

# Proof Of Concept: Linter voor BibLaTeX.

Automatische controle van bibliografische referentielijsten.

---

**Tristan Cuvelier.**

Scriptie voorgedragen tot het bekomen van de graad van  
Professionele bachelor in de toegepaste informatica

**Promotor:** Dhr. Bert Van Vreckem

**Co-promotor:** Dhr. Bert Van Vreckem

**Academiejaar:** 2023–2024

**Eerste examenperiode**

**Departement IT en Digitale Innovatie .**

**HO  
GENT**



# Woord vooraf

Het schrijven van deze bachelorproef was een uitdagende en leerzame ervaring die ik niet had kunnen voltooien zonder de steun en begeleiding van verschillende mensen. Allereerst wil ik mijn promotor en co-promotor, *Dhr. Bert Van Vreckem*, bedanken voor zijn voortdurende steun, geduld en zowel snelle als waardevolle feedback gedurende het hele proces. Zijn deskundigheid en begeleiding hebben mijn onderzoek naar een hoger niveau getild en dit project mogelijk gemaakt. Ik had geen geschiktere persoon kunnen wensen om me hierbij te begeleiden.

Daarnaast wil ik mijn dank uitspreken naar mijn docenten van het Departement IT en Digitale Innovatie, die mij de benodigde kennis en vaardigheden hebben bijgebracht gedurende mijn opleiding. Ook wil ik mijn medestudenten bedanken voor hun steun en samenwerking. Hun feedback was meer dan welkom en zonder hen zouden er nog meer bugs in de linter zitten en zou het nut ervan nog niet bewezen zijn. In het bijzonder wil ik mijn medestagiairs, *Jasper V.D.* en *Sarah E.*, extra bedanken voor de "*last-minute bug reports*" die hebben geholpen om meerdere toekomstige problemen te voorkomen.

Ik heb gekozen voor het onderwerp "**Proof Of Concept: Linter voor BibLaTeX**" vanwege mijn interesse in softwareontwikkeling en mijn passie voor het helpen van mensen. Het ontwikkelen van een linter voor BibLaTeX biedt een concrete oplossing voor een veelvoorkomend probleem bij studenten en onderzoekers, namelijk het correct beheren van bibliografische referenties. Door dit probleem aan te pakken, kon ik een impact maken op de kwaliteit van bachelorproeven en andere (academische) werken, waardoor ik wellicht meer mensen heb kunnen helpen dan ik me kon voorstellen.

Deze bachelorproef onderzoekt de huidige stand van zaken op het gebied van BibLaTeX-linters, de technische uitdagingen en mogelijkheden, en presenteert een proof of concept voor een linter die specifiek is ontworpen voor gebruik met BibLaTeX. Het doel is om een hulpmiddel te bieden dat fouten in bibliografische referenties automatisch detecteert en corrigeert, wat de kwaliteit van academisch werk ten goede komt.

De afgelopen maanden waren intensief maar ook verrijkend. Ik heb veel geleerd over softwareontwikkeling, projectmanagement en het uitvoeren van technisch

onderzoek. Dit project heeft mijn interesse in het vakgebied verder aangewakkerd en ik kijk ernaar uit om de groei van het bekomen resultaat te kunnen volgen.

Tot slot wens ik graag een woord van dank te richten aan *Arne Van Den Kerchove* voor het maken van bibl, wat uiteindelijk de hele basis werd van deze proof of concept. Zonder zijn professionele werk zou bibla niet staan waar het vandaag staat. Ook wil ik *Alexander Veldeman* bedanken, die als eerste de kans greep om de online versie van bibla te testen en mij extra energie gaf met zijn enthousiasme over dit project. Daarnaast wil ik mijn vriendin bedanken, omdat ze voor mij gezorgd heeft tijdens deze drukke periodes en steeds met veel geduld bleef omgaan met mij. Ik ben er mij van bewust dat het niet altijd even leuk was.

Verder wil ik mijn familie bedanken voor hun voortdurende steun en de mogelijkheden die ze mij hebben gegeven om dit te kunnen bereiken.

Ik hoop dat dit werk bijdraagt aan verdere ontwikkelingen op dit gebied en dat het nuttig zal zijn voor toekomstige studenten en onderzoekers.

Met dank en waardering,

Tristan Cuvelier

# Samenvatting

Deze bachelorproef richt zich op de ontwikkeling van een **Proof of Concept (PoC)** voor een BibLaTeX-linter. Het doel van dit project is om een open-source tool te creëren die studenten en onderzoekers helpt bij het identificeren en bewustmaken van fouten in bibliografische referenties in BibLaTeX.

De motivatie voor deze studie komt voort uit de behoefte aan een betrouwbaar hulpmiddel om veelvoorkomende fouten in referentielijsten te detecteren. Meer bepaald door lectoren aan HOGENT, de **Proof of Concept** volgt dan ook hun ideologie. Dit helpt de kwaliteit en nauwkeurigheid van academische werken te verbeteren. Daarnaast hoeven lectoren aan HOGENT niet telkens dezelfde fouten aan te duiden bij studenten, aangezien het vaak dezelfde fouten zijn die terugkeren. Hierdoor komt er tijd vrij om belangrijkere fouten op te merken en kan het niveau naar een hoger niveau worden getild.

Het onderzoek is uitgevoerd door middel van een uitgebreide literatuurstudie, een technische analyse van bestaande linters en programmeertalen, en de eigenlijke softwareontwikkeling van de linter. De literatuurstudie gaf inzicht in de huidige stand van zaken en best practices, terwijl de technische analyse zich richtte op het evalueren van geschikte programmeertalen en tools. De softwareontwikkeling omvatte het ontwerpen, implementeren en testen van de linter, met aandacht voor gebruiksvriendelijkheid en uitbreidbaarheid.

De resultaten van dit project tonen aan dat de ontwikkelde linter effectief is in het detecteren van fouten in BibLaTeX-referenties. De linter werd grondig getest door verschillende gebruikers, wat leidde tot waardevolle feedback en suggesties voor verdere verbeteringen. De belangrijkste bijdrage van dit onderzoek is een solide basis voor een linter die verder ontwikkeld en aangepast kan worden door de community en bewezen is om compatibel te zijn met BibLaTeX.

De relevantie van deze resultaten voor het werkveld is aanzienlijk. De linter kan door studenten, docenten, onderzoekers en  $\text{\LaTeX}$ -gebruikers in het algemeen worden gebruikt om de kwaliteit van hun (academische) werk te verhogen, wat de betrouwbaarheid en professionaliteit van hun publicaties ten goede komt. Bovendien biedt het project een platform voor toekomstige ontwikkelingen en verbeteringen binnen de open-source community.

Met deze **Proof of Concept** is een belangrijke stap gezet naar een volledig functionele BibLaTeX-linter. Het is de hoop dat dit project als fundament zal dienen voor verdere innovatie en samenwerking in het academische veld.

# Inhoudsopgave

<b>Lijst van figuren</b>	<b>ix</b>
<b>Lijst van tabellen</b>	<b>x</b>
<b>Lijst van Codefragmenten</b>	<b>xi</b>
<b>Glossary</b>	<b>xiii</b>
<b>1 Inleiding</b>	<b>1</b>
1.1 Probleemstelling	2
1.2 Onderzoeksdoelstelling	2
1.3 Opzet van deze bachelorproef	2
<b>2 Stand van zaken</b>	<b>4</b>
2.1 Wat is een linter?	4
2.2 $\text{\LaTeX}$	5
2.2.1 Werking	5
2.2.2 Voordelen	5
2.2.3 Nadelen	5
2.2.4 Conclusie	5
2.3 BibTeX en BibLaTeX	6
2.3.1 Inleiding	6
2.3.2 Overzicht van BibTeX	6
2.3.3 Overzicht van BibLaTeX	7
2.3.4 Gedetailleerde Vergelijking	8
2.3.5 Conclusie	9
2.4 Huidige stand van zaken	10
2.5 Andere linters	10
2.5.1 Ruff	11
2.5.2 DirtyRat	11
2.5.3 bibl	12
2.6 Programmeertaal	14
2.6.1 Rust	14
2.6.2 Python	15
2.6.3 JavaScript	15

2.7	Opbouw linter	15
2.8	bibl pakket - Diepere blik	16
2.8.1	__init__.py	17
2.8.2	__main__.py	17
2.8.3	bibl.yml - Configuratiebestand	20
2.8.4	config.py - Configuratielogica	22
2.8.5	lint.py - Hoofdlogica	25
2.8.6	rule.py - Structuur voor het Beheren van Regels	29
2.8.7	text_utils.py - Helper Functies voor Tekstverwerking	34
2.8.8	Regels	38
2.8.9	Conclusie	42
2.9	Open-source	43
2.10	Pipelines	44
2.11	Kanban	45
<b>3</b>	<b>Methodologie</b>	<b>47</b>
3.1	Fase 1: Literatuurstudie	47
3.2	Fase 2: Technische Analyse en Experimenten	47
3.3	Fase 3: Software Ontwikkeling   Proof of Concept	48
3.4	Conclusie	48
<b>4</b>	<b>Technische Analyse en Experimenten</b>	<b>49</b>
4.1	Functionele en Niet-Functionele Requirements	50
4.2	Keuze van de Programmeertaal	50
4.2.1	JavaScript	51
4.2.2	Python	54
4.2.3	Rust	56
4.2.4	Keuze	59
4.3	Gebruik van bibl als Basis	60
4.3.1	Overbodige regels	61
<b>5</b>	<b>Uitwerkingsfase</b>	<b>64</b>
5.1	bibla - 2.0.5	64
5.1.1	Behouden regels	65
5.1.2	Aangepaste regels	65
5.1.3	Nieuwe regels	69
5.1.4	Diverse aanpassingen	72
5.1.5	Configuratie	73
5.1.6	Pipelines	74
5.1.7	Testen	77

<b>6 Conclusie</b>	<b>78</b>
6.1 Nog uit te werken requirements	78
6.2 Was de proof of concept een succes?	79
6.3 Toekomstvisie	79
6.4 Aanbevelingen	80
<b>A Onderzoeksvoorstel</b>	<b>82</b>
A.1 Introductie	82
A.2 State-of-the-art	83
A.3 Methodologie	83
A.3.1 Fase 1: Literatuurstudie	84
A.3.2 Fase 2: Technische Analyse en Experimenten	84
A.3.3 Fase 3: Software Ontwikkeling (PoC - Proof of Concept)	85
A.3.4 Algemeen + Fase 4	85
A.4 Verwacht resultaat, conclusie	85
<b>Bibliografie</b>	<b>87</b>



# Lijst van figuren

2.1	Foutmelding BibLaTeX-linter	10
2.2	bibl repository	13
2.3	bibl repository - Source code	17
6.1	Effectiviteit bibla	81

# Lijst van tabellen

4.1 Geprioriteerde Functionele Vereisten   MoSCoW . . . . .	62
4.2 Tabel met de uitvoeringstijden van elke PoC (in milliseconden) . . . . .	63

