Supporting Architecture Evolution by Mining Software Repositories

Vanya, A.

2012

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

Samenvatting

Software is een onvermijdelijk deel van ons leven geworden. Bijna alles wat wij willen bereiken wordt tegenwoordig ondersteund door software en er zijn weinig doelen die wij zonder softwaregebruik kunnen of willen bereiken. Voor velen begint de dag al met het gebruik van software: de software in onze mobiele telefoons wekt ons. Wij gebruiken software in onze televisie of het gedistribueerd software systeem van het Internet om bijvoorbeeld informatie over het weer op te zoeken. De software van autonavigatiesystemen brengt ons naar onze plaats van bestemming. Bewust of onbewust gebruiken wij software wanneer wij reizen, kopen, communiceren of ons vermaken. Ook groeit het softwaregebruik iedere dag en het is niet te verwachten dat software een minder belangrijk deel van ons leven wordt.

De meeste software functioneert in een omgeving die voortdurend verandert. Deze omgeving bestaat uit hardware, en de mensen die van de hardware gebruik maken. Mobiele telefoons, desktop PCs, printers, en televisies zijn voorbeelden van hardware waarop software functioneert. Mensen gebruiken deze hardware met bepaalde verwachtingen en deze verwachtingen veranderen met de tijd. Ieder jaar verwachten wij bijvoorbeeld steeds meer van een mobiele telefoon. Wij verwachten nu dat we een mobiele telefoon met geluid of aanraking kunnen besturen. Om dit te kunnen bereiken zijn nieuwe hardwarecomponenten nodig. Hiermee verandert de omgeving van de software.

Het is van belang te zorgen dat software gemakkelijk en goedkoop gewijzigd kan worden. Veranderingen in de omgeving van de software leiden tot veranderingen in de software. Als wij de software niet blijven veranderen dan gaat de kwaliteit van die software omlaag. Omdat softwareveranderingen vaak optreden is het ook van belang dat zij snel en zo goedkoop mogelijk plaatsvinden. Deze architecturele eigenschap of kwaliteit van software noemen wij software evolvabiliteit. Software kwaliteit moet onderhouden worden. Dit geldt in het bijzonder ook voor software evolvabilitiet. Hoe beter de evolvabilitiet van software is, hoe beter software aangepast kan worden. Het aanbrengen van veel veranderingen zorgt voor ontwerperosie met als gevolg, slechtere evolvabilitiet van de software. Daarom is het van cruciaal belang dat software evolvabiliteit continu wordt onderhouden.

Dit proefschrift beschrijft hoe software-architecten ondersteund kunnen worden om de evolvabiliteit van hun software-architectuur te analyseren, verbeteren en onderhouden. Om dit doel te bereiken, werd historische data uit versiebeheerssystemen gebruikt. Dit proefschrift kijkt naar het verleden en neemt aan dat als een bepaald deel van de software vaak veranderd is, dit ook in de nabije toekomst opnieuw zal veranderen. Dit staat bekend als de 'Yesterday's Weather' benadering. Omdat versiebeheerssystemen een bepaald type softwarebewaarplaatsen zijn, draagt dit proefschrift bij aan het gebied dat bekend staat als mining van softwarebewaarplaatsen.

Het minen van softwarebewaarplaatsen kan op verschillende manieren gebeuren. In dit proefschrift wordt historische metadata verzameld om architecten te helpen ongewenste koppelingen tussen componenten van het software systeem te identificeren en te corrigeren. Om dit te bereiken analyseren we veranderingen tussen verschillende versies van het software sys-

teem. Dit proefschrift gebruikt het minen van softwarebewaarplaatsen deels op verzoek van de ondersteunde software-architect en deels op basis van karakteristieke eigenschappen van het geanalyseerde software systeem.

Dit onderzoek is uitgevoerd als onderdeel van het DARWIN project, onder begeleiding van het Embedded Systems Institute in Eindhoven. Het project liep van september 2005 tot september 2010. Vijf Nederlandse universiteiten werkten samen met de industriële partner, Philips Healthcare MRI, Best, om de uitdagingen van systeem evolvabiliteit te bestuderen. De doelstelling van het project was om hulpmiddelen, methoden en architecturen te ontwikkelen waarmee een geoptimaliseerde systeem evolvabiliteit bereikt kan worden. Het onderzoek betrof zowel hardware als software evolvabiliteit. Sommige onderzoekers werkten aan de synergie tussen beiden. Het DARWIN onderzoek leidde niet alleen tot publicaties maar ook tot concrete veranderingen in het ontwerp- en ontwikkelproces van het bestudeerde systeem.

