ONDERZOEKSVOORSTEL

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

Bachelorproef, 2025-2026

Wauters Sander

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

Samenvatting

Een applicatie updaten naar een nieuwe versie van een softwareframework kan veel tijd in beslag nemen, zeker als de applicatie meerdere versies achterloopt. Hetzelfde updateproces vervolgens herhalen over meerdere applicaties zorgt ervoor dat de onderhoudstijd snel toeneemt. Het doel van dit onderzoek is om de update van het Angular-webframework van versie 16 naar versie 20 in meerdere applicaties te automatiseren, met als doel de onderhoudstijd voor de ontwikkelaars te verlagen. Framework updates kunnen de performantie en veiligheid van de applicatie verbeteren. Het is daarom belangrijk om deze tijdig 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 om alle veranderingen systematisch aan te brengen veel tijd in beslag. Dit onderzoek start met het maken van een oplijsting 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 proof of concept ontwikkeld die het updateproces zal automatiseren waar mogelijk. Deze proof of concept wordt eerst getest in een gecontroleerde omgeving en vervolgens op een reële applicatie van een bedrijf. De effectiviteit wordt beoordeeld aan de hand van het aantal nodige veranderingen die het automatisatieproces correct kan opsporen en uitvoeren ten opzichte van het totaal aantal uit te voeren aanpassingen. Dit onderzoek verwacht dat het 65% van alle nodige veranderingen kan automatiseren. Het automatisatieproces zal naar verwachting de nodige tijd om 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 als consumentenkredieten. Ze ondersteunen de kredietverstrekker voor de dienstverlening aan consumenten. Binnen het bedrijf zijn er verschillende applicaties die gebruikmaken van het Angular webframework. Angular is een open-source front-end framework, ontwikkeling door Google, gebaseerd op de TypeScript programmeertaal voor de ontwikkeling van dynamische webapplicaties (Cincovic e.a., 2019). Momenteel is Angular versie 20 (v20) de meest recente stabiele versie. Binnen Stater maken de applicaties gebruik van Angular versie 16 (v16). Het bedrijf is van plan de applicaties te updaten naar de recentste versie, Angular v20.

De updates niet uitvoeren is geen optie. Volgens de studie door Vaniea en Rashidi (2016) zijn software-updates nodig, omdat het nieuwe functionaliteiten introduceert, de performantie verbetert en verzekert dat de software compatibel blijft met andere nieuwe software. Verder omschrijft deze bron dat het up-to-date houden van software cruciaal is om de cyberveiligheid te garanderen.

Het grote verschil in versies zorgt ervoor dat het updaten van alle applicaties veel tijd in beslag neemt. Dit is geen eenmalig probleem; volgens Callaghan (2023) krijgt Angular een nieuwe versie om de 6 maanden. De studie door Kaur en Singh (2015) beweert dat het onderhouden van een softwareproject gemiddeld 60% van de kostprijs in beslag neemt. Een manier om de tijd voor software-onderhoud in te korten is daarom best interessant. Hieruit komt de vraag: in welke mate kan de automatisering van het updateproces van Angular v16 naar v20, bij meerdere applicaties, de

onderhoudstijd voor de ontwikkelaars verlagen? Om deze vraag te beantwoorden zijn de volgende deelvragen geformuleerd:

- Hoeveel veranderingen moeten uitgevoerd worden om Angular van v16 naar v20 te updaten?
- Welke manieren bestaan om code automatisch aan te passen zonder ongewenste veranderingen uit te voeren?
- Welke manier om code automatisch aan te passen is het meest geschikt om in deze casus toe te passen?
- Wat zijn statistisch gezien de meest voorkomende problemen bij het updaten van code?

Gedurende dit onderzoek zal als proof of concept een applicatie ontwikkeld worden die een softwareproject in Angular v16 automatisch updatet naar Angular v20. 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. Het doelpubliek van de updater zijn de personen die anders deze aanpassingen aan de broncode manueel uitvoeren. De effectiviteit van de updater wordt bepaald aan het aantal gedetecteerde en opgeloste aanpassingen tegenover het totaal van de nodige aanpassingen.

