Deel 5: Voordelen van goede en risico's van slechte geïmplementeerde testautomatisering

In de vorige blog hebben we de opbouw van veelvoorkomende test frameworks behandeld.

In deze blog staan we stil bij de voordelen van goede testautomatisering en de risico's van slecht geïmplementeerde testautomatisering en kijken we naar hoe je de kwaliteit van testautomatisering bewaakt.

Voordelen bij goede invoering

Goed ingevoerde testautomatisering levert grote voordelen op. Goed geïmplementeerde geautomatiseerde tests:

- kunnen zelfstandig draaien, ook buiten kantooruren;
- zijn 100% reproduceerbaar en herhaalbaar;
- voeren tests uit met hoge doorloopsnelheid;
- realiseren extra testdekking die met de hand moeilijk of niet haalbaar is.

Risico's bij foute invoering

Testautomatisering kan op vele manieren misgaan waardoor de genoemde voordelen gedeeltelijk of zelfs volledig te niet gedaan worden. Risico's bij het opzetten van testautomatisering zijn:

- Testautomatisering beschouwen als het geautomatiseerde equivalent van handmatige testen en daardoor wordt de heuristiek van de testpiramide niet gevolgd. Hierdoor ontstaan te veel E2E gebaseerde tests die traag verlopen en veel beheerinspanning vragen.
- Niet stilstaan bij het feit dat testautomatisering softwareontwikkeling is. Als bij testautomatisering onvoldoende wordt gelet op codekwaliteit in de vorm van onderhoudbaarheid, schaalbaarheid en beheerbaarheid dan zal het initiatief na een succesvolle, snelle start op de lange duur krakend tot stilstand komen.
- Testautomatisering coderen in een taal anders dan de ontwikkelaars gebruiken. Dit maakt dat testautomatisering op een eiland komt te staan, alleen de maker (Software Developer in Test) kan dan de code onderhouden. Door dezelfde taal te gebruiken als waarin de ontwikkelaars programmeren borg je dat het hele team de testautomatisering kan onderhouden.
- Alles automatiseren. Dit is niet slim. Testautomatisering is niet gratis dus er moet een positieve business case zijn. Alleen testen die vaak herhaald moeten worden (bv de regressietesten) hebben een positieve business case. Verder is het goed om stil te

staan bij het doel van elke test; welk risico of welke requirement dek ik af met de test?

QA van geautomatiseerd testen

Bij geautomatiseerd testen is het belangrijk om stil te staan bij de vraag; hoe borg ik de kwaliteit van mijn tests?

Voor geautomatiseerde tests geldt namelijk, net als voor alles wat door een computer gedaan wordt, dat mensen er 100% op gaan vertrouwen. Ze zijn niet meer kritisch of de test wel juist is. Het is dus belangrijk om tijdens het maken van een geautomatiseerde test, elke test minimaal één keer geconditioneerd te laten falen. Alleen wanneer een test faalt als hij ook zou moeten falen, weet je zeker dat hij het juiste test en kun je op de test vertrouwen.

Er zijn drie typen incorrect tests:

- False positive
- False negative
- Flaky

Een **false positive** test geeft, ongeacht de code, altijd een PASS als resultaat. False Positives zijn gevaarlijke tests omdat ze een schijnzekerheid geven, de test faalt namelijk nooit, ook niet bij een fout in het product! Je mist dus een stuk testdekking.

Een **false negative** test geeft altijd een FAIL als resultaat, ook al is de code goed. False Negatives ondermijnen het vertrouwen in de testautomatisering en versterken de neiging om tests “uit” te zetten. Dit laatste is gevaarlijk omdat voor geautomatiseerde tests het principe

geldt: als 1 schaap over de dam is volgen er meer. Met het tijdelijk uitzetten van een test zeg je eigenlijk dat de waarde van de test niet erg groot is, je durft namelijk ook wel zonder de test door te gaan. Daarmee stel je het bestaansrecht van de test ter discussie!

Een **flaky** (breekbare) test geeft niet altijd hetzelfde resultaat door condities waar geen rekening mee gehouden is. Vaak ligt hier een probleem bij de testomgeving welke resulteert in andere timing en dus een ander resultaat. De onbetrouwbaarheid van een flaky test ondermijnt ook het vertrouwen in testautomatisering.

False positive, false negative en flaky tests zijn alle drie even erg, ze ondermijnen de kwaliteit van de informatievoorziening door testen. Daarom is het van belang om kwalitatief goede tests te schrijven met onderhoudbaarheid als belangrijke vereiste. Hiervoor kunnen dezelfde methodes gebruikt worden als bij de ontwikkeling van normale productcode: code review, gebruik van linters en refactoren. In het geval dat een test faalt is het verstandig om met het team af te spreken dat een falende test nooit uitgezet wordt. Een falende test moet zo snel als mogelijk worden geanalyseerd en gerepareerd of worden verwijderd als mocht blijken dat de test irrelevant is geworden.

Mutatietesten is een interessante methode om de kwaliteit van de testautomatisering te testen. Mutatietesten is het bewust muteren van de code volgens vooraf ingestelde mutatieregels. Hiervoor zijn in veel programmeertalen tools beschikbaar. De tools planten heel specifieke fouten in de code (fault seeding). Goed geïmplementeerde testautomatisering zal de geplante fouten vinden. Het rapport met niet gevonden mutaties is het startpunt voor verbetering van de testautomatisering. Het belang (het productrisico) van de code geeft hierbij richting aan de prioritering van het werk.

Vooruitblik volgende blog

Tot zover deze blog. In de volgende blog staan we stil bij de eisen die testautomatisering stelt aan testomgeving en testdata.

STAY TUNED!

Wim ten Tusscher