Het onderzoek, op basis waarvan dit proefschrift is samengesteld, analyseerde het software systeem van Philips Healthcare MRI, Best. Dat systeem bevat ongeveer 34 000 bestanden en meer dan negen miljoen regels code. Honderden ontwikkelaars op drie ontwikkelingsplaatsen werken aan het systeem. Deze ontwikkelingsplaatsen bevinden zich op drie verschillende continenten. De complexiteit van het systeem groeide enorm in de laatste twee decennia en het is momenteel een uitdaging om het systeem verder te ontwikkelen. Bij de ontwikkeling zijn vooral de programmeertalen C#, C++ en C gebruikt. Nieuwe versies van bestanden zijn in een ClearCase versiebeheerssysteem ingecheckt. De gebruikte versie van ClearCase bewaart ook metadata, zoals wie een bestand heeft ingecheckt, wanneer en (heel soms) waarom. Echter, er is geen informatie opgeslagen over welke veranderingen bij welke ontwikkeltaken horen.

Het onderzochte systeem maakt medische diagnoses mogelijk door het maken van medische afbeeldingen die inwendige delen van het menselijke lichaam tonen. Deze afbeeldingen zijn belangrijk bijvoorbeeld bij, kankerbestrijding. Afbeeldingen worden verworven middels een voordurend veranderend magnetische veld wat is geproduceerd met behulp van één of meer magneten.

Het opdelen van een software systeem in delen is een veelgebruikte manier om complexiteit te beheren en software evolvabiliteit te verbeteren. Vaak hebben software systemen meer dan één opdeling. Onderdelen van het systeem horen bij elkaar als zij bijvoorbeeld zijn ontwikkeld door dezelfde ontwikkelingsgroep of als zij de functionaliteit van hetzelfde stuk hardware implementeren. Echter, een willekeurig opdeling van een software systeem garandeert niet dat de evolvabiliteit van dat systeem beter wordt. Een goede opdeling, welke software evolvabiliteit ondersteunt, moet bijvoorbeeld uit delen bestaan die onafhankelijk van elkaar veranderd kunnen worden. Met een dergelijke opdeling is het mogelijk om het aantal te veranderen software entiteiten beperkt te houden, de tijd die nodig is voor testen te reduceren, enzovoort.

Het is mogelijk dat een opdeling niet helemaal voldoet aan de eis van onafhankelijke evolutie. Dergelijke situaties kunnen tot stand komen ook als de opdeling initieel goed doordacht was maar gedegradeerd is door veel veranderingen aan het systeem. Als software entiteiten uit een deel van het software systeem vaak veranderden samen met entiteiten van een ander deel dan is er sprake van een mogelijke ongewenste koppeling. Dergelijke ongewenste koppelingen laten zien waar de opdeling niet aan de eisen van onafhankelijke evolutie vol-

doet. Het is de taak van software-architecten om de ongewenste koppelingen te identificeren en weg te werken. Echter, in een groot software systeem zijn er vele potentieel ongewenste koppelingen waarmee de software-architect om moet gaan. Omdat de beschikbare tijd van software-architecten beperkt is, kunnen zij alleen de meest ernstige ongewenste koppelingen elimineren.

In dit proefschrift wordt een methode beschreven ter ondersteuning van softwarearchitecten om ongewenste koppelingen te identificeren, te analyseren en op te lossen. De hoofdonderzoeksvraag is: "Hoe kunnen software-architecten geholpen worden om de opdeling van een software systeem te verbeteren?" Omdat deze onderzoeksvraag nogal abstract is, zijn er drie gerelateerde concrete onderzoeksvragen geïdentificeerd en aangepakt. Dit zijn: "Hoe kunnen wij groepen van software entiteiten identificeren die vaak samen veranderden?" (OV-1), "Hoe kunnen we groepen van software entiteiten selecteren die ongewenste koppelingen aanduiden? " (OV-2) en "Hoe kan interactieve visualisatie helpen om potentieel ongewenste koppelingen te analyseren? " (OV-3). Om de deelonderzoeksvragen (OV-1, OV-2 en OV-3) te beantwoorden zijn drie onderzoekcycli uitgevoerd.

Als wij willen weten (OV-1) welke software-entiteiten vaak samen veranderden, moeten wij eerst nagaan welke entiteiten samen veranderden, de zgn changesets. In veel omgevingen is deze informatie niet beschikbaar. In de door ons bestudeerde omgeving is bovendien de metadata van individuele veranderingen nogal incompleet. Daarom moet men in zulke omgevingen alternatieve manieren bedenken voor het benaderen van changesets. De benaderde changesets kunnen dan gebruikt worden om entiteiten welke samen veranderen te identificeren.

