Samenvatting

Software speelt een sleutelrol in ons leven. Software is niet alleen verantwoordelijk voor het functioneren van zowel onze digitale als onze fysieke infrastructuur maar is zelfs van groot belang in onze dagelijkse inter-persoonlijke communicatie: We interacteren met organisaties en onze overheden met behulp van software en we vertrouwen niet alleen op de software in onze wekker en digitale videorecorder maar ook op de software in de MRI scanner in het ziekenhuis of in de treinbeveiliging.

Deze ontwikkeling leidt tot een vraag naar steeds grotere en complexere software-systemen enerzijds en toenemende kwaliteitseisen anderzijds. Om niet-functionele systeemeisen (als veiligheid, onderhoudbaarheid en duurzaamheid) te kunnen garanderen is het zaak om de interactie van systeemcomponenten op een hoger niveau van abstractie te ontwerpen. Dit hogere niveau van abstractie wordt software-architectuur genoemd — in lijn met de bouwkundige analogie waar "software engineering", een gangbare aanduiding voor softwareontwikkeling, aan refereert. Methoden, technieken en processen die samenhangen met software-architectuur stellen softwareontwikkelaars in staat om te gaan met toenemende systeemcomplexiteit en stengere kwaliteitseisen. In het wetenschappelijke onderzoek heeft software-architectuur de afgelopen 15 jaar daarom veel aandacht gekregen.

De voordelen van het werken met abstracties en de directere link die daardoor gelegd kan worden met het specifieke domein waarbinnen een bepaald software systeem ontwikkeld wordt, maakt dat modellen een steeds centrale rol spelen binnen softwareontwikkeling. Een veel toegepast concept waarbinnen modellen centraal staan is modelgedreven softwareontwikkeling (MDD). In dit ontwikkelparadigma staan modellen, vaak ontwikkeld in domeinspecifieke talen, centraal. Afhankelijk van de ondersteunende software en specifieke methoden die worden toegepast kan executeerbare code worden gegenereerd uit abstracte modellen of worden deze modellen zelf "uitgevoerd" als waren ze software. De praktische relatie tussen software-architectuur van software-systemen die in meer of mindere mate van de grond af aan ontwikkeld worden (ook wel custom of "greenfield" softwareontwikkeling) en modelgedreven

ontwikkeling is onduidelijk.

Offshoring (ook wel outsourcing of gedistribueerde softwareontwikkeling) is weer een andere, meer recente ontwikkeling die van grote invloed is op de manier waarop software wordt ontwikkeld. Bij deze vorm van softwareontwikkeling wordt software gebouwd op meerdere, geografisch gescheiden locaties tegelijk. Aanvankelijk werden kostenbesparingen verwacht van de lagere lonen van softwareontwikkelaars in landen als Canada en India. Later leek offshoring ook een oplossing voor het tekort aan softwareontwikkelaars door de steeds grotere vraag naar software-systemen.

In een gedistribueerd softwareontwikkelproject vinden de verschillende ontwikkelfasen typisch plaats op verschillende, door significante geografische afstand gescheiden locaties. Het verzamelen van systeemeisen en ontwikkelen van de architectuur vind meestal plaats op een andere locatie dan waar het systeem daadwerkelijk wordt geïmplementeerd (geprogrammeerd). Van de implementatie van software architectuur in een niet-gedistribueerde omgeving weten we dat de mate waarin de implementatie van een architectuur voldoet aan het ontwerp, in hoge mate afhankelijk is van de manier waarop en de mate waarin de software-architect softwareontwikkelaars ondersteunt bij hun werk. Hoewel de beoogde software-architectuur vrijwel altijd wordt beschreven in een document (SAD), lijken softwareontwikkelaars verder veel te leren over de beoogde architectuurimplementatie door informele communicatie met andere teamleden die veel weten van de architectuur. Geografische, temporale en socio-culturele afstanden maken formele (geplande) communicatie in gedistribueerde softwareontwikkeling echter lastig. Informele communicatie is daarom nog lastiger te organiseren. Het is onduidelijk of documentatie in deze situaties nu een grotere rol speelt of dat software nu misschien minder goed aan architectuurspecificaties voldoet.

De centrale doelstelling in dit proefschrift is onderzoeken *hoe software-architectuur op een effectieve manier kan worden gerepresenteerd, overgedragen en gecoördineerd in de context van gedistribueerde en model-centrische softwareontwikkeling*. De onderzoeksvragen die we hebben afgeleid uit deze doelstelling zijn alle geënt op empirische onderzoeksmethoden. In de context van softwareontwikkeling wil dit zeggen dat we zoveel mogelijk gebruik maken van gegevens uit de bedrijfspraktijk. De belangrijkste bijdragen van dit proefschrift zijn:

- de vaststelling dat de processen omtrent software-architectuur flink afwijken als niet-gedistribueerde en gedistribueerde softwareontwikkeling worden vergeleken, vooral dat er in een relatief laat stadium nog veel werk wordt verricht aan de architectuur van systemen,

- dat er een sterk, negatief verband lijkt te bestaan tussen de mate waarin de implementatie van software-architectuur voldoet aan zijn ontwerp en de mate waarin een softwareontwikkelproject op tijd wordt opgeleverd,

- dat de kostenbesparingen inherent aan de motivatie om gedistribueerd software te ontwikkelen direct beperkingen lijken op te leveren voor de kwaliteit van de

ontwikkelde architectuur en haar representatie,

- dat het begrip van softwareontwikkelaars van software-architectuur
  - positief beïnvloed kan worden door diagrammen en tekst in software-architectuur-documenten te verbeteren en door softwareontwikkelaars te scholen in het gebruik van UML,
  - gelimiteerd wordt door de socio-culturele, geografische en temporale afstanden die door gedistribueerde softwareontwikkeling worden geïntroduceerd,
  - wordt verslechterd door kostenreducties, specifiek doordat er minder wordt geïnvesteerd in coördinatie van het software-architectuur-proces,
  - minder belangrijk is in modelgedreven softwareontwikkeling maar dat ontwikkelaars desalniettemin moeten leren werken met een nieuw softwareontwikkelparadigma, nieuwe software tools én de domeinspecifieke taal,
- dat software-implementaties beter voldoen aan de voorgeschreven architectuur als modelgedreven softwareontwikkeling wordt toegepast omdat de software-architectuur vast wordt gelegd om generatie van broncode mogelijk te maken.

Een bijkomstige bevinding is dat het gebruik van modellen als centrale ontwikkelartefacten fundamenteel verschilt van het klassieke model waarin programmacode deze rol vervult. De verschillen zelf en de implicaties van deze verschillende softwareontwikkelparadigma's zijn echter nog onduidelijk en nodigen uitdrukkelijk uit tot nader onderzoek.