# Lijst van Codefragmenten

2.8.1	bibl: __main__.py - Imports	18
2.8.2	bibl: __main__.py - CLI parameters	19
2.8.3	bibl: __main__.py - Commando: lint	19
2.8.4	bibl: __main__.py - Commando: toon alle regels	20
2.8.5	bibl: __main__.py - Commando: huidige versie	20
2.8.6	bibl: __main__.py - Commando: bibl vanuit CLI	20
2.8.7	bibl: bibl.yml - Configuratie	21
2.8.8	bibl: config.py - Configuratielogica	24
2.8.9	bibl: lint.py - Hoofdlogica	28
2.8.10	bibl: rule.py - Regelstructuur basis	31
2.8.11	bibl: rule.py - Regels voor bibliografie entry en tekstlijn evaluatie	32
2.8.12	bibl: rule.py - Regelcontainer en registratiemechanisme	33
2.8.13	bibl: rule.py - Registratie en laden van regels	34
2.8.14	bibl: text_utils.py - Hulpmiddelen en constanten	35
2.8.15	bibl: tex_utils.py - Regelnummer en offset zoeken met regex	36
2.8.16	bibl: tex_utils.py - Regelnummer zoeken van een BibTeX entry	36
2.8.17	bibl: tex_utils.py - Markdown tabellen formatteren voor regels	37
2.8.18	bibl: database_rules.py - Regel voor preamble op eerste regel	39
2.8.19	bibl: database_rules.py - Regel voor mogelijke dubbele invoer op basis van vergelijkbare titels	39
2.8.20	bibl: entry_rules.py - Regel voor vervanging van 'et al.' in auteurveld	40
2.8.21	bibl: field_rules.py - Regel voor ontbrekend verplicht veld	42
4.2.1	Mini-PoC: Test bestand	51
4.2.2	Mini-JS-PoC - Importeren van submodules	51
4.2.3	Mini-JS-PoC - Extractie van alle vermeldingen	52
4.2.4	Mini-JS-PoC - Reguliere expressies voor entries en fields	52
4.2.5	Mini-JS-PoC - Controleren op ontbrekende velden	53
4.2.6	Mini-JS-PoC - Controleren op duplicaten	54
4.2.7	Mini-Python-PoC - Verwerking van BibLaTeX vermeldingen	55
4.2.8	Mini-Python-PoC - Controleren op ontbrekende velden en duplicaten	56
4.2.9	Mini-Rust-PoC - Verwerking van BibLaTeX vermeldingen en controle op duplicaten	57
4.2.10	Mini-Rust-PoC - Controleren op duplicaten	58
5.1.1	bibla: E00 - Correct key formaat	67
5.1.2	bibla: E09 - Correct date formaat	68

5.1.3 bibla: E10 - Gebruik aanbevolen velden . . . . .	70
5.1.4 bibla: E12 - Geen startpagina's gebruiken . . . . .	71
5.1.5 bibla: E13 - Enkel kritische pad van URL . . . . .	71
5.1.6 bibla: M03 - Speciale karakters dienen gegenereerd te worden . . . . .	72
5.1.7 bibla: Extra foutafhandelingen . . . . .	73
5.1.8 bibla: Configuratie   E10 - Gebruik aanbevolen velden . . . . .	73
5.1.9 bibla: CI-pipeline . . . . .	75
5.1.10 bibla: CD-pipeline . . . . .	77

# Woordenlijst

**AI** Artificiële Intelligentie. 54, 56, 77

**API** Application Programming Interface. 76

**CD** Continuous Deployment. xii, 44, 45, 77

**CI** Continuous Integration. xii, 11, 44, 45, 47, 75

**CLI** Command Line Interface. xi, 17–20, 50

**GUI** Graphical User Interface. 15

**JS** JavaScript. xi, 51–54

**MVP** Minimum Viable Product. 46, 49

**PoC** Proof of Concept. v, vii, x, xi, 3, 4, 9–12, 14–17, 43–45, 47, 48, 50–58, 63, 65, 72, 77

**Regex** Reguliere Expressie. 67, 68, 70, 71

**WYSIWYG** What You See Is What You Get. 5

# 1

## Inleiding

LaTeX ( $\text{\LaTeX}$ ) uitspraak: la-tech; is een populair softwaresysteem in de wetenschappelijke wereld omdat het uitblinkt in het zetten van technische documenten en beschikbaar is voor bijna alle computersystemen (Oetiker e.a., 2023).

Niet alleen in de wereld van de wetenschap wordt  $\text{\LaTeX}$  gebruikt. Ook studenten op hogescholen en universiteiten maken er gebruik van voor het schrijven van bachelor- en/of masterproeven.

Bij het schrijven van, al dan niet wetenschappelijke, teksten is het uitermate belangrijk om aan een correcte vorm van bronvermelding te doen.

Binnen  $\text{\LaTeX}$  zijn er verschillende manieren om dit aan te pakken. Eén van deze manieren is met behulp van BibLaTeX, een package speciaal gebouwd voor deze taak.

Studenten te HOGENT dienen gebruik te maken van deze combinatie bij het schrijven van hun bachelorproef. Ondanks dat lectoren veel moeite steken in het bondig toelichten van het correcte gebruik, worden er nog veel fouten gemaakt bij het correct bijhouden van bronnen.

Op deze groep zal deze bachelorproef zich focussen. Met behulp van een statische analysetool, ook wel linter genoemd, zouden al veel van de herhalende fouten voorkomen kunnen worden. Een linter is een programma dat broncode of gestructureerde dataformaten kan controleren op stijl, syntax en logische fouten (Kamunya, 2023).

Het zou daarom uitermate geschikt zijn om de studenten een linter te laten gebruiken om hen zo te helpen bij het voorkomen of opsporen van de gemaakte fouten. Zo dienen lectoren hen niet keer op keer op dezelfde fouten te wijzen.

BibLaTeX is voortgekomen uit BibTeX en biedt meer opties om bibliografieën en citaten te configureren (Cassidy, 2013). Hoewel er voor BibTeX reeds een linter bestaat, is deze niet compatibel met BibLaTeX.

Het doel van deze bachelorproef is om een proof of concept, analyse en de software-architectuur uit te werken voor een BibLaTeX-linter en er een prototype voor te schrijven in een passende programmeertaal.

Concreet betekent dit:

- De lijst van gewenste functionele en niet-functionele requirements aanvullen en structureren naar prioriteiten.
- De werking van bestaande linters bestuderen als inspiratiebron.
- Een gemotiveerde keuze maken voor de te gebruiken programmeertaal en eventuele libraries.
- Een prototype met een minimale set van linting-regels implementeren.
- Unit tests schrijven met zo compleet mogelijke code coverage.
- CI-pipeline opzetten voor packaging en testing.
- Documentatie schrijven voor het gebruik en uitbreiden van de linting-regels.

## 1.1. Probleemstelling

Studenten van HOGENT en derden die teksten schrijven in  $\LaTeX$  en BibLaTeX gebruiken voor hun bronvermeldingen, kunnen baat hebben bij deze proof of concept. Er wordt een (open source) linter gemaakt die ervoor zorgt dat er makkelijker fouten kunnen worden opgespoord en opgelost binnen de bronnenlijst. Merk wel op: de linter proof of concept wordt opgesteld met de regels die door HOGENT worden opgelegd aan haar studenten. Dit betekent dat er meer dan strikt noodzakelijke velden verwacht worden bij bepaalde types bronnen. Eventueel wordt er een parameter voorzien om dit al dan niet aan te passen.

## 1.2. Onderzoeksdoelstelling

Deze bachelorproef heeft als doel een open source proof of concept op te zetten waar iedereen die een bijdrage wenst te leveren dit ook kan doen. Op deze manier zal er een bruikbare linter ontstaan voor de studenten van HOGENT en derden die baat hebben bij een linter voor BibLaTeX.

## 1.3. Opzet van deze bachelorproef

De rest van deze bachelorproef is als volgt opgebouwd:

In Hoofdstuk 2 wordt een overzicht gegeven van de stand van zaken binnen het landschap van BibLaTeX-linters, op basis van een literatuurstudie. Daarnaast wordt er ook direct dieper onderzoek uitgevoerd naar allerlei informatie die gebruikt kan worden om zelf iets uit te werken.

In Hoofdstuk 3 wordt de methodologie toegelicht en worden de gebruikte onderzoekstechnieken besproken om een antwoord te kunnen formuleren op de onderzoeksvragen.

In Hoofdstuk 4 wordt een grondige technische analyse uitgevoerd, waarbij ook een lijst aan functionele en niet-functionele requirements wordt opgesteld. Daarnaast worden er beslissingen genomen die invloed hebben op het volgende hoofdstuk.

In Hoofdstuk 5 vindt de effectieve uitwerking plaats van de **Proof of Concept**.

In Hoofdstuk 6, tenslotte, wordt de conclusie gegeven en vindt er een reflectie plaats over het bereikte resultaat. Daarbij wordt ook een aanzet gegeven voor toekomstige visies.



# 2

## Stand van zaken

Dit hoofdstuk licht toe wat een linter is en wat  $\LaTeX$  en BibLaTeX zijn. Daarnaast wordt er meer verteld over de huidige stand van zaken binnen de wereld van BibLaTeX-linters, het toont aan waarom het gepast is om deze *Proof of Concept* uit te werken en de opportuniteiten die zich bieden binnen dit onderwerp. Alsook wordt er een diepere kijk gegeven aan andere linters, hun functionaliteiten en performantie om een beter inzicht te verwerven in aspecten die van belang kunnen zijn voor het uitwerken van een eigen linter.

### 2.1. Wat is een linter?

Alvorens er uitgelegd wordt waarom het nuttig is om deze *Proof of Concept* uit te werken, is het belangrijk om een goed begrip te hebben van wat er effectief gemaakt wordt en wat het nut ervan is. Het doel van deze *Proof of Concept* is om een linter te maken. Maar wat is dat juist?

Kamunya (2023) legt uit dat linter verwijst naar het proces van broncode automatisch controleren op programmatische en stilistische fouten. Dit houdt in dat een linter programmatisch je code scant om te controleren of er problemen zijn die kunnen leiden tot bugs of inconsistenties met de code-stijl en -gezondheid. De linter is hierbij de tool die ervoor gebruikt wordt.

Een linter is een statische analysetool omdat het de broncode of andere gestructureerde data analyseert zonder de code daadwerkelijk uit te voeren (Møller & Schwartzbach, 2023). Het voert een *statische* analyse uit, wat betekent dat het de code inspecteert op basis van de geschreven tekst en structuur ervan, zonder rekening te houden met de daadwerkelijke uitvoering van de code.

Een linter is dus een statische analysetool die broncode of andere gestructureerde data kan analyseren.

