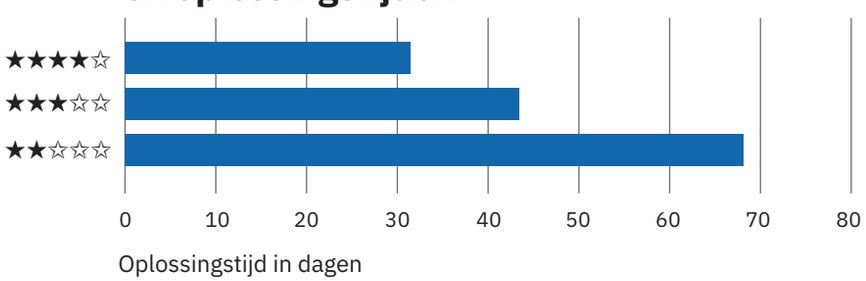
Zo meet en verbeter je de kwaliteit van softwarearchitectuur

Architectuurverbetering schreeuwt om aandacht

De architectuurkwaliteit hangt duidelijk samen met ontwikkelsnelheid en onderhoudbaarheid van de broncode. Ook is architectuurkwaliteit van belang voor strategische keuzes in het ontwikkeltraject. Lodewijk Bergmans en Dennis Bijlsma geven aan wat de belangrijkste aandachtspunten zijn voor het bereiken en behouden van een goede architectuurkwaliteit.

SOFTWARE ENGINEERING IS EEN SOCIO-TECHNISCHE ACTIVITEIT. We vergeten het soms, maar software wordt gebouwd door teams van mensen, die gezamenlijk aan een product werken. Een goede softwarearchitectuur houdt dus rekening met zowel technische als sociale aspecten. De Software Improvement Group (SIG) heeft een architectuurkwaliteitsmodel ontwikkeld om de kwaliteit van socio-technische softwarearchitectuur te kwantificeren. Het model meet meerdere architectuuraspecten, die zowel technische als sociale aspecten bestrijken, en evalueert de resultaten ten opzichte van andere systemen in de SIG-benchmark. Het blijkt dat systemen die beter scoren in dit model ook een kortere time-to-market van aanpassingen laten zien.

De relatie tussen architectuurkwaliteit en oplossingstijden



Dit diagram met de resultaten toont een verband tussen de kwaliteit van de architectuur en de oplossingstijden: het is gemiddeld 30% sneller om veranderingen door te voeren in een 4-sterrensysteem dan in een marktgemiddeld systeem. Omgekeerd is het doorvoeren van wijzigingen in een 2-sterrensysteem ongeveer 40% langzamer dan systemen met een marktgemiddelde architectuurkwaliteit.

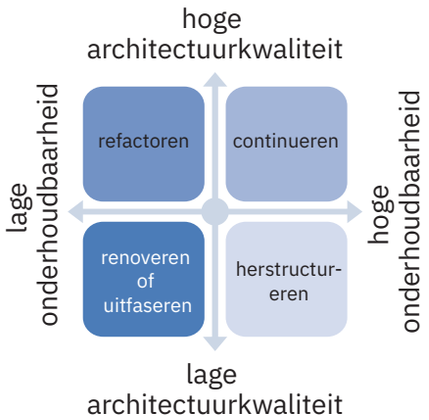
Kortere oplossingstijden

Naarmate de hoeveelheid software in organisaties blijft groeien, moeten organisaties blijven innoveren om aan de steeds hogere verwachtingen van hun klanten te voldoen. Eén innovatietrend is de beweging naar microservice-architecturen. Dit architectuurpatroon vermijdt grote, monolithische toepassingen ten gunste van veel kleine, onafhankelijke componenten. Deze componenten, microservices genoemd, richten zich elk op één specifieke verantwoordelijkheid. De benchmark, met meer dan 10.000 commerciële systemen, laat zien dat microservice-architecturen mainstream zijn (gebleven) vanaf 2017, en dit heeft geleid tot een aanzienlijke toename van het gemiddelde aantal componenten per systeem.

Microservices zijn in de praktijk

Key finding: in systemen met 4-sterrenarchitectuurkwaliteit worden problemen twee keer sneller opgelost dan in 2-sterrensystemen.

Beeld: Marc Kolle



echter niet geheel onafhankelijk; ze zijn nog steeds afhankelijk van elkaar via API's, interfaces, middleware of databases. Dit betekent dat bij het ontwerpen van een systeemarchitectuur zorgvuldig rekening moet worden gehouden met mensen, codestructuur, interfaces en implementatie, ook in relatie tot andere microservices.

Het nieuwe architectuurkwaliteitsmodel meet de belangrijkste aspecten van deze uitdagingen en afwegingen. En dus moet een goede architectuurkwaliteit ontwikkelteams in staat te stellen om zelfstandig en efficiënter te werken. Om die voordelen te kwantificeren hebben we voor vijftig systemen uit de benchmark, met een gemiddelde omvang van veertig persoonsjaren aan code, de architectuurkwaliteit (zoals gemeten door het model) vergeleken met de gemiddelde oplossingstijd (dat wil zeggen de time-to-market voor aanpassingen en nieuwe functionaliteit).