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 is. Tenslotte worden de verwachte resultaten besproken, waarin een inschatting wordt gegeven van de bevindingen van het onderzoek.

2. Literatuurstudie

2.1. Uit te voeren veranderingen

De Angular update guide door Google (2025) geeft een uitgebreid overzicht van alle aanpassingen die nodig zijn om een Angular-applicatie van v16 naar v20 te updaten. Uit deze bron blijkt dat in totaal 80 verschillende stappen uitgevoerd moeten worden. Deze stappen gaan van het uitvoeren van commando's tot verschillende aanpassingen aan de code.

Zoals omschreven in de studie door Cincović en Punt (2020), is de code in Angular applicaties onderverdeeld in 3 verschillende soorten bestanden:

- TypeScript-bestanden die de bedrijfslogica bevatten.
- HTML-bestanden die de structuur van de user interface (UI) omschrijven.
- CSS-bestanden die de visuele representatie van de UI omschrijven.

De studie door Di Penta e.a. (2020) onderzoekt welke veranderingen aan code de meeste kans hebben om nieuwe bugs te introduceren. Deze studie maakt het mogelijk om een geïnformeerde inschatting te maken van welke aanpassingen geautomatiseerd kunnen worden zonder ongewenste bijwerkingen te introduceren.

2.2. Het automatisatie process

Eén van de simpelste manieren om code in bulk aan te passen is het gebruikmaken van zoek- en vervangfuncties gebaseerd op text of reguliere expressies (Regex). De studie van Michael e.a. (2019) omschrijft Regex als een sequentie van karakters die een patroon in een tekst omschrijft. Uit dezelfde studie blijken enkele problemen bij de implementatie van Regex, namelijk dat het moeilijk leesbaar, vindbaar, valideerbaar en documenteerbaar is. Verder kan Regex geen rekening houden met de semantiek van de programmeertaal, aangezien het enkel op tekst gebaseerd is.

Om met de semantiek van de programmeertaal rekening te houden, kan gebruikgemaakt worden van een abstract syntax tree. Zoals omschreven door Sun e.a. (2023), een abstract syntax tree is een datastructuur die de broncode van een applicatie illustreert en rekening houdt met de syntax en semantiek van de programmeertaal. In combinatie met zoek- en vervangfuncties laat dit toe om een stuk code aan te passen, enkel als het in een bepaalde context zit.

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

Een alternatief voor de TypeScript Compiler API dat ook gebruikmaakt van een abstract syntax tree is het Angular 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 programmeertaalspecifieke functies voorziet zoals: automatisch code aanvullen en code-diagnostiek. Dezelfde bron omschrijft LSP's als het de facto standaardprotocol voor de implementatie van deze functies in IDE's.

Tenslotte is het mogelijk om artificiële intelligentie in te zetten om deze aanpassingen te maken. Met de recente opkomst van AI-tools die specifiek gemaakt zijn voor programmeren, is het mogelijk om deze taak uit te besteden aan AI. Er zijn echter problemen met deze aanpak voor deze casus. Uit de studie door Hodovychenko en Kurinko (2025) blijkt dat AI-gedreven tools een gebrek hebben aan transparantie en risico lopen de semantiek van de programmeertaal in de loop van de tijd fout te interpreteren. Verder maakt deze studie de bewering dat voor het maken van dit soort AI-tools er nood is aan een grote hoeveelheid betrouwbare trainingsdata. Het bemachtigen van deze data is problematisch, vooral als het gaat om code die gebruikmaakt van de allernieuwste updates.

3. Methodologie

3.1. Literatuurstudie

Dit onderzoek start met een uitgebreide literatuurstudie naar de verschillen tussen Angular v16 en Angular v20. De nodige veranderingen worden opgelijst en onderverdeeld in verschillende categorieën. Deze oplijsting bepaalt de capaciteit van de updater. Verder geeft dit een beter overzicht van welke aanpassingen al dan niet geschikt zijn voor automatisatie. Tenslotte zal deze oplijsting dienen als maatstaf om de effectiviteit van de updater op te meten.