## 2.2. L<sup>A</sup>T<sub>E</sub>X

L<sup>A</sup>T<sub>E</sub>X (uitspraak: LaTech), een uitbreiding op het T<sub>E</sub>X-typesetting systeem van Donald E. Knuth (1984), is bedoeld voor het opmaken van tekst en wiskundige formules. TeX werd ontwikkeld in 1977 met als doel de typografische kwaliteit te verbeteren. De stabiele versie van TeX kwam uit in 1982 en ondersteunt meerdere talen en 8-bit karakters. L<sup>A</sup>T<sub>E</sub>X zelf, ontwikkeld door Lamport (1994) in 1985, voegt een reeks macro's toe om het gebruik van TeX te vereenvoudigen en heeft zich ontwikkeld tot een standaard voor het produceren van wetenschappelijke en wiskundige documentatie (Oetiker e.a., 2023).

### 2.2.1. Werking

Personen die al ervaring hebben met L<sup>A</sup>T<sub>E</sub>X zullen net als Oetiker e.a. (2023) kunnen bevestigen dat L<sup>A</sup>T<sub>E</sub>X functioneert door middel van commando's die de logische structuur van een document definiëren (zoals hoofdstukken, secties, en paragrafen) en dat dit anders is dan de typische **What You See Is What You Get (WYSIWYG)** tekstverwerkers zoals Microsoft Office Word, waar de lay-out interactief wordt bepaald tijdens het typen. L<sup>A</sup>T<sub>E</sub>X vereist dat de auteur zijn tekst structureert met behulp van vooraf gedefinieerde commando's die de inleiding van het document bepalen.

### 2.2.2. Voordelen

1. Het biedt professionele vormgeving van lay-outs.
2. Het ondersteunt de zetting van wiskundige formules.
3. Het moedigt aan tot goed gestructureerd schrijven. Dit resulteert in duidelijk georganiseerde documenten.
4. Er zijn veel uitbreidingen beschikbaar via packages om functionaliteit zoals PDF-output en betere font-ondersteuning toe te voegen.

### 2.2.3. Nadelen

1. Het instellen van een volledig nieuwe lay-out kan ingewikkeld en tijdrovend zijn.
2. Niet ideaal voor zeer ongestructureerde documenten.
3. Leercurve, de *slechte* gewoontes afleren van typische **WYSIWYG** programma's vergt enige tijd om aan te wennen.

### 2.2.4. Conclusie

L<sup>A</sup>T<sub>E</sub>X (Oetiker e.a., 2023) wordt vooral gewaardeerd in academische en technische kringen waar de precisie van de inhoud en de structuur voorop staan. Eens er een

lay-out template bestaat, is het zeer eenvoudig om veel professionele documenten op te maken op eenzelfde manier. Met dank aan: de uitbreidbaarheid, consistentie, brede ondersteuning van wiskundige formules, en de mogelijkheid om complexe documenten zoals proefschriften en wetenschappelijke artikelen nauwkeurig op te maken, ook wel typesetten genoemd blijft  $\LaTeX$  relevant.

## 2.3. BibTeX en BibLaTeX

### 2.3.1. Inleiding

BibTeX en BibLaTeX zijn hulpmiddelen die worden gebruikt voor het beheren van referenties in  $\LaTeX$ -documenten. Beide hulpmiddelen hebben verschillende kenmerken en mogelijkheden die inspelen op verschillende behoeften van gebruikers bij het beheren van bibliografieën. Deze vergelijking belicht de technische aspecten van beide en biedt een diepgaande analyse om de verschillen tussen beide beter te begrijpen.

### 2.3.2. Overzicht van BibTeX

BibTeX werd in 1985 gecreëerd door Patashnik (1988) en is ontworpen om samen te werken met  $\LaTeX$ . Het vergemakkelijkt het genereren van bibliografieën door gebruikers in staat te stellen citatievermeldingen te definiëren in een `.bib`-bestand en deze te formatteren volgens vooraf gedefinieerde stijlen.

#### Belangrijkste Kenmerken van BibTeX

1. **Compatibiliteit:** BibTeX is compatibel met  $\LaTeX$  versie 2.09 en latere versies. Het integreert goed met  $\LaTeX$ -documenten en maakt gebruik van `.bst`-bestanden voor stijldefinities (Patashnik, 1988).
2. **Kruisverwijzingen:** BibTeX ondersteunt kruisverwijzingen (Engels: cross-referencing) tussen vermeldingen, waardoor een vermelding velden van een andere kan erven. Dit is nuttig voor het citeren van proceedings en verzamelwerken (Patashnik, 1988).
3. **Geaccentueerde Karakters:** BibTeX kan geaccentueerde karakters verwerken, wat cruciaal is voor internationalisatie. Het vereist dat geaccentueerde karakters tussen accolades worden geplaatst om correcte verwerking te garanderen (Patashnik, 1988).
4. **Sorteren en Labels:** BibTeX sorteert vermeldingen standaard op auteur, jaar en titel. Aangepaste sorteer- en labelgeneratie kan worden gespecificeerd in `.bst`-bestanden (Patashnik, 1988).
5. **String-concatenatie:** BibTeX ondersteunt string-concatenatie in veldwaarden, wat de flexibiliteit van vermeldingsdefinities vergroot (Patashnik, 1988).

6. **Preamble-commando:** Het `@PREAMBLE`-commando stelt gebruikers in staat om  $\LaTeX$ -commando's in de bibliografie op te nemen, wat extra aanpassings-opties biedt (Patashnik, 1988).

### Beperkingen van BibTeX

1. **Aanpassingsbeperkingen:** Het aanpassen van bibliografische stijlen in BibTeX vereist het maken of wijzigen van `.bst`-bestanden, die een gespecialiseerde taal bevatten die complex kan zijn voor nieuwe gebruikers (Patashnik, 1988).
2. **Unicode Ondersteuning:** BibTeX heeft beperkte ondersteuning voor Unicode, wat een beperking kan zijn voor documenten die uitgebreid gebruik maken van niet-ASCII-karakters (Patashnik, 1988).
3. **Statisch Sorteren en Formatteren:** Wijzigingen in sorteren en formatteren vereisen aanpassingen aan de `.bst`-bestanden, wat niet eenvoudig is voor dynamische of complexe sorteerbehoeften (Patashnik, 1988).

### 2.3.3. Overzicht van BibLaTeX

BibLaTeX, oorspronkelijk ontwikkeld door Philipp Lehman en later verder door Kime e.a. (2024), is een modernere en flexibeler alternatief voor BibTeX. Het is ontworpen om samen te werken met de `biber`-backend en biedt uitgebreide aanpassingsmogelijkheden en geavanceerde functies voor bibliografiebeheer.

### Belangrijkste Kenmerken van BibLaTeX

1. **Verbeterde Aanpassing:** BibLaTeX maakt uitgebreide aanpassing van bibliografieën mogelijk via  $\LaTeX$ -macro's. Gebruikers kunnen aangepaste stijlen en citatiecommando's definiëren zonder een aparte programmeertaal te moeten leren (Kime e.a., 2024).
2. **Unicode Ondersteuning:** BibLaTeX ondersteunt volledig Unicode, waardoor het geschikt is voor documenten die een breed scala aan karakters en scripts bevatten (Kime e.a., 2024).
3. **Dynamisch Sorteren en Filteren:** BibLaTeX biedt krachtige sorteer- en filteropties, waardoor gebruikers vermeldingen dynamisch kunnen sorteren en groeperen op basis van verschillende criteria zoals auteur, jaar en type (Kime e.a., 2024).
4. **Meertalige Ondersteuning:** BibLaTeX kan samenwerken met `babel`- en `polyglossia`-pakketten om meertalige documenten te ondersteunen. Het past automatisch bibliografie- en citatiestijlen aan op basis van de taalinstellingen van het document (Kime e.a., 2024).

5. **Gesegmenteerde Bibliografieën:** Gebruikers kunnen gesegmenteerde bibliografieën en meerdere bibliografieën binnen een enkel document maken, gecategoriseerd op onderwerpen of secties (Kime e.a., 2024).
6. **Aanpassing van het Datamodel:** Het datamodel in BibLaTeX kan worden uitgebreid en aangepast, waardoor gebruikers nieuwe vermeldingssoorten en velden kunnen definiëren indien nodig (Kime e.a., 2024).
7. **Annotatie en Annotatiecommando's:** BibLaTeX ondersteunt data-annotatie en querycommando's, waardoor meer geavanceerd beheer van bibliografische gegevens mogelijk is (Kime e.a., 2024).

### Beperkingen van BibLaTeX

1. **Leercurve:** De uitgebreide aanpassingsmogelijkheden en flexibiliteit van BibLaTeX brengen een steilere leercurve met zich mee in vergelijking met BibTeX (Kime e.a., 2024).
2. **Compatibiliteitsproblemen:** BibLaTeX is niet volledig compatibel met sommige oudere  $\text{\LaTeX}$ -pakketten die specifiek voor BibTeX zijn ontworpen. Gebruikers moeten ervoor zorgen dat hun documentopstelling compatibel is met BibLaTeX (Kime e.a., 2024).
3. **Afhankelijkheid van Biber:** BibLaTeX is afhankelijk van de `biber`-backend voor het verwerken van bibliografiebestanden, wat een extra afhankelijkheid met zich meebrengt en installatie- en configuratie-inspanningen kan vereisen (Kime e.a., 2024).

### 2.3.4. Gedetailleerde Vergelijking

#### Database-vermeldingssoorten

BibTeX en BibLaTeX ondersteunen verschillende soorten vermeldingen voor verschillende soorten referenties. BibLaTeX biedt echter meer flexibiliteit en extra vermeldingssoorten die niet beschikbaar zijn in BibTeX.

- **BibTeX:** Standaard vermeldingssoorten zijn onder andere `article`, `book`, `inbook`, `incollection`, `inproceedings`, `manual`, `mastersthesis`, `misc`, `phdthesis`, `proceedings`, `techreport`, en `unpublished` (Patashnik, 1988).
- **BibLaTeX:** Ondersteunt alle BibTeX-vermeldingssoorten en extra zoals `mvbook` (meerdelige boeken), `mvcollection` (meerdelige verzamelingen), `dataset`, `online`, `patent`, en meer. Elke vermeldingssoort kan een rijk stel velden hebben die zijn afgestemd op specifieke referentiebehoeften (Kime e.a., 2024).

#### Citatie- en Bibliografiecommando's

BibLaTeX biedt een breed scala aan citatiecommando's en bibliografiebeheeropties die de mogelijkheden van BibTeX overtreffen.

- **BibTeX:** Basis citatiecommando's zijn `\cite`, `\nocite`, `\bibliography`, en `\bibliographystyle` (Patashnik, 1988).
- **BibLaTeX:** Biedt uitgebreide citatiecommando's zoals `\cite`, `\parencite`, `\footcite`, `\textcite`, en meer. Bibliografiecommando's bieden fijnere controle over het sorteren, filteren en formatteren van vermeldingen (Kime e.a., 2024).

### Aanpassing en Stijldefinitie

BibTeX en BibLaTeX verschillen aanzienlijk in hun benadering van aanpassing en stijldefinitie.