Impact op IT-strategie

Historisch gezien is de meeste aandacht voor oudere systemen gericht op de functionele, operationele en technische uitdagingen. De sociaal-technische architectuur is echter een steeds belangrijker aandachtsgebied: het wordt bijvoorbeeld genoemd als de belangrijkste trend voor enterprise-architecten in een recent rapport van Ardoq [1].

SIG beschouwt onderhoudbaarheid als de basis voor het waarborgen van wendbaarheid en flexibiliteit van een codebase. Dus hoe verhoudt de SIG-onderhoudbaarheidsbeoordeling zich tot de sociaal-technische beoordelingen van softwarearchitectuur?

Wanneer je voor alle systemen in de benchmark zowel de onderhoudbaarheids- als architectuurbeoordelingen bekijkt, kan je de systemen verdelen in vier kwadranten, waarbij elk kwadrant een combinatie van hoge/lage onderhoudbaarheid en architectuurkwaliteit representeert. Het blijkt dan dat onderhoudbaarheid en sociaal-technische architectuur grotendeels onafhankelijke uitdagingen zijn: er zijn veel systemen met een slechte onderhoudbaarheid maar een acceptabele sociaal-technische architectuur, maar er zijn ook veel goed onderhoudbare systemen met een slechte architectuur.

Elk van de kwadranten kan worden gekarakteriseerd door de meest waarschijnlijke actie die moet worden overwogen:

- **Renoveren of uitfaseren:** systemen met zowel een lage onderhoudbaarheid als een lage architectuurkwaliteit

zijn kandidaten voor serieuze kwaliteitsverbetering, of om volledig te verwijderen of vervangen.

- **Refactoren:** voor systemen met een goede architectuurkwaliteit, maar een lage onderhoudbaarheid is het herstructureren van de code om de onderhoudbaarheid te verbeteren een zinvolle optie.
- **Herstructureren:** dit kwadrant bevat de systemen die een hoge onderhoudbaarheid hebben, maar lijden onder een lage architectuurkwaliteit. Vooral voor systemen waarvan wordt verwacht dat ze in de toekomst substantieel zullen evolueren, kan het een zinvolle investering zijn om de structuur en organisatie van de software te herzien.
- **Continueren:** systemen met zowel een goede architectuur als een hoge onderhoudbaarheid bevinden zich in een goede situatie voor verdere verbetering en evolutie.

Het blijkt dat over het algemeen de systemen die goed te onderhouden zijn, kleiner in omvang zijn. Voor de architectuurkwaliteit doet de omvang van de systemen er veel minder toe: er zijn veel middelgrote tot grote systemen met een hoge architectuurkwaliteit, al zien we dat de grootste systemen in de categorie ‘renoveren of uitfaseren’ vallen.

Afhankelijkheden en kennis

Socio-technische architectuur heeft dus impact op het moderniseren van bestaande systemen. Maar welke factoren hebben de grootste invloed op de architectuurkwaliteit? Het architectuurkwaliteitsmodel is opgebouwd uit tien deelscores, die elk verschillende architectuuraspecten meten. Onderstaand diagram toont de top vijf architectuuraspecten waarvoor high performers beter scoren dan het gemiddelde systeem in de benchmark.

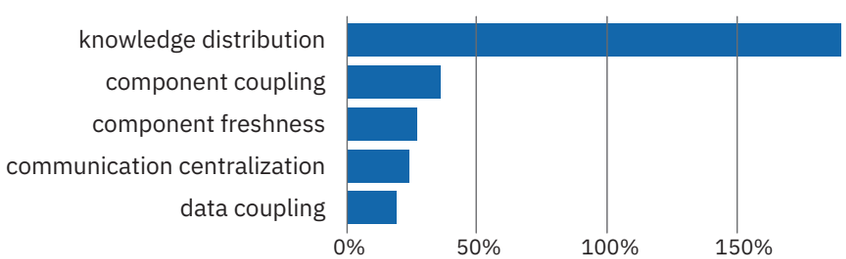
Key finding: Uit de architectuurbenchmark blijkt dat grotere systemen vaak in kwaliteit achteruitgaan, maar niet altijd. Het is dus inderdaad mogelijk om grote systemen te bouwen met een hoge architectuurkwaliteit.

Key finding: Knowledge Distribution en Component Coupling zijn de belangrijkste aspecten voor hoge architectuurkwaliteit in de architectuurkwaliteitsbenchmark: focus vooral op deze factoren om de architectuurkwaliteit te verhogen.

Het aspect Knowledge Distribution geeft de mate aan waarin ontwikkelaars effectief parallel kunnen werken, met een gezamenlijke kennis die de gehele codebasis bestrijkt: het meet in welke mate er een evenwichtige verdeling is van de ontwikkelingsactiviteit over alle softwarecomponenten. Dit betekent dat er voor alle componenten in een systeem actieve ontwikkelaars zijn en dat ontwikkelaars niet te veel aan dezelfde delen van de code werken, wat een bekende bron van inefficiëntie en bugs is.

