie gingen we in op het concept testprocesverbetering, methodieken om mogelijke verbeteringen te vinden en te adresseren en hoe je testprocesverbetering integraal kunt inbedden voor continue verbetering.

Bekijk hier de replay van het Polteq webinar.