- **BibTeX:** Aanpassing vereist het wijzigen van `.bst`-bestanden, wat een gespecialiseerde en minder intuïtieve taal inhoudt (Patashnik, 1988).
- **BibLaTeX:** Aanpassing gebeurt met behulp van  $\LaTeX$ -macro's, waardoor het toegankelijker is voor gebruikers die bekend zijn met  $\LaTeX$ . Gebruikers kunnen eenvoudig citatie- en bibliografiestijlen definiëren en aanpassen binnen hun  $\LaTeX$ -documenten (Kime e.a., 2024).

### Compatibiliteit en Integratie

BibTeX is meer gevestigd en heeft daarom een bredere compatibiliteit met oudere  $\LaTeX$ -pakketten, terwijl BibLaTeX moderne functies biedt maar compatibiliteitsproblemen kan ondervinden met sommige oudere pakketten.

- **BibTeX:** Compatibel met de meeste oudere  $\LaTeX$ -pakketten en stijlen (Patashnik, 1988).
- **BibLaTeX:** Vereist specifieke versies van  $\LaTeX$  en `biber`, en is mogelijk niet compatibel met alle pakketten die voor BibTeX zijn ontworpen (Kime e.a., 2024).

### 2.3.5. Conclusie

Zowel BibTeX als BibLaTeX spelen een belangrijke rol in het beheer van bibliografieën binnen  $\LaTeX$ -documenten. BibTeX is geschikt voor gebruikers die een eenvoudige, gevestigde oplossing nodig hebben met brede compatibiliteit. BibLaTeX daarentegen is ideaal voor degenen die geavanceerde functies, uitgebreide aanpassing en moderne mogelijkheden zoals Unicode-ondersteuning en dynamisch bibliografiebeheer nodig hebben. Gebruikers moeten het hulpmiddel kiezen dat het beste aansluit bij hun technische vereisten en document specificaties. Bij HO-GENT is het gebruik van BibLaTeX verplicht. Deze **Proof of Concept** is daarom bijzonder relevant, aangezien het kan bijdragen aan het voldoen aan de behoeften van zowel studenten als docenten door te zorgen dat de gebruikte BibLaTeX-bestanden optimaal georganiseerd en correct zijn.

## 2.4. Huidige stand van zaken

Op het ogenblik van het schrijven zijn er nog geen *optimale* BibLaTeX-Linters beschikbaar en de linters van de voorganger BibTeX zijn niet compatibel met BibLaTeX. De enige beschikbare linter voor BibLaTeX op dit moment staat op een GitHub-repository van Pez Cuckow<sup>1</sup>. Deze hoort functioneel te zijn, maar lijkt niet optimaal wat de code betreft. Er is dus duidelijk mogelijkheid tot verbetering. De BibLaTeX-Linter van Pez Cuckow is geschreven in Python en heeft een webinterface. Naast het feit dat deze lastig werkend te krijgen was, werkt deze ook zeker niet zonder fouten. Het is mogelijk om deze effectief uit te testen, maar bij het testen werd er gemerkt dat er fouten optraden die niet direct op te lossen waren, zie Figuur 2.1. Het was niet mogelijk om zomaar elk .bib-bestand te gebruiken bij deze checker, wat het niet gunstig maakt om te gebruiken. Wel was het interessant om te kijken hoe Cuckow bepaalde aspecten interpreteerde en uitvoerde. Dit was uiteindelijk ook een bron waaruit inspiratie gehaald kon worden.

```

UnboundLocalError at /validate
local variable 'lastLine' referenced before assignment

Request Method: POST
Request URL: https://biblatex-linter.onrender.com/validate
Django Version: 3.1.12
Exception Type: UnboundLocalError
Exception Value: local variable 'lastLine' referenced before assignment
Exception Location: /opt/render/project/src/linting/views.py, line 134, in validate
Python Executable: /opt/render/project/src/.venv/bin/python
Python Version: 3.7.10
Python Path:
  /opt/render/project/src/
  /opt/render/project/src/.venv/bin
  /usr/local/lib/python3.7
  /usr/local/lib/python3.7
  /usr/local/lib/python3.7/lib-dynload
  /opt/render/project/src/.venv/lib/python3.7/site-packages

Server time: Fri, 01 Mar 2024 11:20:10 +0000

Traceback [switch back to interactive view]
'django.contrib.staticfiles',
  'linting']
Installed Middleware:
(['whitenoise.middleware.WhiteNoiseMiddleware',
'django.middleware.security.SecurityMiddleware',
'django.contrib.sessions.middleware.SessionMiddleware',
'django.middleware.common.CommonMiddleware',
'django.middleware.csrf.CsrfViewMiddleware',
'django.contrib.auth.middleware.AuthenticationMiddleware',
'django.contrib.messages.middleware.MessageMiddleware',
'django.middleware.clickjacking.XFrameOptionsMiddleware'])

Traceback (most recent call last):
  File "/opt/render/project/src/.venv/lib/python3.7/site-packages/django/core/handlers/exception.py", line 47, in inner
    response = get_response(request)
  File "/opt/render/project/src/.venv/lib/python3.7/site-packages/django/core/handlers/base.py", line 181, in _get_response
    response = wrapped_callback(request, *callback_args, **callback_kwargs)
  File "/opt/render/project/src/linting/views.py", line 134, in validate
    if lastLine == lineNumber - 1:
Exception Type: UnboundLocalError at /validate
Exception Value: local variable 'lastLine' referenced before assignment

```

**Figuur (2.1)**

Lintingerror bij het valideren van een BibLaTeX-bestand, gebruikmakende van de linter geschreven door Pez Cuckow<sup>1</sup>.

Daarnaast leek de aanpak van de geschreven code ook niet optimaal te zijn op vlak van leesbaarheid en uitbreidbaarheid. Gezien dit echter wel criteria zijn waaraan de **Proof of Concept** moest voldoen, werd er besloten om deze linter niet verder te onderzoeken. Er werd wel gekeken naar de functionaliteiten die deze linter aanbood en andere zaken die handig leken om over te nemen in de eigen implementatie.

## 2.5. Andere linters

Gezien het wat BibLaTeX-linters betreft zeer beperkt is, werd er op zoek gegaan naar andere soorten linters. Bijvoorbeeld een linter voor JavaScript, Python of andere programmeertalen. Er werd gekeken naar het soort features dat de linters

<sup>1</sup><https://github.com/Pezmc/BibLaTeX-Linter>

aanbieden, hun performantie, alsook hun broncode indien deze toegankelijk was. Enkele linters die bekeken werden zijn: JSHint, Stylelint, Ruff, PyType en bibl. Net als de [Proof of Concept](#) waren al de bekeken linters gratis te gebruiken en open source.

Daaruit bleek dat naast de manier van implementeren, de taal waaruit de linter opgebouwd is ook van belang is voor de performantie. Dit was het begin naar een onderzoek voor de ideale programmeertaal te vinden voor deze [Proof of Concept](#).

### 2.5.1. Ruff

Ruff<sup>2</sup> is een Python linter ontwikkeld door Astral (2024), geschreven in de programmeertaal Rust. Dit onderscheidt Ruff van andere Python linters, die doorgaans in Python zelf zijn geschreven. Rust staat bekend om zijn snelheid en veiligheid, wat het een uitstekende keuze maakt voor een linter. Bovendien is Rust lichter om op hardware te draaien dan Python, aangezien Python een interpretatieve taal is en Rust een gecompileerde taal. Dit maakt Ruff bijzonder geschikt voor gebruik in CI-pipelines.

Een ander belangrijk voordeel van Rust is de stabiliteit van de taal. Er is slechts één versie van Rust en die wordt continu doorontwikkeld, wat ervoor zorgt dat code die vandaag gecompileerd wordt, ook over tien jaar nog gecompileerd kan worden. Dit biedt een aanzienlijke zekerheid voor de toekomstbestendigheid van software.

Ruff kan op bepaalde taken tien tot wel honderd keer sneller zijn dan zijn concurrenten. Dit wekt vanzelfsprekend de interesse om de prestaties van de Rust-taal te testen. Op de website van Astral, de makers van Ruff, is een vergelijking te vinden tussen Ruff en andere Python linters. Ook Turner-Trauring (2023) bevestigen in hun artikel dat de snelheid van Ruff duidelijk merkbaar is en onderbouwen dit met testresultaten. Deze prestaties zijn vooral voordelig in pipelines op virtuele machines, aangezien vCPU's (virtuele processor units) meestal trager zijn dan de verwerkingsunits in moderne laptops of desktops.

### 2.5.2. DirtyRat

DirtyRat is een JavaScript linter die werd ontdekt tijdens het onderzoek naar hoe een linter gemaakt kon worden. Het is een linter die zelf in JavaScript geschreven is en waarvan de stapsgewijze opbouw te vinden is in het artikel van Borges Late (2021). Daarnaast is ook de broncode zelf te vinden op GitHub<sup>3</sup>. Dit was ook de reden waarom er besloten werd om JavaScript als kandidaat-programmeertaal te nemen.

Hoewel het artikel zeer in detail gaat en het grondig onderzocht werd, werd er vastgesteld dat het toch wat te uitgebreid is voor deze [Proof of Concept](#). JavaScript en andere programmeertalen hebben veel meer complexe structuren dan een BibLaTeX-

---

<sup>2</sup><https://astral.sh/ruff>

<sup>3</sup><https://github.com/JoanaBLate/dirtyrat>



bestand. Dus hoewel het interessant was om te zien hoe een linter in JavaScript gemaakt kon worden, was het niet nodig om elke component over te nemen.

### 2.5.3. bibl

`bibl`<sup>4</sup> is een linter voor BibTeX, de voorloper van BibLaTeX, geschreven in Python door Arne Van Den Kerchove. Het onderzoeken van een linter voor de voorloper van BibLaTeX leek bijzonder interessant om een goed inzicht te krijgen in wat precies verwacht kan worden van een linter. `bibl` biedt een uitgebreide lijst van regels, evenals een gedetailleerde projectstructuur en implementatiewijze van zowel de lintercode als de bijbehorende tests.

`bibl` werd ontdekt net voordat er begonnen werd met de ontwikkeling van een eigen linter. Na een grondige analyse bleek dat `bibl` een goed opgebouwde, modulaire en efficiënte open-source linter is. `bibl` voldeed aan vrijwel alle criteria die voor de *Proof of Concept* linter waren opgesteld, wat leidde tot een nieuwe visie.

Het maken van een linter zoals `bibl`, maar dan voor BibLaTeX in plaats van BibTeX, leek een haalbaar einddoel. Echter, vanwege een achterstand op de planning, zou het niet mogelijk zijn om een eigen linter even uitgebreid uit te werken. Hoewel het concept van een *Proof of Concept* wellicht bereikt kon worden, leek het voordeliger om `bibl` te proberen gebruiken en compatibel te maken met BibLaTeX.

