6 Ontwerp versus implementatie – de kans om ze niet uiteen te laten lopen

Eric Bouwers

Als men aan het einde van een ontwikkeltraject de geïmplementeerde architectuur vergelijkt met het initiële ontwerp, is vaak de conclusie dat beide niet compleet met elkaar overeenkomen. In dit artikel gaan we in op de vraag hoe deze afwijkingen ontstaan en bekijken we welke middelen beschikbaar zijn om snel op deze afwijkingen te kunnen reageren.

6.1 Inleiding

Het implementeren van een softwaresysteem begint normaal gesproken vanuit een ontworpen softwarearchitectuur. In dit ontwerp zijn de belangrijkste beslissingen vastgelegd met betrekking tot welke componenten er zijn, waar elke component verantwoordelijk voor is en hoe de verschillende componenten met elkaar communiceren.

In het ideale geval komt het ontwerp van de architectuur precies overeen met de implementatie van het systeem; er zijn precies evenveel componenten die alleen via de vooraf gedefinieerde kanalen met elkaar communiceren. Helaas bewijst de praktijk ons keer op keer dat het eerste ontwerp niet hetgeen is wat uiteindelijk wordt gebouwd. Zo zijn er bijvoorbeeld vaak meer componenten in de implementatie te vinden omdat de functionaliteit tijdens het ontwikkelproces is aangepast, of zijn er afhankelijkheden tussen componenten welke men niet verwacht, zoals bijvoorbeeld een afhankelijkheid vanuit een component voor dataopslag naar een component met businesslogica.

Dit soort discrepanties tussen het originele ontwerp en de implementatie zorgen voor onduidelijkheid over de aanpasbaarheid van het systeem, wat resulteert in projecten die langer duren dan gepland met hogere kosten. Dat is bijvoorbeeld het geval als binnen het ontwerp van het systeem het

toevoegen van een andere gegevensbron triviaal lijkt te zijn (omdat deze functionaliteit niet afhankelijk hoort te zijn van andere componenten), maar dit tijdens de implementatie voor grote problemen zorgt (omdat er toch afhankelijkheden gecreëerd zijn tijdens de implementatie waardoor meerdere componenten moeten worden aangepast).

Hoe komt het dat de implementatie van een architectuur uiteindelijk niet overeenstemt met het originele ontwerp? En belangrijker nog, hoe zorgen we ervoor dat beide representaties van de architectuur met elkaar overeenkomen aan het einde van het implementatietraject?

6.2 Waarom wijkt de implementatie af?

Een eerste reden voor een afwijkende implementatie van de architectuur heeft te maken met de grenzen van wat mogelijk is met de gekozen techniek. In sommige gevallen dwingt een technologie een bepaalde architectuur af die niet gelijk is aan het ontwerp. In andere gevallen laat de technologie bepaalde constructies niet toe, wat doorgaans opgelost wordt door functionaliteit toe te voegen aan componenten 'omdat het niet anders kan'. Ten slotte kan een ongebruikelijke combinatie van technieken leiden tot talloze technische problemen, welke niet op te lossen zijn binnen de ontworpen architectuur.

Naast deze technische redenen kunnen afwijkingen ook ontstaan vanuit het proces, doordat bijvoorbeeld nieuwe functionaliteit wordt toegevoegd aan het systeem zonder dat er rekening wordt gehouden met de bestaande architectuur (omdat een belangrijke speler in het proces de functionaliteit snel wil hebben). Eenzelfde situatie ontstaat als er een grote fout optreedt en de architectuur geen snelle oplossing toelaat op een nette manier. In de meeste gevallen zal het ontwikkelteam de fout op een minder nette (maar snellere) manier oplossen. Hoewel dit soort oplossingen van tijdelijke aard zouden moeten zijn, krijgt het achteraf oplossen van dit soort 'schoonheidsfoutjes' geen prioriteit, het systeem werkt immers zoals het moet werken.

