Onderzoeksvoorstel

Proof of concept: De update automatiseren van Angular versie 16 naar Angular versie 20 in de applicaties van een end-to-end kredietdienstverlener.

Bachelorproef, 2025-2026

Wauters Sander

E-mail: sander.wauters@student.hogent.be

Samenvatting

Updaten naar een nieuwe versie van een software framework kan veel tijd in beslag nemen, zeker als de huidige applicatie meerdere versies achter loopt. Het doel van dit onderzoek is om de update van het web framework Angular van versie 16 naar versie 20 te automatiseren zonder de functionele en niet-functionele vereisten te schenden. Framework updates kunnen de performantie en veiligheid van de applicatie verbeteren, dus is het van belang deze uit te voeren. Deze updates zijn niet altijd even simpel om toe te passen, zeker als deze gepaard gaan met veranderingen aan de broncode van de applicatie. In grotere broncodes neemt het proces van systematisch alle veranderingen aan te brengen veel tijd in beslag. Dit onderzoek begint met een oplijsting te maken van alle nodige aanpassingen tussen Angular versie 16 en versie 20. Vervolgens worden de verschillende manieren om aanpassingen automatisch uit te voeren onderzocht. Op basis hiervan wordt een applicatie uitgewerkt dat het update proces zal automatiseren. Deze applicatie wordt eerst getest in een gecontroleerde omgeving en vervolgens op de een reële applicatie van een bedrijf. De effectiviteit wordt beoordeeld aan de hand van het aantal veranderingen de applicatie correct kan uitvoeren ten opzichte van het totaal aantal uit te voeren aanpassingen. Er wordt verwacht dat 75% van alle nodige veranderingen automatisch uitgevoerd kunnen worden. Het automatisatie proces zal naar verwachting de nodige tijd voor de applicaties te updaten verminderen.

Keuzerichting: Mobile & Enterprise development
Sleutelwoorden: Angular, Static code analysis, automatisatie

Inhoudsopgave

1. Inleiding

Het bedrijf Stater is een end-to-end dienstverlener voor zowel hypothecaire en consumentenkredieten, ze ondersteunen de kredietverstrekker voor de dienstverlening aan consumenten. Binnen het bedrijf zijn er verschillende applicatie dat gebruik maken van het Angular framework. Angular is een open-source front-end framework, gebaseerd op de TypeScript programmeertaal, dat gebruikt wordt voor de ontwikkeling van dynamische web applicaties (Cincovic e.a., 2019). Momenteel is Angular versie 20 (v.20) de meest recente stabiele versie. Binnen Stater maken de applicaties gebruikt van Angular versie 16 (v.16), maar het bedrijf is van plan de applicaties te updaten naar de recentste versie, Angular v.20.

Vaniea en Rashidi (2016) omschrijven software updates als het introduceren van nieuwe functionaliteiten, de performantie van de applicatie verbeteren en verzekeren dat de software compatibel blijft met nieuwe software en hardware. Verder omschrijft deze brond dat het up-to-date houden van software cruciaal voor de cyberveiligheid te garanderen.

Vanwege het grote verschil in versies zal het updaten van alle applicaties veel tijd in beslag nemen. Volgends Kaur en Singh (2015) neemt het onderhoudt van een softwareproject gemiddeld 60% van de kostprijs in beslag. Een manier om de tijd voor software-onderhoud in te korten is daarom best interesant. Hieruit komt de vraag: In welke mate is het mogelijk om een applicatie in Angular v.16 automatisch te updaten naar Angular v.20? Voor deze vraag te beantwoorden worden volgende deelvragen geformuleerd:

- Wat zijn de veranderingen tussen Angular v.16 en Angular v.20?
- Welke van deze veranderingen kunnen automatisch uitgevoerd worden zonder de functionele en niet-functionele vereisten van de applicatie in drang te brengen?

- Wat zijn de manieren om code automatisch aan te passen?
- Welke manier(en) is meest geschikt voor toe te passen in deze context?

Gedurende dit onderzoek zal een applicatie ontwikkeld worden dat een software project in Angular v.16 automatisch update naar Angular v.20. In de rest van dit document wordt naar deze applicatie verwezen als de "updater". De updater doorloopt de broncode van een applicatie en maakt een oplijsting van alle nodige aanpassingen en tracht de aanpassing zelf uit te voeren indien mogelijk. Voor de effectiviteit van de updater te meten, worden de uit te voeren updates opgedeeld in verschillende categorieën en wordt gemeten hoeveel van de nodige updates automatisch uitgevoerd kunnen worden.