Gezien de verschillen tussen BibTeX en BibLaTeX, was er enige twijfel over de haalbaarheid hiervan, vooral omdat `bibl` gebruik maakt van `pybtex` als parser. Op de site van `pybtex`<sup>5</sup> staat het volgende vermeld:

*Pybtex is a BibTeX-compatible bibliography processor written in Python.*

Dit impliceert dat `pybtex` verantwoordelijk is voor het extraheren van gegevens uit bibliografiebestanden. Hoewel de quote niet expliciet vermeldt dat het alleen voor BibTeX-bestanden werkt, wordt compatibiliteit met BibLaTeX nergens expliciet genoemd. Daarom was het noodzakelijk om dit zelf te testen.

Voor de test werd een BibLaTeX-document gebruikt dat door de promotor was gedeeld. Dit document bevatte geldige bronvermeldingen en was daarmee perfect geschikt om te onderzoeken of `bibl` als basis gebruikt kon worden. `bibl` werd snel geïnstalleerd via de Python package manager, `pip`. Vervolgens werd het lint-commando uitgevoerd op het BibLaTeX-document en werd bekeken wat `bibl` rapporteerde.

`bibl` crashte niet en genereerde een uitgebreide lijst opmerkingen; dit was een positief resultaat. Er waren echter enkele bestaande regels die niet compatibel waren met BibLaTeX-bestanden.

Name
📁 .idea
📁 bibl
📁 test
📁 test_data
📄 .flake8
🔖 .gitignore
🐱 .gitlab-ci.yml
📄 LICENSE
📄 MANIFEST.in
📄 README.md
🐍 setup.py

**Figuur (2.2)**

Repository overzicht bibl.

### Algemene structuur

De gitlab-repository van bibl<sup>6</sup> is een goed voorbeeld van hoe een goed gestructureerd project eruit kan zien. Naast de bibl folder, die alle broncode van bibl zelf bevat, zijn er ook: testen, test-data, lintervoorkeuren, een pipeline (zie 2.10), een licentie, een duidelijke readme en een installatie-script aanwezig. Ook de .idea, .gitignore en MANIFEST.in zijn niet onbelangrijk. de .idea bevat de voorkeursinstellingen voor de IDE (Integrated development environment, ook gekend als werkomgeving) van de auteur, het .gitignore bestand bevat een opsomming van bestanden of soort bestanden die niet op de repository dienen te komen. Denk hier aan tijdelijke bestanden die vanzelf aangemaakt worden; deze zijn niet nodig op de repository omdat ze onnodig plaats gebruiken. Tot slot wordt het MANIFEST.in bestand in Py-

<sup>4</sup><https://gitlab.com/arnevdk/bibl>

<sup>5</sup><https://pybtex.org/>

<sup>6</sup><https://gitlab.com/arnevdk/bibl>

thon gebruikt om aan te geven welke bestanden moeten worden opgenomen bij het maken van een distributiekpakket; wat nodig is om het beschikbaar te stellen bij de pakketbeheerder<sup>7</sup> (pip). In een latere sectie (2.8) wordt er dieper ingegaan op de volledige werking van de linter.

## 2.6. Programmeertaal

Om de gepaste programmeertaal te vinden, werden diverse kandidaat-programmeertalen overwogen. De kandidaat-programmeertalen waren: Rust, JavaScript en Python. Door het analyseren van reeds bestaande linters en het afdraan van een hele lijst aan programmeertalen, werd er besloten om met deze drie programmeertalen te experimenteren.

In een latere fase (sectie 4.2) werden er beperkte prototypes uitgewerkt in elk van deze talen; deze werden met elkaar vergeleken om te bepalen welke taal het meest geschikt zou zijn. Alle opties zijn cross-platform, wat belangrijk is gezien de linter bruikbaar dient te zijn voor iedereen. Alsook dient het een programmeertaal te zijn die gekend is, hiermee wordt er bedoeld dat het een programmeertaal moet zijn die al matuur is en door een groot aantal mensen gekend is.

### 2.6.1. Rust

Gezien de performantie van Ruff, leek Rust een gepaste optie om de linter in te maken. Ook de stijgende populariteit van de taal maakt dit een aantrekkelijke optie. Ondanks dat dit een onbekende taal is, leek het zeker wel een interessante taal om te leren. De haalbaarheid van de *Proof of Concept* kwam echter meer in gevaar door deze keuze. Hoewel de populariteit stijgt, blijft het wel nog een taal die niet door iedereen gekend is, in tegenstelling tot bijvoorbeeld JavaScript of Python. Een ander groot pluspunt aan Rust is dat code die vandaag compiled, ook nog binnen vijf of tien jaar zal kunnen compilen. Dit omdat de makers ervoor streven dat het een super compatibele programmeertaal is. Wat men niet met volle zekerheid van bijvoorbeeld Python<sup>8</sup> zou kunnen zeggen.

Dankzij Klabnik en Nichols (2022) kan er geleerd worden dat Rust een nadruk legt op typeveiligheid (typesafety) en geheugenveiligheid. Wanneer een taal of systeem geheugenveilig is, betekent dit dat het ontworpen is om toegangsfouten zoals bufferoverlopen, dangling pointers (verwijzingen naar vrijgegeven geheugen), en dubbele bevrijdingen van geheugen (double frees) te voorkomen. Deze soorten fouten kunnen leiden tot onvoorspelbaar gedrag, crashes, en beveiligingslekken zoals het uitvoeren van willekeurige code of informatie-lekken. Het is daarom een goede eigenschap om te hebben.

---

<sup>7</sup><https://pypi.org/>

<sup>8</sup><https://devguide.python.org/versions/>

### 2.6.2. Python

Python was misschien significant (tot meer dan tien keer) trager in vergelijking met Rust in de test die gevonden werd, maar het blijft wel een taal waar iedereen gemakkelijk mee kan beginnen te werken. Voor het gemak van uitbreidbaarheid is dit dan weer een interessante optie om deze taal in optie te nemen. Zoals eerder vermeld door Turner-Trauring (2023) zou de performantie vooral bij de pipeline te merken zijn. De echte vraag blijft momenteel nog altijd in hoeverre het minder performant zijn een grote zorg zal worden voor deze [Proof of Concept](#).

### 2.6.3. JavaScript

Een zeer gekende scripting taal dat cross-platform gebruikt kan worden. Hoewel het eerder gericht is voor het gebruik in websites of [GUI](#) gerichte scripts, is het ook mogelijk om er cli-tools mee te maken. De performantie werd onderzocht. JavaScript is ook een dynamisch typerende taal, wat in theorie voor problemen zou kunnen zorgen in sommige use-cases, al leek dit geen zorg te zijn voor deze [Proof of Concept](#)(Simpson, 2023).

## 2.7. Opbouw linter

Aan het begin van dit project werd de beslissing genomen om direct met de ontwikkeling van een prototype aan de slag te gaan, wat leidde tot de zoektocht naar geschikte bronnen. Deze beslissing werd genomen gezien de beperkte tijd en de hoeveelheid aan onderzoek die diende te gebeuren. Indien Rust de taal zou moeten worden, moest deze nog volledig aangeleerd worden en zou er immers geen kostbare tijd verloren mogen gaan.

In dit proces werden de artikelen van Borges Late (2021), die een stapsgewijze handleiding bieden voor het bouwen van een JavaScript-linter ([DirtyRat](#)<sup>9</sup>), als bijzonder nuttig beschouwd. Ondanks dat de lectuur van deze serie aanvankelijk de indruk wekte dat het project complexer zou zijn dan verwacht, veranderde dit vermoeden na verder onderzoek naar de structuur van .bib-bestanden. Het werd duidelijk dat de ontwikkeling van een linter voor BibLaTeX minder complex is dan voor programmeertalen. Dit is te wijten aan het feit dat programmeertalen loops, conditionele statements en een scala aan andere complexe elementen bevatten, terwijl BibLaTeX gekenmerkt wordt door een consistente en gestandaardiseerde structuur, waardoor de noodzaak voor het opleggen van regels aanzienlijk vermindert. Het was echter wel zeer nuttig om alle componenten te zien die aan bod kunnen komen bij het bouwen van een linter en om deze in gedachte te houden voor de effectieve uitwerking van de BibLaTeX-linter [Proof of Concept](#).

Om een idee te krijgen van een algemene structuur, werd er gekeken naar zowel [DirtyRat](#), gemaakt door Borges Late (2021) en naar [Flake8](#)<sup>10</sup>. Dit onderzoek liet blij-

---

<sup>9</sup><https://github.com/JoanaBLate/dirtyrat>

<sup>10</sup><https://github.com/PyCQA/flake8>








ken dat een algemene structuur van een linter bestaat uit verschillende essentiële componenten die gezamenlijk zorgen voor een effectieve werking en uitbreidbaarheid van de tool. Deze componenten omvatten:

- **Kernfunctionaliteitsbestanden:** Deze bestanden bevatten de hoofdlogica en de hoofdingangen van de linter, zoals `main.js` en `rat.js` in DirtyRat, en `__main__.py` en de `main/` map in Flake8.
- **Parsen en Tokenisatie:** Deze modules zijn verantwoordelijk voor het lezen, parsen en tokeniseren van de code die geanalyseerd moet worden. Voorbeelden hiervan zijn `parser.js` en `tokenizer.js` in DirtyRat, en `processor.py` en `checker.py` in Flake8.
- **Helper- en Hulpprogramma's:** Hulpprogramma's ondersteunen diverse taken binnen de linter. `helper.js` in DirtyRat en `utils.py` in Flake8 zijn hier voorbeelden van.
- **Regeldefinities en Controles:** Deze bestanden definiëren de specifieke linting regels en de logica om de code aan deze regels te toetsen. Voorbeelden zijn `check-names.js` en `expression.js` in DirtyRat, en `style_guide.py` en `violation.py` in Flake8.
- **Configuratie en Standaarden:** Bestanden zoals `defaults.py` en de `options/` map in Flake8 bevatten instellingen en standaardwaarden voor de werking van de linter. DirtyRat kan ook vergelijkbare configuratiebestanden hebben, hoewel ze niet direct zichtbaar zijn in de screenshot.
- **Documentatie:** Documentatiebestanden zoals `README.md` in DirtyRat bieden informatie over de linter en het gebruik ervan. Flake8 heeft mogelijk ook documentatiebestanden die niet zichtbaar zijn in de screenshot.
- **Plugin- en Extensieondersteuning:** Ondersteuning voor plugins maakt het mogelijk om de functionaliteit van de linter uit te breiden. Flake8 bevat bijvoorbeeld een `plugins/` map voor dit doel.

Deze componenten zorgen ervoor dat linters modulair, onderhoudbaar en uitbreidbaar zijn, waardoor ze effectief code kunnen analyseren en codestandaarden kunnen handhaven voor hun respectieve programmeertalen. Het zijn dus ook belangrijke componenten voor de eigen [Proof of Concept](#).