Ondanks dat deze voorbeelden niet uitputtend zijn – er bestaan nog veel meer situaties die leiden tot een afwijkende implementatie – illustreren deze categorieën dat er legitieme redenen zijn om af te wijken van het originele ontwerp. Als de beslissing om af te wijken bewust wordt ge-

maakt kan de architect in het beslissingsproces betrokken worden. Hierdoor kan een afgewogen keuze worden gemaakt tussen het aanpassen van het ontwerp en een andere manier van implementeren. Dit zorgt ervoor dat het ontwerp en de code samen evolueren.

In de praktijk blijkt echter dat dit soort beslissingen niet altijd bewust worden gemaakt, omdat ontwikkelaars simpelweg niet altijd op de hoogte zijn van alle details in het ontwerp. Zelfs als de ontwikkelaars het gehele ontwerp kennen, kunnen fouten niet altijd worden voorkomen. Daarnaast is er, door bijvoorbeeld tijdsdruk of vanwege organisatorische redenen, bij een bewuste keuze ook niet altijd de mogelijkheid om met de architect te overleggen. Deze situatie leidt tot suboptimale oplossingen in de implementatie zonder een vermelding in het ontwerp.

6.3 Architectuurevaluaties

De oplossing om dit soort fouten te ondervangen, is om de geïmplementeerde architectuur periodiek te evalueren met het ontwikkelteam en de architecten. Er zijn voor deze evaluaties talrijke methodes beschikbaar, variërend in diepte, uitgangspunt en doel. Door het kiezen van de juiste evaluaties beschikt men na afloop over een gedetailleerd overzicht van de architectuur zoals hij is geïmplementeerd, een eventueel aangepast ontwerp en een overzicht van de zwakke en sterke punten van de architectuur. Deze overzichten dienen vervolgens als basis voor een eventueel stappenplan om de geïmplementeerde architectuur aan te passen en te verbeteren.

Door deze evaluatie na elke release te laten plaatsvinden, wordt ervoor gezorgd dat er altijd een (redelijk) up-to-date beeld is van de huidige architectuur. Daarnaast levert de evaluatie een moment van bezinning op waardoor nieuwe inzichten boven kunnen komen drijven. Als laatste biedt het evaluatiemoment de ontwikkelaars de kans om kritisch naar bepaalde oplossingen te kijken en samen met de architect te brainstormen over mogelijke verbeteringen aan zowel het ontwerp als de implementatie.

Het nadeel van het uitvoeren van dit soort terugkerende evaluaties na elke release is dat problemen pas achteraf ontdekt worden. De logische oplossing hiervoor is om de evaluatie vóór een release te laten plaatsvinden,

alleen wil niemand grote structurele veranderingen doorvoeren vlak voor een release.

Ongeacht het tijdstip van de evaluatie, bestaat er altijd het risico dat de implementatie al langere tijd afwijkt van het ontwerp. Doordat er nu eenmaal wordt doorontwikkeld aan het huidige systeem kunnen deze afwijkingen al sterk verweven zijn in de implementatie, wat het verwijderen van de afwijkingen sterk bemoeilijkt. Om het oplossen van dit soort problemen makkelijker te maken, moeten dit soort afwijkingen van het ontwerp zo snel mogelijk opgemerkt worden. Dit kan door de architectuur dagelijks te evalueren, in de praktijk is dit echter te kostbaar en te tijdrovend om toe te passen.

6.4 Architectuurmonitoren

Gelukkig is er een kostenefficiënte manier om de architectuur dagelijks te 'evalueren', namelijk het gebruik van softwaremetrieken toegepast op architectuurniveau. Tijdens een normaal ontwikkeltraject zullen de ontwikkelaars al gebruikmaken van metrieken om bepaalde aspecten van het systeem op codeniveau te meten. Hierdoor weten ontwikkelaars bijvoorbeeld waar de code ingewikkeld is, welke code op meerdere plekken gedupliceerd is en welke afhankelijkheden er tussen bepaalde stukken code zijn.