In de volgende sectie wordt een kort overzicht gegeven van de huidige stand van zaken binnen het probleem en oplossingsdomein. Hierna volgt een beschrijving van de methodologie waar de werking en evaluatie van de updater in meer detail beschreven worden. En tenslotte worden de verwachte resultaten besproken, waarin een inschatting wordt gegeven naar de bevindingen van het onderzoek.

2. Literatuurstudie

2.1. Veranderingen in Angular

De "Angular update guide" (2025) geeft ons een uitgebreid overzicht van alle aanpassingen die nodig zijn voor een Angular applicatie van v.16 naar v.20 te updaten. Uit deze bron blijkt dat in totaal er 79 verschillende stappen uitgevoerd worden.

De studie door Bavota e.a. (2012) onderzoek welke veranderingen aan code het meeste kans hebben om nieuwe bugs te introduceren. In deze studie zijn deze verandering onderverdeeld in 4 categorieën: schadelijk, potentieel schadelijk, niet schadelijk en niet geclassificeerd. Deze studie maakt het mogelijk om een geïnformeerde inschatting te maken naar welke aanpassingen geautomatiseerd kunnen worden zonder onderwachte bijwerkingen te introduceren.

2.2. Automatisatie process

Eén van de meest bekende manieren om code in bulk aan te passen is het gebruik maken van zoek en vervang functies gebaseerd op reguliere expressies (Regex). Een reguliere expressie is een sequentie van karakters dat een patroon in een tekst omschrijven. Aangezien dat Regex text gebaseerd is, kan het geen rekening houden met de semantiek van de programmeertaal. Uit de studie van Michael e.a. (2019) blijken nog enkele problemen bij de implementatie van Regex, namelijk dat het moeilijk leesbaar, vindbaar, valideerbaar en documenteerbaar is.

Om met de semantiek van de programmeertaal rekening te houden kan gebruik gemaakt worden van een abstract syntax tree. Zoals omschreven door Sun e.a. (2023), een abstract syntax tree is een data structuur dat de broncode van een applicatie illustreert en rekening houdt met de syntax en semantiek van de programmeertaal. Dit laat ons toe om een stuk code aan te passen enkel als het in een bepaalde scope zit.

Herinner dat Angular gebaseerd is op de TypeScript programmeertaal. Een bestaande tool voor TypeScript dat gebruik maakt van een abstract syntax tree is de TypeScript Compiler API. De studie door Reid e.a. (2023) onderzoekt hoe de TypeScript compiler gebruikt kan worden voor het corrigeren van foutieve code fragmenten. Deze studie raad aan om de TypeScript compiler te gebruiken voor statische code analyse vanwege de effectiviteit, accuraatheid en mogelijkheid om foutieve code te detecteren.

Een alternatief op de TypeScript compiler dat ook gebruikt maakt van een abstract systax tree is het TypeScript Language Server Protocol (LSP). Het LSP, als omschreven door Bork en Langer (2023), is een open protocol voor gebruik in verschillende code editors of integrated development environments (IDEs) dat programmeertaal specifieke functies voorziet zoals: automatische code aanvullen en code diagnostiek. Dezelfde bron omschrijft LSPs als het de facto standaard protocol voor de implementatie van deze functies in IDEs.

Tenslotte is het mogelijk om AI-tools in te zetten voor deze aanpassingen te maken. Met de recente opkomst van AI-tools dat specifiek gemaakt zijn voor programmeren is het mogelijk om deze taak uit te besteden aan AI. Uit de studie door Hodovychenko en Kurinko (2025) blijkt dat AI gedreven tools een gebrek hebben aan transparantie en risico lopen voor de semantiek van de programmeertaal over tijd fout te interpreteren.

3. Methodologie

3.1. Literatuurstudie

Dit onderzoek start met een uitgebreide literatuurstudie naar de verschillen tussen Angular v.16 en Angular v.20. De nodige verandering worden opgelijst en onderverdeeld in verschillende categorieën. Deze lijst zal gebruikt worden voor de capaciteiten van de updater te bepalen.

Verder wordt onderzocht wat de verschillende manieren zijn om automatisch code te updaten. Eén of meerdere manieren worden verkozen om te implementeren op basis van volgende parameters:

- De complexiteit van de implementatie. Een

voorkeur wordt gegeven aan het gebruik maken van bestaande tools over het ontwikkelen van nieuwe algoritme.

- De betrouwbaarheid van de output. Is het mogelijk om een correct configuratie een incorrecte output te krijgen?

Deze literatuurstudie neemt één tot twee weken in beslag en het resultaat dient als basis voor de volgende fase van het onderzoek.

3.2. Ontwikkeling

In deze fase van het onderzoek zal de updater ontwikkeld worden op basis van de voorafgaande literatuurstudie.