## 2.8. bibl pakket - Diepere blik

De structuur van bibl 2.3 volgt een gebruikelijke pakket-gebaseerde aanpak voor het organiseren van code. Deze aanpak wordt vaak toegepast wanneer een project is opgezet als een herbruikbaar pakket, in plaats van een los script. De aanwe-

Name
..
rules
 <code>__init__.py</code>
 <code>__main__.py</code>
 <code>bibl.yml</code>
 <code>config.py</code>
 <code>lint.py</code>
 <code>rule.py</code>
 <code>text_utils.py</code>

**Figuur (2.3)**

Repository overzicht bibl broncode.

zigheid van `__init__.py` zorgt er namelijk voor dat het pakket importeerbaar is (Loubser, 2021). Dit is direct ook een indicator dat het om een pakket gaat.

Om bibl te begrijpen, wordt de broncode van bibl zelf geanalyseerd. Gezien bibl uiteindelijk de basis kon worden van deze [Proof of Concept](#), was het van groot belang om een zorgvuldige analyse uit te voeren en de codebase zo goed mogelijk te begrijpen.

### 2.8.1. `__init__.py`

Zoals eerder vermeld, is dit bestand kenmerkend bij pakketten. Het bestand is namelijk vereist voor Python om de directory als een pakket te behandelen. Het kan leeg zijn, maar het kan ook initialisatiecode of pakket-niveau-definities bevatten; in dit geval bevat het de huidige softwareversie.

### 2.8.2. `__main__.py`

Dit bestand wordt gebruikt wanneer het pakket als een script wordt uitgevoerd (bijvoorbeeld `python -m bibl`). Het bevat de implementatie van de [Command Line Interface \(CLI\)](#) voor het linten van bibliografische bestanden in BibTeX-formaat. Het

script, hiervoor als referentie bestand, maakt gebruik van de Click-bibliotheek om een gebruiksvriendelijke interface te bieden voor het controleren van bibliografische referenties op consistentie en correctheid. Hieronder volgt een overzicht van de verschillende onderdelen binnen dit bestand.

### Imports en Configuratie

Het script begint met het importeren van noodzakelijke modules en functies, waaronder Click voor het CLI-framework, en functies uit de bibl-module voor linting, configuratie en regels:

```
1 import os
2 import warnings
3 import click
4 from bibl import __version__
5 from bibl.lint import lint as bibl_lint
6 from bibl.config import load_config_file, set_config
7 from bibl.rule import load_rules
8 from bibl.text_utils import format_rules_markdown_tables
```

**Listing 2.8.1:** bibl: \_\_main\_\_.py - Imports.

### CLI Basisgroep

Een CLI basisgroep functioneert als een container, of groepering, voor meerdere gerelateerde command-line commando's. Hiermee wordt het mogelijk gemaakt om een hoofdcommando te definiëren waaraan subcommando's kunnen worden toegevoegd. De basisgroep wordt gedefinieerd met behulp van de `@click.group()` decorator. Deze groep fungeert als een container voor subcommando's die specifieke taken uitvoeren. De subcommando's kunnen tegelijkertijd ook als parameter beschouwd worden. Opties die van toepassing zijn op de gehele groep kunnen hier ook worden gespecificeerd. Een voorbeeld van de definitie van een basisgroep in het script is als volgt:

```
1 @click.group()
2 @click.option('-c', '--config', help='Custom configuration file
   path.', type=str)
3 @click.option('--select', help='Comma separated list of enabled
   rules, all other rules will be disabled.', type=str)
4 @click.option('--ignore', help='Comma separated list of disabled
   rules, all other rules will be enabled.', type=str)
5 @click.option('--indent-spaces', help='Number of trailing
   whitespaces for indented line, used by T01.', type=int)
6 @click.option('--max-line-length', help='Max line length before wrap
   recommended, used by T03.', type=int)
```

```

7 def cli(config, select, ignore, indent_spaces, max_line_length):
8     if config is not None:
9         load_config_file(config)
10    elif os.path.isfile('bibl.yml'):
11        load_config_file('bibl.yml')
12    elif os.path.isfile('.bibl.yml'):
13        load_config_file('.bibl.yml')
14    if select is not None:
15        set_config('select', select.split(','))
16    if ignore is not None:
17        set_config('ignore', ignore.split(','))
18    set_config('indent_spaces', indent_spaces)
19    set_config('max_line_length', max_line_length)

```

**Listing 2.8.2:** bibl: \_\_main\_\_.py - CLI parameters.

### Lint Commando

Het lint commando lint één of meer opgegeven BibTeX-bestanden:

```

1 @cli.command(help="Lint a BibTeX bibliography file.")
2 @click.argument('bibliography', type=str, nargs=-1)
3 def lint(bibliography):
4     warnings.filterwarnings("ignore")
5     for bib in bibliography:
6         bibl_lint(bib)

```

**Listing 2.8.3:** bibl: \_\_main\_\_.py - Lint commando.

### Lijst Commando's

Er zijn twee commando's om de lintingregels weer te geven: `list_all` toont alle beschikbare regels, terwijl `list_enabled` alleen de ingeschakelde regels toont.

```

1 @cli.command(help="Show all available rules.")
2 @click.option('-m', 'markdown', help='Format rules as markdown
3     table.', is_flag=True)
4 def list_all(markdown):
5     rules = load_rules().all
6     if markdown:
7         click.echo(format_rules_markdown_tables(rules))
8     else:
9         for rule in rules:
10            click.echo(rule)

```



```

10
11 @cli.command(help="Show all rules enabled by the configuration.")
12 @click.option('-m', 'markdown', help='Format rules as markdown
    table.', is_flag=True)
13 def list_enabled(markdown):
14     rules = load_rules().enabled
15     if markdown:
16         click.echo(format_rules_markdown_tables(rules))
17     else:
18         for rule in rules:
19             click.echo(rule)

```

**Listing 2.8.4:** bibl: \_\_main\_\_.py - Commando om alle regels te tonen.

### Versie Commando

Het version commando toont de huidige versie van het bibl-pakket:

```

1 @cli.command(help="Show the package version.")
2 def version():
3     click.echo('bibl version: ' + __version__)

```

**Listing 2.8.5:** bibl: \_\_main\_\_.py - Toon de huidige versie van het pakket.

### Hoofdprogramma

Het hoofdprogramma start de CLI met `cli(prog_name='bibl')`:

```

1 if __name__ == '__main__':
2     cli(prog_name='bibl')

```

**Listing 2.8.6:** bibl: \_\_main\_\_.py - Vanuit CLI kan bibl rechtstreeks aangeroepen worden.

### 2.8.3. bibl.yml - Configuratiebestand

Het volgende configuratiebestand is een YAML (.yml) bestand dat de standaardinstellingen voor de bibliografische lintingtool definieert. Dit bestand specificeert welke regels ingeschakeld of uitgeschakeld moeten worden, evenals andere configuratie-opties zoals de maximale regellengte en het aantal inspringende spaties.

```

1 ## DEFAULT BIBL CONFIG
2 select: [ ] # List of enabled rules, all other rules will be disabled
3 ignore: [ ] # List of disabled rules, all other rules will be enabled
4 # Specify max 1 of the two options above. If none are specified, all
    rules will be enabled

```

```
5 indent_spaces: 4 # number of trailing whitespaces for indented line,
   used by T01
6 max_line_length: 120 # max line length before wrap recommended,
   used by T03
7 abbreviation_dot: True # abbreviate middle names with dot (John F.
   Kennedy) as opposed to without (John F Kennedy), used by E02
8
9 # Specification from https://en.wikipedia.org/wiki/BibTeX extended
   with fields used in JabRef.
10 # This specification is used to generate M01 and U01 rules
11 type_spec:
12   article: # An article from a journal or magazine.
13     required: [ title, journal, year, volume ]
14     optional: [ number, pages, month, doi, key, file ]
15   book: # A book with an explicit publisher.
16     required: [ title, publisher, year ]
17     optional: [ volume, number, series, address, edition, month,
18               key, url, file, isbn ]
18 # NOTE: Overige entry types weggelaten wegens plaats!
```

**Listing 2.8.7:** bibl: bibl.yml - Deel code uit het hoofdconfiguratiebestand. Geselecteerde en te negeren regels kunnen hier toegekend worden. Alsook de standaard aantal spaties ter indentatie. Ook kan er gekozen worden of middelnamen afgekort mogen worden met een '.' of niet. Daaronder volgt een lijst van entry types met zowel de verplichte als optionele velden.

### Beschrijving van de Configuratieopties

De YAML-configuratie begint met het definiëren van een aantal algemene opties voor de linter:

- `select`: Een lijst van ingeschakelde regels. Indien gespecificeerd, worden alle andere regels uitgeschakeld.
- `ignore`: Een lijst van uitgeschakelde regels. Indien gespecificeerd, worden alle andere regels ingeschakeld.
- `indent_spaces`: Het aantal inspringende spaties voor ingesprongen regels, gebruikt door de regel T01.
- `max_line_length`: De maximale regellengte voordat een regel wordt afgebroken, aanbevolen door de regel T03.
- `abbreviation_dot`: Bepaalt of middelste namen met een punt worden afgekort (bijvoorbeeld John F. Kennedy) of zonder punt (bijvoorbeeld John F Kennedy), gebruikt door de regel E02.

### Type Specificaties

Daarnaast bevat het configuratiebestand een specificatie van verschillende typen BibTeX-entries en hun vereiste en optionele velden. Deze specificatie is gebaseerd op de standaard BibTeX-specificatie en is uitgebreid met velden die worden gebruikt in JabRef. Deze specificatie wordt gebruikt om de regels M01 en U01 te genereren. Hieronder zijn er enkelen opgelijst, maar let op, het zijn ze niet allemaal! Raadpleeg de gehele lijst op de GitLab-repository van bibl<sup>11</sup>

- `article`: Een artikel uit een tijdschrift of magazine. Vereist velden zoals `title`, `journal`, `year`, en `volume`.
- `book`: Een boek met een expliciete uitgever. Vereist velden zoals `title`, `publisher`, en `year`.
- `booklet`: Een werk dat is gedrukt en gebonden, maar zonder een genoemde uitgever of sponsorende instelling. Vereist het `title` veld.
- ...

Deze configuratie-instellingen en type-specificaties vormen de basis voor het configureren en uitvoeren van de bibliografische lintingtool, waarbij consistentie en correctheid van de bibliografische gegevens worden gewaarborgd.

