Lullen en poetsen: van uitdagingen naar verbeteringen

Introductie

Ruim een jaar ben ik door Polteq ingezet bij een Rotterdams havenbedrijf dat wereldwijd actief is. De kernactiviteit van dit bedrijf is goederen opslaan in en uitslaan uit loodsen. Zoals te verwachten bij een havenbedrijf heerst er een mentaliteit van aan- en doorpakken: niet lullen maar poetsen. De directe aanleiding voor mijn aanstelling was dan ook een groot en ambitieus project: de constructie van een nieuwe loods, waar een geautomatiseerd productieproces plaats zou vinden. Een aantal applicaties die door het havenbedrijf in een brede automatiseringsslag aangekocht waren of ontwikkeld werden, zouden bij dit project voor het eerst in gebruik genomen worden. Ik kreeg als taak een van die applicaties te testen.
Dit stuk behandelt tegen welke verbeterpunten ik in het ontwikkelproces van deze applicatie als tester aanliep en welke stappen zijn gezet om deze verbeterpunten aan te pakken. Sommige verbeteringen kwamen op mijn initiatief tot stand, andere kwamen vanuit andere teamleden of het bedrijf voerde een project overstijgende verbetering door. Belangrijker dan waar de verbetering vandaan komt, is dat er binnen het team of bedrijf een overeenstemming is over:

- wát verbeterd moet worden;
- hóe het verbeterd moet worden
- en ook belangrijk: wat de manier van werken is ná het implementeren van een verbetering.

Er moet niet alleen maar gepoetst worden; analyse en afstemming zijn net zo belangrijk. Anders loop je het risico te gaan dweilen met de kraan open.
Eerst geef ik aan tegen welke verbeterpunten in het ontwikkelproces ik aanliep, vervolgens beschrijf ik vier stappen die zijn gezet om het ontwikkelproces te verbeteren en tot slot geef ik aan wat er nu en in de nabije toekomst nog verbeterd kan worden.

Grootste verbeterpunten

De opdracht bij dit havenbedrijf was mijn eerste als tester en zeker in het begin had ik de neiging om mijn (test)werkzaamheden aan te passen aan de methodes (of het gebrek daaraan) die al gaande waren. Maar in de loop der maanden wist ik, dankzij mijn training van Polteq en de opgedane ervaringen, een aantal verbeteringen aan te wijzen in het ontwikkelproces die zouden leiden tot structureel minder fouten in productie en een overzichtelijker ontwikkelproces. Ik was uiteraard niet de enige die deze verbeterpunten zag, maar als tester liep ik regelmatig tegen deze punten aan.

Deze verbeterpunten waren:

1. Geen beschikbare acceptatieomgeving (enkel een ontwikkel-, test- en productieomgeving).
2. Ongestructureerde uitrollen van code naar de test- en productieomgeving, met een tekort aan afstemming.
3. Te weinig afstemming tussen applicaties die integratiepunten met elkaar hebben.
4. Nauwelijks mogelijkheden om de applicatie te testen zonder eerst handelingen in een andere applicatie te verrichten (*stand-alone* testen).
5. Te weinig of geen interactie tussen het ontwikkelteam en de gebruikers.
6. Te weinig of geen prioriteitstelling. Binnen de applicatie, maar ook tussen applicaties.

Elk verbeterpunt heeft van zichzelf een belemmerende werking, maar ze versterken elkaar ook nog eens. In mijn geval was het bijvoorbeeld nodig om eerst handelingen te verrichten in een andere applicatie voordat ik kon gaan testen wat ik wilde testen. Toen er ook nog eens weinig afstemming bleek te zijn met die andere applicatie over de status van wederzijdse ontwikkelingen, was het duidelijk dat deze punten aangepakt moesten worden.

Stappen gezet om proces te verbeteren

Ik beschrijf vier stappen die zijn gezet om het ontwikkelproces te verbeteren. De eerste twee kwamen op mijn initiatief tot stand, de andere werden bedrijfsbreed ingezet.

1. Afstemming code naar testomgeving zetten

Omdat er nog geen acceptatieomgeving was, was het belangrijk om de testomgeving zo stabiel mogelijk te houden. We gebruikten de testomgeving namelijk ook voor het regressietesten. De ontwikkelaars waren gewend om wanneer ze zelf het moment juist achtten code naar de testomgeving te zetten. Het eerste dat ik met hen afsprak was dat zodra de regressietest begon, geen code meer naar test mocht zonder eerst analyse en afstemming. Als nu uit de regressietest een probleem voortkomt, bepalen we eerst óf een fix nodig is, daarna of de fix niet kan wachten tot een volgende release, vervolgens wat de impact zou zijn van de fix en uiteindelijk zetten we de code naar de testomgeving. Vervolgens hertesten we de functies die geraakt worden door de nieuwe code. Hierdoor komt geen ongeteste code meer in productie terecht en het aantal fouten in de productieomgeving daalde significant door deze afstemming. De grootste uitdaging hierbij was het bepalen van de noodzaak voor fixes. Ik moest dus op zoek naar iemand die een direct belang heeft bij het goed functioneren van de applicatie, het volgende punt:

2. Iemand vinden om prioriteiten mee af te stemmen

Voor het bepalen van prioriteiten voor ontwikkelingen of fixes voor de applicatie missen we een vertegenwoordiger van de gebruikers. Nadat we dit hadden aangegeven bij de programmamanager werd besloten dat de procesanalist die het oorspronkelijke ontwerp van de applicatie had gemaakt, de prioriteiten voor nieuwe functionaliteit zou aangeven. Maar zeker na de livegang had ik nog steeds iemand nodig om te helpen bepalen hoe belangrijk het was om bijvoorbeeld een defect te repareren. Ik kwam uit bij de functioneel beheerder van de applicatie. Hij gebruikt de applicatie vaak om issues te analyseren die bij hem terechtkomen. De applicatie geeft namelijk een inzicht in de gebeurtenissen binnen de twee applicaties waar het integratiepunten mee heeft. Het is niet ideaal om met een proces analist en een functioneel beheerder in plaats van met een gebruikersvertegenwoordiger prioriteiten af te stemmen, maar het belangrijkste is een klankbord te hebben in iemand met een belang bij het goed functioneren van de applicatie.

3. Gebruik van issue-tracker (JIRA)

In het begin communiceerden we met name per e-mail. Binnen mijn applicatieteam werd vervolgens de tool Basecamp gebruikt en sinds een half jaar gebruiken bijna alle ontwikkelteams binnen het bedrijf JIRA. Hierdoor is inzichtelijk gemaakt welke functionaliteit wanneer te verwachten is en het is makkelijker afstemming te vinden tussen teams die werken aan applicaties met afhankelijkheden van elkaar. Elke applicatie heeft een eigen JIRA-project. Zo kunnen we functionaliteit die meerdere applicaties beslaat opdelen in aan elkaar gekoppelde taken per applicatie. Hierdoor houden we eenvoudiger bij wat er aan de andere kant van de schutting gebeurt. Er is dus bijvoorbeeld een grotere kans om fouten te voorkomen die ontstaan doordat zonder de nodige afstemming een wijziging in de ene applicatie wordt doorgevoerd die impact heeft op de andere applicatie.

4. Het synchroniseren van releases

In het begin was buiten de grenzen van een ontwikkelteam van een applicatie geen planning, coördinatie of afstemming wanneer voor een applicatie functionaliteit in productie werd bracht. Ook stemden teams die aan verschillende applicaties werken, niet met elkaar af wanneer zij nieuwe functionaliteit naar de testomgeving zetten. Dit had een nog grotere impact vanwege het ontbreken van een acceptatieomgeving en een beperkte mogelijkheid de applicatie die ik test *stand-alone* te testen. Als er dus in een applicatie waar mijn applicatie afhankelijk van was, code naar test werd gezet en er viel iets om, kon dat zeer blokkerend zijn. Sinds drie maanden is er een release kalender, waar de belangrijkste applicaties in de keten in opgenomen zijn. De releases naar productie zijn nu gesynchroniseerd, waardoor getest kan worden in omgevingen die gelijk zijn aan wat na de release ook in productie staat. Ook hoeft er niet voortdurend een regressietest door de gehele keten plaats te vinden, omdat er niet meer voor de ene, dan weer voor de andere applicatie een uitrol naar productie uitgevoerd wordt. Nu is er één moment waarop alle applicaties in de keten naar productie gaan en is er dus een vaste periode om regressietesten door de gehele keten uit te voeren.

En nu verder

Momenteel nemen we een acceptatie omgeving in gebruik, door de gehele keten. Hiermee zullen we bijvoorbeeld een stabiele omgeving hebben voor gebruikersacceptatie testen en kunnen we het regressietesten van de keten en het testen van losse veranderingen beter van elkaar scheiden.

Sommige van de doorgevoerde verbeteringen zijn tijdelijke oplossingen. Zo moet echt nog een vruchtbare connectie ontstaan tussen het ontwikkelteam en de gebruikers. Ook missen we nog een daily stand-up, regelmatige demo's en reguliere evaluatiemomenten. Er is nu goede communicatie tussen analist, ontwikkelaar en tester, maar zeker als we met een gebruikersvertegenwoordiger het team uitbreiden, zullen we een (vorm van) daily stand-up, demo's en reflectiemomenten nodig hebben om bijvoorbeeld voortgang te kunnen monitoren.

Ook het gebruik van JIRA kan nog beter. Nu is het vooral gericht op pure ontwikkelactiviteiten, en nauwelijks op bijvoorbeeld testtaken of procesverbeteringen. Voor de verbeterpunten die ik beschrijf kan je ook heel goed in JIRA taken, user stories of epics maken. Deze zijn net als het bouwen van functionaliteit te prioriteren en in te schalen qua hoeveelheid werk en impact.

Zo zullen er altijd verbeterpunten blijven bestaan. Zodra je de grootste knelpunten hebt weggehaald, zal altijd een volgend punt bovenaan de lijst komen te staan. Je kan bij het doorvoeren van verbeteringen in het ontwikkelproces nooit achterover leunen omdat je een zichtbaar significant resultaat hebt behaald, want voor je het weet loop je alweer achter de feiten aan, en vind je jezelf terug middenin een jungle van elkaar versterkende moeilijkheden. Het is dus van belang om:

- voortdurend scherp te blijven op verbeterpunten;
- de impact van deze punten te bepalen;
- te bedenken welke korte en/of lange termijnoplossingen door te voeren zijn;
- en niet te vergeten: na het doorvoeren van een oplossing te kijken of er een meetbare verbetering is en zo ja, dit vooral te laten zien!

Olivier Mesker