De updater heeft drie functies: het detecteren van aanpassingen in de broncode, het evalueren of deze aanpassingen automatisch kunnen uitgevoerd worden, en de aanpassingen uitvoeren indien mogelijk. Elke gedetecteerde aanpassing wordt geregistreerd, waaronder de locatie in de broncode, de aard van de wijziging, de categorie, en of de updater in staat is om de wijziging automatisch uit te voeren.

De updater wordt in een gecontroleerde omgeving getest om de stabiliteit te verzekeren. Voor het ontwikkelen van de applicatie wordt vier tot vijf weken voorzien.

3.3. Evaluatie

De updater wordt uitgevoerd op één van de applicatie binnen Stater. Vervolgens wordt het resultaat van de updater geëvalueerd aan de hand van het aantal uitgevoerde aanpassingen ten opzichte van het totaal aantal gedetecteerde aanpassingen.

4. Verwacht resultaat, conclusie

Op basis van de literatuurstudie en de gehanteerde methodologie verwacht dit onderzoek dat minstens 75% van alle nodige aanpassingen automatisch uitgevoerd kunnen worden. Het automatisatie proces zal naar verwachting de nodige tijd voor de applicaties te updaten verminderen. De hoeveelheid tijd dat in totaal bespaard zal worden zal afhangen van het aantal applicatie en de grootte van de broncode dat geüpdatet moet worden. Het ontwikkelen van de updater heeft uiteraard ook tijd in beslag genomen. Deze oplossing zal wellicht meer tijd in beslag nemen als enkel één kleine applicatie geüpdatet moet worden.

Of dit de meest effectieve manier is voor deze casus op te lossen is open voor debat. Verder onderzoek zal uitgevoerd worden naar de performantie, accuraatheid en complexiteit van de verschillende implementaties besproken in de literatuurstudie.

Referenties

Angular update guide. (2025). Verkregen september 8, 2025, van https://angular.dev/update-guide?v=16.0-20.0&l=3

Bavota, G., De Carluccio, B., De Lucia, A., Di Penta, M., Oliveto, R., & Strollo, O. (2012). When does a refactoring induce bugs? an empirical study. *2012 IEEE 12th International Working Conference on Source Code Analysis and Manipulation*, 104–113. https://www.inf.usi.ch/faculty/bavota/papers/scam2012.pdf

Bork, D., & Langer, P. (2023). Language server protocol: An introduction to the protocol, its use, and adoption for web modeling tools. *Enterprise Modelling and Information Systems Architectures (EMISAJ)*, *18*, 9–1. https://doi.org/10.18417/emisa.18.9

Cincovic, J., Delcev, S., & Draskovic, D. (2019). Architecture of web applications based on Angular Framework: A Case Study. *methodology*, *7*(7), 254–259. https://www.eventiotic.com/eventiotic/files/Papers/URL/df6b5054-816e-4bee-b983-663fb87be2cd.pdf

Hodovychenko, M. A. H. M. A., & Kurinko, D. D. K. D. D. (2025). Analysis of existing approaches to automated refactoring of object-oriented software systems. *Вісник сучасних інформаційних технологій*, *8*(2), 179–196. https://doi.org/10.15276/hait.08.2025.11

Kaur, U., & Singh, G. (2015). A review on software maintenance issues and how to reduce maintenance efforts. *International Journal of Computer Applications*, *118*(1), 6–11. https://d1wqtxts1xzle7.cloudfront.net/49247409/A_Review_on_Software_Maintenance_Issues_and_How_to_Reduce_Maintenance_Efforts-libre.pdf

Michael, L. G., Donohue, J., Davis, J. C., Lee, D., & Servant, F. (2019). Regexes are hard: Decision-making, difficulties, and risks in programming regular expressions. *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 415–426. https://arxiv.org/pdf/2303.02555

Reid, B., Treude, C., & Wagner, M. (2023). Using the TypeScript compiler to fix erroneous Node.js snippets. *2023 IEEE 23rd International Working Conference on Source Code Analysis and Manipulation (SCAM)*, 220–230. https://arxiv.org/pdf/2308.12079

Sun, W., Fang, C., Miao, Y., You, Y., Yuan, M., Chen, Y., Zhang, Q., Guo, A., Chen, X., Liu, Y., e.a. (2023). Abstract syntax tree for programming language understanding and representation: How far are we? *arXiv preprint arXiv:2312.00413*. https://arxiv.org/pdf/2312.00413

Vaniea, K., & Rashidi, Y. (2016). Tales of software updates: The process of updating software. *Proceedings of the 2016 chi conference on human factors in computing systems*, 3215–3226. https://doi.org/10.1145/2858036.2858303