#### 2.8.4. `config.py` - Configuratielogica

De volgende code implementeert de logica voor het laden en beheren van configuratie-instellingen voor de bibliografische lintingtool. Deze configuratie-instellingen worden gedefinieerd in een YAML-bestand en geladen bij de initialisatie van de linter.

```
1 """Linter configuration logic."""
2 from typing import Dict, Any
3
4 import pkg_resources
5 import yaml
6
7 _config = dict()
8 _read_default = False
9
10 _DEFAULT_CONFIG_FILE = 'bibl.yml'
11
12
13 def get_config() -> Dict:
```

<sup>11</sup><https://gitlab.com/arnevdk/bibl/-/blob/master/bibl/bibl.yml>

```
14     """Return the loaded config or instantiate and return default
15         config."""
16     if not _read_default:
17         _load_default_config()
18     return _config
19
20 def set_config(key: str, value: Any, default: bool = False):
21     """Set a value in the configurations.
22
23     :param key: configuration entry key
24     :param value: configuration entry value
25     :param default: If true, set as default value to be overwritten
26                     by later
27     """
28     if value is not None:
29         if default:
30             _config.setdefault(key, value)
31         else:
32             _config[key] = value
33             _validate_and_clean_config(_config)
34
35 def load_config_file(file):
36     """Read a YAML config file and use it as the configuration.
37
38     :param file: .yaml config file path
39     """
40     with open(file) as config_file:
41         config = yaml.load(config_file, Loader=yaml.FullLoader)
42         for k, v in config.items():
43             set_config(k, v)
44
45
46 def _load_default_config():
47     global _read_default
48     _read_default = True
49     with open(pkg_resources.resource_filename(__name__,
50                                             _DEFAULT_CONFIG_FILE)) as \
51         default_config_file:
```

```
52     default_config = yaml.load(default_config_file,  
53                               Loader=yaml.FullLoader)  
54     for k, v in default_config.items():  
55         set_config(k, v, default=True)  
56  
57 def _validate_and_clean_config(config):  
58     if 'select' in config and 'ignore' in config and  
59         config['select'] and \  
60         config['ignore']:  
61         raise ValueError(  
62             "Configuration cannot contain both included and selected  
63             and "  
64             "ignored rules. Use either include or exclude to select"  
65             "enabled rules."  
66         )
```

**Listing 2.8.8:** bibl: config.py - Bevat de configuratielogica, de standaard configuratie wordt ingesteld en indien er achteraf aangepaste configuraties aanwezig zijn van de gebruiker, worden deze ook ingelezen en overschrijven ze de standaard configuraties.

### Beschrijving van de Functies

- `get_config`: Retourneert de geladen configuratie of instantieert en retourneert de standaardconfiguratie indien deze nog niet is geladen.
- `set_config`: Stelt een waarde in de configuraties in. Indien de parameter `default` waar is, wordt de waarde alleen ingesteld als deze nog niet bestaat. Deze functie valideert en reinigt ook de configuratie.
- `load_config_file`: Leest een YAML-configuratiebestand en gebruikt deze als de huidige configuratie. Elk configuratie-item wordt ingesteld met behulp van de `set_config` functie.
- `_load_default_config`: Laadt de standaardconfiguratie vanuit het standaard configuratiebestand (`bibl.yaml`). Deze functie wordt aangeroepen bij de eerste toegang tot de configuratie.
- `_validate_and_clean_config`: Valideert de configuratie door te controleren of niet zowel de `select` als de `ignore` opties tegelijk zijn gespecificeerd. Indien beide zijn gespecificeerd, wordt een `ValueError` opgeworpen.

### Toepassing en Validatie

De configuratielogica zorgt ervoor dat de linter correct wordt ingesteld volgens de specificaties van de gebruiker, zoals gedefinieerd in een YAML-configuratiebestand.

De standaardconfiguratie wordt geladen bij de eerste toegang tot de configuratie om ervoor te zorgen dat de linter altijd met geldige instellingen werkt. Daarnaast wordt de configuratie gevalideerd om conflicterende instellingen te voorkomen, zoals het gelijktijdig gebruik van `select` en `ignore`.

Deze gestructureerde benadering voor het laden en beheren van configuraties helpt bij het waarborgen van de consistentie en flexibiliteit van de linter, waardoor gebruikers specifieke instellingen kunnen aanpassen aan hun behoeften zonder de integriteit van het systeem in gevaar te brengen.

### 2.8.5. lint.py - Hoofdlogica

De hoofdlogica van `bibl` bevindt zich in `lint.py`. Hier gebeuren de controles van de bibliografische referenties, zowel consistentie als correctheid worden gecontroleerd op basis van de opgestelde regels.

```
1 """Main linter logic."""
2 import logging
3 import sys
4 from dataclasses import dataclass
5 from typing import List, Iterable
6
7 import pybtex
8 from pybtex.database import parse_file
9 from bibl.rule import load_rules, Rule, EntryRule, TextRule
10 from bibl.text_utils import find_entry_line_number, MONTH_NAMES
11
12 logger = logging.getLogger()
13 logger.setLevel(logging.WARNING)
14
15 handler = logging.StreamHandler(sys.stdout)
16 handler.setLevel(logging.WARNING)
17 logger.addHandler(handler)
18
19
20 @dataclass
21 class LintWarning:
22     """Dataclass to represent and report a linter rule violation."""
23
24     # file path of the detected violation
25     file: str
26     # line number of the detected violation
27     line: int
28     # the violated rule
```

```
29     rule: Rule
30
31     def log(self):
32         """Print the warning with details to stdout."""
33         msg = "{}: {} {}".format(self.file, self.line, str(self.rule))
34         logger.warning(msg)
35
36
37 def lint(bibliography: str, verbose: bool = True) ->
38     List[LintWarning]:
39     """Execute the main linter program.
40
41     The linter will first scan the bibliography text file and check
42     all text
43     rules for each line in the text file. Next, the file will be
44     parsed by the
45     pybtex parser and all entry rules will be checked.
46
47     :param bibliography: a .bib bibliography file path
48     :param verbose: log linter warnings to stdout
49     :return: a list of LintWarning objects representing the linter
50     violations
51     found while running
52     """
53     bib_data = parse_file(bibliography, macros=MONTH_NAMES)
54     bib_data.file = bibliography
55     with open(bibliography, 'r') as bib_file:
56         bib_text = bib_file.read()
57
58     rules = load_rules()
59
60     text_warnings = _apply_text_rules(bibliography, bib_text,
61                                     rules.enabled_text_rules)
62     entry_warnings = _apply_entry_rules(bibliography, bib_data,
63                                       bib_text,
64                                       rules.enabled_entry_rules)
65
66     warnings = text_warnings + entry_warnings
67
68     warnings.sort(key=lambda w: w.rule.rule_id)
69     warnings.sort(key=lambda w: w.line)
```

```
65     if verbose:
66         for warning in warnings:
67             warning.log()
68     return warnings
69
70
71 def _apply_text_rules(bibliography: str, bib_text: str,
72                      text_rules: Iterable[TextRule]) ->
73                      List[LintWarning]:
74     """Check all text rules in the bibliography text file.
75
76     :param bibliography: bibliography file path
77     :param bib_text: bibliography file contents
78     :param text_rules: list of text rules to be evaluated on each
79                       line of
80                       the file
81     :return: a list of LinterWarnings representing found rule
82             violations
83     """
84     warnings = []
85     for i, line in enumerate(bib_text.split('\n')):
86         line_number = i + 1
87         for rule in text_rules:
88             result = rule(line_number, line, bib_text)
89             if not result:
90                 warnings.append(LintWarning(bibliography,
91                                             line_number, rule))
92     return warnings
93
94 def _apply_entry_rules(bibliography: str,
95                       bib_data: pybtex.database.BibliographyData,
96                       bib_text: str, entry_rules:
97                       Iterable[EntryRule]) \
98                       -> List[LintWarning]:
99     """Check all entry rules in the bibliography text file.
100
101     :param bibliography: bibliography file path
102     :param bib_data: parsed bibliography data containing entries
103     :param bib_text: bibliography file contents
```



```
100     :param entry_rules: list of entry rules to be evaluated on each
        entry
101     :return: a list of LintWarning representing found rule violations
102     """
103     warnings = []
104     for key, entry in bib_data.entries.items():
105         for rule in entry_rules:
106             line_number, offset = find_entry_line_number(bib_text,
                key)
107             result = rule(key, entry, bib_data)
108             if not result:
109                 warnings.append(LintWarning(bibliography,
                line_number, rule))
110     return warnings
```

**Listing 2.8.9:** bibl: lint.py bevat de hoofdlogica van de linter. Het kan gezien worden als het startpunt van waaruit de regels effectief ingeladen en gecontroleerd zullen worden.

### Beschrijving van de Functies

- **LintWarning:** Een dataclass die een linterwaarschuwing vertegenwoordigt. Deze bevat de volgende attributen:
  - **file:** Het pad naar het bestand waarin de overtreding is gedetecteerd.
  - **line:** Het regelnummer van de gedetecteerde overtreding.
  - **rule:** De overtreden regel.
- **log** methode wordt gebruikt om de waarschuwing met details naar stdout te loggen.
- **lint:** Voert het hoofdprogramma van de linter uit. De functie controleert eerst de bibliografietekst op alle tekstregels voor elke regel in de tekst. Vervolgens wordt het bestand geparseerd met de pybtex parser en worden alle invoerregels gecontroleerd.
  - **bibliography:** Het pad naar een .bib bibliografisch bestand.
  - **verbose:** Logt linterwaarschuwingen naar stdout indien ingesteld op True.
  - Retourneert een lijst van **LintWarning** objecten die de tijdens het uitvoeren gevonden overtredingen vertegenwoordigen.
- **\_apply\_text\_rules:** Controleert alle tekstregels in het bibliografietekstbestand.
  - **bibliography:** Het pad naar het bibliografisch bestand.

- `bib_text`: De inhoud van het bibliografiebestand.
  - `text_rules`: Een lijst van tekstregels die op elke regel van het bestand worden geëvalueerd.
  - Retourneert een lijst van `LintWarning` objecten die gevonden overtredingen van de tekstregels vertegenwoordigen.
- `_apply_entry_rules`: Controleert alle invoerregels in het bibliografietekstbestand.
    - `bibliography`: Het pad naar het bibliografisch bestand.
    - `bib_data`: Geparste bibliografische gegevens die entries bevatten.
    - `bib_text`: De inhoud van het bibliografiebestand.
    - `entry_rules`: Een lijst van invoerregels die op elke entry worden geëvalueerd.
    - Retourneert een lijst van `LintWarning` objecten die gevonden overtredingen van de invoerregels vertegenwoordigen.

### Logica en Validatie

De hoofdlogica van de linter begint met het instellen van de loggingconfiguratie om waarschuwingen naar `stdout` te sturen. Vervolgens worden de regels voor de linter geladen en toegepast op de bibliografische gegevens. Er wordt eerst gekeken naar overtredingen in de tekstregels, gevolgd door overtredingen in de invoerregels. De gevonden overtredingen worden gesorteerd op regel-ID en regelnummers, en indien de `verbose` parameter is ingesteld, worden deze overtredingen gelogd naar `stdout`. Deze aanpak zorgt voor een gestructureerde controle van bibliografische bestanden, waarbij consistentie en correctheid worden gewaarborgd.

#### 2.8.6. `rule.py` - Structuur voor het Beheren van Regels

De volgende code implementeert een structuur voor het beheren van lintingregels op een uniforme manier. Deze structuur maakt het mogelijk om regels te definiëren, registreren en evalueren binnen `bibl`.