Het aspect Component Coupling gaat over technische verwevenheid: hoe meer afhankelijkheden tussen componenten, hoe moeilijker het wordt om deze componenten aan te passen zonder gevolgen voor andere componenten en ontwikkelaars.

Welke aspecten onderscheiden de top performers van architectuurkwaliteit



Incrementele aanpak

In dit artikel hebben we vooral besproken waarom socio-technische architectuur belangrijk is, en welke aspecten daarvan het meest van belang zijn. De logische vervolgvraag is vervolgens: hoe kan je de socio-technische architectuur verbeteren?

SIG helpt organisaties met het verbeteren van hun software, en op basis van deze ervaring hebben we een aantal best practices voor onze klanten opgesteld die hen helpen bij het moderniseren en verbeteren van hun architectuur:

1. Definieer architectuurprincipes, geen regels.

Een evoluerende architectuur betekent ook dat het niet mogelijk is om de architectuur vooraf volledig vast te leggen. Dit is ook niet wenselijk, want dit verkleint de ruimte voor autonomie en voortschrijdend inzicht. Onafhankelijke systemen en componenten faciliteren de teams die eraan werken om zelf architectuurbeslissingen te nemen, zonder dat elke beslissing moet worden goedgekeurd door een centraal beslisorgaan. Teams moeten in staat zijn hun eigen architectuur te sturen, zolang hun keuzes in lijn blijven met de architectuurprincipes van de organisatie. Deze architectuurprincipes moeten teams helpen bij het nemen van beslissingen, zonder elk aspect te dicteren of op microniveau te beheren. Uiteraard is er nog steeds een feedbackloop nodig om er voor te zorgen dat de principes

daadwerkelijk in de praktijk worden toegepast. Het gebruik van metrieken, zoals in dit artikel toegelicht, kan helpen als objectieve input voor deze feedbackloop.

2. Leg de overwegingen voor architectuurbeslissingen vast.
Software-architectuur wordt gevormd door een reeks ontwerpbeslissingen. Voor mensen die niet bij de oorspronkelijke beslissing betrokken waren, is het vaak niet meer duidelijk wat nu precies de onderliggende afwegingen zijn geweest. De code en de architectuur zelf kunnen alleen de huidige staat communiceren, maar niet hoe en waarom die huidige staat tot stand is gekomen.

Architecture Decision Records (ADR's) kunnen helpen om deze architectuurbeslissingen op een gestructureerde en efficiënte manier vast te leggen. ADRs zijn een nuttig communicatiemiddel die een geschiedenis vormen van de in het verleden gemaakte afwegingen, welke ook van belang kunnen zijn bij nieuwe beslissingen of het herzien van eerdere beslissingen op basis van nieuwe omstandigheden: architectuurbeslissingen blijven dus niet altijd voor eeuwig gelden.

Te veel ontwikkelaars op hetzelfde stuk code vergroot het risico op inefficiëntie en bugs

3. Pak architectuur op een incrementele manier aan.
Het lijkt vaak onhaalbaar om de architectuur stapsgewijs te veranderen. En inderdaad, het is niet realistisch om duizenden ongewenste afhankelijkheden tussen twee systemen in één sprint of iteratie op te lossen. Maar je kunt het uiteindelijke doel ook opdelen in kleinere delen. Bijvoorbeeld: onderzoek eerst de afhankelijkheden tussen subcomponenten, en streef er vervolgens naar om die binnen een sprint te veranderen. Het incrementeel verbeteren en moderniseren van de architectuur beperkt de risico's. Kleine, incrementele veranderingen hebben een kleinere reikwijdte en leiden daarom tot minder stabiliteitsrisico's. Bovendien wordt een situatie vermeden waarin de modernisering van de architectuur rechtstreeks concurreert met functionele veranderingen.

Om ervoor te zorgen dat architectuurverbeteringen een aandachtspunt blijven, moeten ze ook worden opgenomen in de 'definition of done' voor elke sprint. Hiermee wordt architectuur een onderdeel van het normale ontwikkelproces, in plaats van een parallelle activiteit. Dit helpt ook bij het definiëren van expliciete tussendoelen, en op die manier kan langzaam worden toegewerkt naar het uiteindelijke langetermijndoel.

Referentie

[1] https://content.ardoq.com/enterprise-architecture-trends-infographic

Dennis Bijlsma en Lodewijk Bergmans
zijn researchers bij de Software Improvement Group (SIG), een onafhankelijke organisatie gespecialiseerd in het meten en verbeteren van de kwaliteit, veiligheid, kosten en risico's van softwaresystemen.

Reacties en bijdragen

Voor reacties en nieuwe bijdragen van IT-experts:
Tanja de Vrede
020-2467230
t.d.vrede@agconnect.nl