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

- Complexiteit van de implementatie. Een voorkeur wordt gegeven aan het gebruiken van bestaande tools boven het ontwikkelen van nieuwe algoritmes.
- Betrouwbaarheid van de output. Zolang de input hetzelfde blijft, mag de output niet veranderen.
- Gebruiksvriendelijkheid. Het moet bruikbaar zijn voor de persoon die normaal manueel de applicaties updatet.

3.2. Ontwikkeling van de proof of concept

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

De updater heeft als minimum de volgende twee functies: het detecteren van code die aangepast moet worden en de aanpassingen uitvoeren indien mogelijk. Het detecteren van de code speelt een dubbele rol. In eerste instantie is het nodig om de aanpassing op de correcte plaats uit te voeren. Indien de aanpassing niet geautomatiseerd kan worden, zorgt het voor een overzicht van waar alle nodige aanpassingen gemaakt moeten worden.

De updater wordt in een gecontroleerde omgeving getest om de stabiliteit te verzekeren. Deze gecontroleerde omgeving bestaat uit een testapplicatie gemaakt in Angular v16. De testapplicatie bevat een codefragment voor elke vooraf geïdentificeerde stap in het updateproces.

3.3. Evaluatie

De effectiviteit van de updater wordt bepaald aan het aantal gedetecteerde en opgeloste aanpassingen tegenover het totaal aantal aanpassingen.

Een eerste meting wordt uitgevoerd op de gecontroleerde omgeving die gemaakt is in de proof of concept. Dit geeft een totaalresultaat voor alle aanpassingen die theoretisch nodig zijn.

Tenslotte wordt een tweede meting uitgevoerd op één van de applicaties binnen Stater. Dit geeft een resultaat voor alle aanpassingen die praktisch nodig zijn.

4. Verwacht resultaat, conclusie

Op basis van de literatuurstudie en de gehanteerde methodologie verwacht dit onderzoek dat minstens 65% van alle nodige aanpassingen automatisch uitgevoerd kan worden.

Het automatisatieproces zal naar verwachting de nodige tijd voor de applicaties te updaten verminderen. De totale hoeveelheid tijd die in werkelijkheid bespaard wordt, is afhankelijk van verschillende factoren: de ervaring van de programmeur, hun kennis van de broncode, de grootte van de applicaties, ... Deze oplossing zal wellicht meer tijd in beslag nemen als enkel één kleine applicatie geüpdatet moet worden.

Dit onderzoek tracht de beste methode te implementeren voor deze casus op basis van gekende literatuur. Echter, kan het interessant zijn om andere manieren te implementeren en te vergelijken. Verder onderzoek kan uitgevoerd worden naar de performantie, accuraatheid en complexiteit van de verschillende implementaties besproken in de literatuurstudie.

Referenties

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

Callaghan, M. D. (2023). Upgrading Angular. In *Angular for Business: Awaken the Advocate Within and Become the Angular Expert at Work* (pp. 95–118). Springer. https://doi.org/10.1007/978-1-4842-9609-7_8

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

Cincović, J., & Punt, M. (2020). Comparison: Angular vs. React vs. Vue. Which framework is the best choice? *Zdravković, M., Konjović, Z., Trajanović, M.(Eds.) ICIST 2020 Proceedings*, 250–255. https://www.eventiotic.com/eventiotic/files/Papers/URL/50173409-699e-4b17-8edb-9764ecc53160.pdf

Di Penta, M., Bavota, G., & Zampetti, F. (2020). On the relationship between refactoring actions and bugs: a differentiated replication. *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 556–567. https://doi.org/10.1145/3368089.3409695

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

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://doi.org/10.5120/20707-3021

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://doi.org/10.1109/ASE.2019.00047

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://doi.org/10.1109/SCAM59687.2023.00031

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://doi.org/10.48550/arXiv.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