Dit proefschrift beschrijft vijf varianten van het bekende "sliding window" algoritme om changesets uit de historische metadata van Philips Healthcare MRI te benaderen. Het gebruikte ontwikkelproces had een grote invloed op de benaderingsalternatieven die zijn geïdentificeerd. De benadering die in dit proefschrift is beschreven is gebaseerd op de nauwkeurigheid van de benaderde changesets. De nauwkeurigheid van benaderde changesets heeft weer invloed op de beslissingen van de software-architect. Aangezien deze beslissingen zorgvuldig moeten worden genomen, is het belangrijk om de nauwkeurigheid van de benaderde changesets te evalueren.

Wij gebruikten twee manieren om de nauwkeurigheid van de benaderde changesets te evalueren. De ene maakte gebruik van een online enquête die door ontwikkelaars ingevuld moest worden. De andere manier was gebaseerd op zorgvuldig geselecteerde postlisten, d.w.z. lijsten van wijzigingen voorgelegd aan de systeemintegrator. De resultaten van beide alternatieven waren vergelijkbaar. Gebaseerd op deze resultaten was het mogelijk een benaderingsalternatief te selecteren waar de nauwkeurigheid- en precisiewaarden optimaal waren gecombineerd (respectievelijk 89% en 84%).

Na de nauwkeurige benadering van changesets, is de volgende stap het groeperen van software entiteiten die vaak samen veranderden. Hiervoor is in dit proefschrift een clusteringmethode gebruikt. Het resultaat van deze clustering is een binaire geparametriseerde boom, of dendrogram, waar de knooppunten groepen van software entiteiten aanduiden die samen veranderden.

Om onderzoeksvraag OV-2 te beantwoorden, stelden wij in de tweede cyclus van het on-

derzoek een methode voor. Deze methode is in Philips Healthcare MRI en op een open source software systeem (ArgoUML) toegepast. De methode karakteriseert eerst evolutionaire clusters gebaseerd op bepaalde eigenschappen. Mogelijke eigenschappen zijn de grootte van een cluster, of de metriek welke aangeeft of de veranderingen vooral recentelijk zijn opgetreden. Deze eigenschappen zijn gebruikt om, in overleg met de software-architecten, te zoeken naar die evolutionaire clusters welke op ongewenste koppelingen duiden. Het resultaat hiervan zijn die potentiële ongewenste koppelingen welke verder geanalyseerd moeten worden.

In deze methode is het van essentieel belang dat het mogelijk is om gebruik te maken van de input van de software-architecten over ongewenste koppelingen. Welke koppelingen echt ongewenst zijn, is subjectief. Iedere architect moet daarom zelf kunnen aangeven welke koppelingen zij ongewenst vinden. Uit de resultaten van de methode blijkt dat de geselecteerde ongewenste koppelingen inderdaad naar ongewenste koppelingen wezen. De mogelijkheid om evolutionaire clusters te filteren is cruciaal ook omdat er veel evolutionaire clusters zijn. Echter, software-architecten zijn slechts in enkele van deze clusters geïnteresseerd. De behoefte om de bijbehorende koppelingen te kunnen analyseren leidt ons tot het probleem van de derde onderzoekcyclus, zie ook OV-3.

Interactieve visualisaties maken het voor de gebruiker mogelijk wat er wordt gevisualiseerd en hoe dit gebeurt, tijdens de visualisatie aan te passen. Door interactie met visualisaties krijgt de gebruiker zijn informatie in iteraties. In iedere iteratie wordt informatie getoond en de gebruiker interacteert met de visualisatie om nog meer informatie krijgen. De interacties worden gestuurd door de interesse van de gebruiker en deze interesses worden beïnvloed door de eerder verzamelde informatie.

In dit proefschrift beschrijven we hoe zo'n interactief exploratieproces architecten en software ontwikkelaars kan helpen om ongewenste koppelingen te analyseren en op te lossen. Wij stellen ook een aantal mogelijke typen interactie voor. Deze realiseerden wij in een interactief visualisatieprogramma (iVis). Wij gebruikten dit programma om de geselecteerde evolutionaire clusters te visualiseren. Om deze clusters te analyseren werd een aantal sessies met de architect en ontwikkelaars georganiseerd. Tijdens deze sessies zijn de voordelen van interactieve visualisaties gebleken. Hiervan kwam naar voren dat interactieve visualisaties niet alleen kunnen helpen om over ongewenste koppelingen te redeneren, maar dat zij ook kunnen helpen om oplossingsalternatieven te vinden en om hun impact te bepalen.

Uit het proces om de architect te ondersteunen destilleerden wij een aantal lessen. Deze lessen kunnen in drie categorieën worden verdeeld:

- maak gebruik van frequente terugkoppelingen naar de software-architect en ontwikkelaars
- zorg voor nauwkeurige informatie over het gevolgde ontwikkelproces
- bouw een goede kennis op over gebruikte data.