Op architectuurniveau kunnen dezelfde soort metrieken gebruikt worden. Allereerst kan er gekeken worden naar kwantitatieve metrieken, zoals het aantal componenten, het aantal afhankelijkheden per component en de grootte van elke component. Ook kan er gekeken worden naar hoe de grootte van de componenten zich onderling verhouden (is er een component die vele malen groter is dan alle andere componenten?), of hoe de relaties tussen de componenten zijn (waarom is de ene component afhankelijk van alle andere componenten?).

Het is mogelijk om deze metrieken automatisch uit te rekenen, wat ervoor zorgt dat ze na elke wijziging aan het systeem geïnspecteerd kunnen worden. Hierdoor wordt het mogelijk om continu de belangrijke aspecten van de geïmplementeerde architectuur in de gaten te houden. Het toevoegen van bijvoorbeeld een extra component of een nieuwe afhankelijk-

heid kan meteen worden opgemerkt, waardoor sneller actie kan worden ondernomen.

Daarnaast levert het continu uitrekenen van deze metrieken ook de mogelijkheid op om trends op te sporen die anders vrijwel onmogelijk op te merken zijn. Zo kan het gebeuren dat een component over een bepaalde periode alleen maar groeit, terwijl alle andere componenten kleiner worden. In de meeste gevallen is het niet gewenst dat een component alle functionaliteit naar zich toe trekt, in deze situatie zal er dus actie ondernomen moeten worden. Zonder het continu meten van de grootte van de componenten zijn dit soort observaties niet te maken, waardoor dit ongewenste gedrag (te) laat wordt opgemerkt.

Al met al bieden metrieken op architectuurniveau de mogelijkheid om de belangrijkste aspecten van een architectuur continu in de gaten te houden. Dit betekent overigens niet dat de terugkerende architectuurevaluaties hierdoor overbodig worden. Sterker nog, een verandering in de metrieken is een uitgelezen indicator dat een architectuurevaluatie wenselijk is. Door tijdens deze evaluaties gebruik te maken van een uitgebreidere architectuurevaluatie-methodiek, worden ook de architectuureigenschappen die niet geautomatiseerd door de metrieken gemeten worden, kritisch bekeken.

6.5 Conclusie

Om ervoor te zorgen dat het ontwerp en de implementatie van een architectuur niet uiteen gaan lopen, moeten de architecten en het ontwikkelteam samenwerken. Het implementeren van terugkerende evaluaties van de architectuur binnen het ontwikkelproces, zorgen voor een vast overlegmoment tussen beide partijen, waarbij de sterke en zwakke punten van de architectuur besproken worden.

Om echter te voorkomen dat problemen pas in een laat stadium worden ontdekt, is het mogelijk om de belangrijkste aspecten van de softwarearchitectuur dagelijks met behulp van softwaremetrieken te meten. Hierdoor kunnen structurele problemen zoals de toevoeging van extra componenten of de introductie van een (wellicht ongewenste) afhankelijkheid snel worden opgemerkt. In beide gevallen kan er gelijk actie worden ondernomen om ervoor te zorgen dat ontwerp en implementatie synchroon

evolueren. Bijvoorbeeld door de nieuwe componenten in het ontwerp op te nemen, of door de afhankelijkheid te verwijderen voordat deze zich te veel verankerd heeft in de geïmplementeerde architectuur.

Over de auteur: Eric Bouwers is bij de Software Improvement Group (SIG) medeverantwoordelijk voor het onderhouden en uitbreiden van de analyse en rapportage tooling van de SIG. Verder is hij als technisch consultant betrokken bij het analyseren en reviewen van de systemen van verschillende klanten. Daarnaast houdt hij zich voor zijn promotieonderzoek bezig met onderzoek naar de kwaliteit van softwaresystemen, met een focus op de rol van de softwarearchitectuur.