Software Architecture Discovery for Testability, Performance, and Maintainability of Industrial Systems

Ganesan, D.

2012

document version
Publisher's PDF, also known as Version of record

Samenvatting

In dit proefschrift hebben we, door de praktijk geïnspireerd, een aanpak voor software architectuur analyse uitgevoerd op industriële systemen. De aanpak volgt de filosofie van "industry-as-laboratory". Door het toepassen van de beginselen van "Action Research" konden we in samenwerking met de industrie, echte problemen identificeren en de oplossingen valideren. In dit proefschrift hebben we aangetoond dat deze aanpak heeft geleid tot tools en design leermomenten die de kwaliteit van deze software producten heeft verbeterd.

Het ontwerpen van software architecturen is een uitdagende activiteit. Architecten hebben behoefte aan referentie-materiaal van bewezen *best architecture practices*, zodat ze hun eigen systemen kunnen toetsen dat ze voldoen aan kwaliteitseisen, zoals onderhoudbaarheid, testbaarheid en runtime-performance. Ons belangrijkste doel was om te komen tot een aantal praktische en relevante architectuur feiten, door systematisch, uit bestaande industriële systemen, een reeks aanbevelingen af te leiden. Om bestaande real-world systemen efficiënt en effectief te analyseren, volgden we een benadering waarin de software architectuur centraal stond. Dat betekende dat we de geïmplementeerde software architectuur geanalyseerd hebben op zijn testbaarheid, performance en onderhoudbaarheid. Zo ontwikkelden we de Architectuur Discovery and Analysis Method (ADAM).

1. In dit proefschrift hebben we een analyse methode voorgesteld om structuur en gedrag vanuit de implementatie te herkennen en vast te leggen. Door het maken van een goede vocabulaire en beperkingen (constraints) op types van componenten en connectoren te leggen, konden we regels voor architectuur stijlen afleiden. We hebben deze stijlen geformaliseerd met behulp van gekleurde Petri netten. Tijdens de werking van het systeem hebben we gebeurtenissen (events) verzameld om, met behulp van deze Petrinetten, component-connector views, sequence diagrammen en verschillende architectuur stijlen te herleiden. Dit staat beschreven in hoofdstuk 2, hoofdstuk 4 en hoofdstuk 5.
2. Wij hebben een methode voorgesteld om onderhoudbaarheid te bepalen door unit test code te analyseren. We bespraken architectuur beslissingen die unit test code eenvoudiger te maken. Wij hebben een vragenlijst ontwikkeld om de onderhoudbaarheid van unit test code te bepalen. Verder toonden we aan dat deze vragen eenvoudig, edoch effectief, zijn om unit test code te bekijken. Voor meer details wordt de lezer verwezen naar hoofdstuk 3.

3. Wij hebben een methode voorgesteld om de implementatie te analyseren om de kwaliteit aspecten van de software architectuur te beschrijven. In het bijzonder de volgende kwaliteiten, met bijbehorende risico's: performance, testbaarheid (met nadruk op unit testen) en onderhoudbaarheid. De kerngedachte bestaat uit het feit dat architectuur beslissingen van systemen zijn geïnspireerd en zijn beïnvloed door afhankelijkheden met externe entiteiten (zoals Commercial Off The Shelf (COTS) componenten, frameworks en libraries). Door deze afhankelijkheden systematisch te onderzoeken kan de software architectuur en kunnen de kwaliteit risico's die diep zijn verborgen toch worden ontdekt. Onze methode helpt bij het verbeteren van ons begrip voor de relatie tussen software-architecturen en testen. Voor meer details wordt de lezer verwezen naar hoofdstuk 6 en hoofdstuk 7.

4. Wij hebben een methode voorgesteld om uit de source code de runtime structuur van software architectuur te halen. We toonden aan dat deze methode goed is om, op architectuur niveau, te redeneren over de veiligheid van kritische medische software systemen. Voor meer details wordt de lezer verwezen naar hoofdstuk 7.

5. Wij hebben een methode voorgesteld om (bedrijfs-) organisatie aspecten te analyseren die invloed hebben op de software architectuur. Onze methode ondersteunt het verbeteren van de relaties tussen software-architecturen en organisatorische aspecten. Voor meer details wordt de lezer verwezen naar hoofdstuk 8.

6. We hebben algemeen toepasbare aanbevelingen gegeven om de kwaliteit tijdens design te verbeteren in plaats van kwaliteit te meten tijdens test. Gebaseerd op architectuur analyse van de diverse praktijk systemen, hebben we een reeks van algemeen geldende aanbevelingen ontdekt die de kwaliteit reeds tijdens ontwerp realiseert, in plaats van achteraan in het proces tijdens het testen. Onze aanbevelingen zijn te karakteriseren als a) hoe architectuur overtredingen te voorkomen in de eerste plaats, b) hoe het testen te vergemakkelijken door het testen expliciet te ontwerpen en c) hoe de performance risico's te verminderen door gebruik te maken performance-georiënteerde design patterns. Voor meer details wordt de lezer verwezen naar hoofdstuk 9.

7. We hebben een suite van reverse engineering tools ontwikkeld. Deze tools dragen bij aan a) extractie van gegevens tijdens de werking van het systeem, hetzij statisch uit de source code of dynamisch tijdens runtime, en b) abstractie en visualisatie van de verzamelde gegevens. Voor meer details wordt de lezer verwezen naar hoofdstuk 10.

8. In Hoofdstuk 11 hebben we opnieuw naar onze onderzoeksvragen gekeken vanuit het perspectief van gedane onderzoek. De nog openstaande kwesties van belang voor toekomstig onderzoek zijn hierin ook beschreven.

Een goede schilder weet wanneer hij moet stoppen met schilderen.