#### De Rule Klasse

De `Rule` klasse is een generieke klasse voor een lintingregel. Deze klasse bevat attributen voor het identificeren en beschrijven van de regel, evenals een `callable` om de regel te controleren. Een `callable` is een object dat kan worden aangeroepen als een functie.

```
1 import fnmatch
2 import os
3 from typing import Callable, Dict, List
4 from pybtex.database import Entry
```

```
5 from bibl.config import get_config
6
7 class Rule:
8     """Generic rule class.
9
10    :param rule_id: identifying code for this rule, starting with a
11    letter
12    indicating the rule type, followed by a number or a string
13    specification.
14    :param description: a full sentence description of the rule
15    """
16
17    def __init__(self, rule_id: str, description: str, rule:
18    Callable):
19        """Create Rule object.
20
21    :param rule_id: identifying code for this rule, starting with
22    a letter
23    indicating the rule type, followed by a number or a string
24    specification.
25    :param description: a full sentence description of the rule
26    :param rule: a callable returning a boolean to execute when
27    checking
28    this rule
29    """
30
31    self.rule_id = rule_id
32    self.description = description
33    self._rule = rule
34
35    def __str__(self):
36        """Return rule as string representation."""
37        return "{}: {}".format(self.rule_id, self.description)
38
39    def __call__(self, *args, **kwargs):
40        """Handle Rule objects as callables.
41
42    When calling a rule, it is evaluated over (a part of) the
43    bibliography.
44    :param args, kwargs: Rule arguments
45    :return: True if the bibliography is consistent with the
46    rule, False if
```

```

39         the rule is violated.
40         """
41         return self._rule(*args, **kwargs)
42
43     @property
44     def enabled(self):
45         """Evaluate wheter a rule should be checked this run.
46
47         :return: True if the rule should be checked based on the
48                 configuration
49                 used for running the linter, False otherwise
50         """
51         if get_config()['select']:
52             for pattern in get_config()['select']:
53                 if fnmatch.fnmatch(self.rule_id, pattern):
54                     return True
55             return False
56         if get_config()['ignore']:
57             for pattern in get_config()['ignore']:
58                 if fnmatch.fnmatch(self.rule_id, pattern):
59                     return False
60         return True

```

**Listing 2.8.10:** bibl: rule.py - Bevat de basisstructuur van de regels binnen de linter, ook wordt er gekeken of een specifieke regel al dan niet verwacht wordt om uitgevoerd te worden.

### De EntryRule en TextRule Klassen

De EntryRule en TextRule zijn subklassen van Rule. EntryRule wordt gebruikt om regels te evalueren op geparseerde bibliografie-entries, terwijl TextRule wordt gebruikt om regels te evalueren op tekstregels in bibliografie-entries.

```

1 class EntryRule(Rule):
2     """Rule type evaluating a parsed bibliography entry."""
3
4     def __init__(self, rule_id, description,
5                 rule: Callable[[str, Entry, Dict[str, Entry]],
6                               bool]):
7         """Create EntryRule object.
8
9         :param key: The key of the current bibliography entry
10        :param entry: The current bibliography entry

```

```

10     :param database: All bibliography entries
11     """
12     super().__init__(rule_id, description, rule)
13
14
15 class TextRule(Rule):
16     """Rule type evaluating a text line in the bibliography entry."""
17
18     def __init__(self, rule_id, description,
19                 rule: Callable[[int, str, str], bool]):
20         """Create TextRule object.
21
22         :param line_number: The number of the current line in the
23             bibliography
24         :param line: The content of the current line in the
25             bibliography
26         :param text: The entire bibliography
27         """
28     super().__init__(rule_id, description, rule)

```

**Listing 2.8.11:** bibl: rule.py - Bevat de definitie van EntryRule en TextRule klassen, die respectievelijk een bibliografie entry en een tekstlijn in de bibliografie evalueren.

### De RuleStore Klasse

De RuleStore klasse dient als container voor alle geladen regels. Deze klasse bevat methoden om regels te registreren en op te halen.

```

1 class RuleStore:
2     """Container for all loaded rules."""
3
4     def __init__(self):
5         """Create RuleStore object."""
6         self._rules = []
7
8     def register(self, rule: Rule):
9         """Register a new rule.
10
11         :param: a Rule object to register
12         """
13         position = 0
14         while position < len(self._rules) \
15             and self._rules[position].rule_id < rule.rule_id:

```

```

16         position += 1
17         self._rules.insert(position, rule)
18
19     @property
20     def all(self) -> List[Rule]:
21         """Return all loaded rules."""
22         return self._rules
23
24     @property
25     def enabled(self) -> List[Rule]:
26         """Return all loaded rules enabled by the configuration."""
27         return [rule for rule in self._rules if rule.enabled]
28
29     @property
30     def enabled_entry_rules(self) -> List[EntryRule]:
31         """Return all loaded entry rules."""
32         return [rule for rule in self.enabled if isinstance(rule,
33                     EntryRule)]
34
35     @property
36     def enabled_text_rules(self) -> List[TextRule]:
37         """Return all loaded text rules."""
38         return [rule for rule in self.enabled if isinstance(rule,
39                     TextRule)]

```

**Listing 2.8.12:** bibl: rule.py - Definieert de RuleStore klasse die verantwoordelijk is voor het opslaan, registreren en ophalen van regels, inclusief geactiveerde entry en tekstregels.

### Registratie en Laden van Regels

Door gebruik te maken van het decorator design patroon, zie `register_entry_rule` en `register_text_rule`, kunnen nieuwe regels eenvoudig worden toegevoegd aan de linter. De `load_rules` functie importeert automatisch alle modules in het `bibl/rules` pakket, waardoor alle daarin gedefinieerde regels automatisch worden geladen en geregistreerd.

```

1  _ALL_RULES: RuleStore = RuleStore()
2
3  def register_entry_rule(rule_id, description: str) -> Callable:
4      """Register a function as an entry rule."""
5      def decorator(f: Callable[[str, Entry, Dict[str, Entry]], bool]):
6          rule = EntryRule(rule_id, description, f)
7          _ALL_RULES.register(rule)

```

```
8     return decorator
9
10 def register_text_rule(rule_id: str, description: str) -> Callable:
11     """Register a function as a text rule."""
12     def decorator(f: Callable[[int, str, str], bool]):
13         rule = TextRule(rule_id, description, f)
14         _ALL_RULES.register(rule)
15     return decorator
16
17 def load_rules() -> RuleStore:
18     """Import all modules in the `bibl/rules` package."""
19     for module in os.listdir(os.path.join(os.path.dirname(__file__),
20         'rules')):
21         if module == '__init__.py' or module[-3:] != '.py':
22             continue
23         __import__('bibl.rules.' + module[:-3], locals(), globals())
24     del module
25     return _ALL_RULES
```

**Listing 2.8.13:** `bibl: rule.py` - Bevat functies voor het registreren van entry en tekstregels, evenals het laden van alle regels vanuit de 'bibl/rules' package.

### Logica en Validatie

De structuur begint met het importeren van de benodigde modules en het definiëren van de `Rule` klasse, die de basis vormt voor alle regels. Vervolgens worden de `EntryRule` en `TextRule` subklassen gedefinieerd voor respectievelijk invoer- en tekstregels. De `RuleStore` klasse dient als container voor alle geregistreerde regels en biedt methoden om deze op te halen en te filteren op basis van de huidige configuratie.

Door gebruik te maken van decorators zoals `register_entry_rule` en `register_text_rule`, kunnen nieuwe regels eenvoudig worden toegevoegd aan de linter. De `load_rules` functie importeert automatisch alle modules in het `bibl/rules` pakket, waardoor alle daarin gedefinieerde regels automatisch worden geladen en geregistreerd.

Deze aanpak zorgt voor een flexibele en uitbreidbare structuur voor het beheren van lintingregels, waardoor de consistentie en correctheid van bibliografische gegevens gewaarborgd blijven. Dit is van groot belang om `bibl` als basis te kunnen zien voor de eigen linter die uitgewerkt zal worden.

#### 2.8.7. `text_utils.py` - Helper Functies voor Tekstverwerking

In dit bestand staan helperfuncties voor tekstverwerking gedefinieerd die worden gebruikt door de lintingregels en andere commando's. Deze functies helpen bij

het vinden van regelnummers, het formatteren van regels als Markdown-tabellen en andere tekstverwerkingsactiviteiten.

### Importeren van Noodzakelijke Modules

Net gelijk bij de andere componenten, worden er ook hier eerst de benodigde modules en bibliotheken geïmporteerd. In dit geval zijn dit: reguliere expressies, typing voor typeannotaties, en een externe bibliotheek voor het genereren van Markdown-tabellen.

```
1 import re
2 from typing import List
3
4 import markdown_table
5
6 # Month name dictionary for pybtex
7 from bibl.rule import Rule
8
9 MONTH_NAMES = {
10     'jan': 'jan',
11     'feb': 'feb',
12     'mar': 'mar',
13     'apr': 'apr',
14     'may': 'may',
15     'jun': 'jun',
16     'jul': 'jul',
17     'aug': 'aug',
18     'sep': 'sep',
19     'oct': 'oct',
20     'nov': 'nov',
21     'dec': 'dec'
22 }
```

**Listing 2.8.14:** bibl: text\_utils.py - Bevat helperfuncties en constanten, inclusief de maandnaam dictionary voor pybtex.

De declaratie van `MONTH_NAMES` (regel 9 en volgende) zorgt voor een gestandaardiseerde verwerking van maandnamen in bibliografische gegevens. Dit biedt consistentie, eenvoudige toegang, compatibiliteit met externe bibliotheken zoals in dit geval `pybtex`, en verbetert de leesbaarheid en onderhoudbaarheid van de code.

### Functie `find_match_line_number`

De functie `find_match_line_number` wordt gebruikt om het regelnummer en de *offset*, het aantal karakters vanaf het begin van de tekst, van een regex-match in een gegeven tekst te vinden.



```

1 def find_match_line_number(text: str, pattern: str, group: int) ->
  (int, int):
2     r"""Find the line number and offset of a regex match.
3
4     :param text: text to match
5     :param pattern: regex pattern
6     :param group: regex group number of the intended match
7     :return: Line number (based on \n characters) of first
8             occurrence of the
9             specified match, offset of the first occurrence of the match in
10            the line
11            """
12     regex = re.compile(pattern)
13     match = next(regex.finditer(text))
14     start = match.start(group)
15     lineno = text.count('\n', 0, start)
16     if lineno:
17         offset = start - text.rfind('\n', 0, start)
18     else:
19         offset = start
20     return lineno + 1, offset + 1

```

**Listing 2.8.15:** bibl: tex\_utils.py - Definieert de functie `find_match_line_number` die het regelnummer en de offset van een regex match binnen een tekst retourneert.

### Functie `find_entry_line_number`

De functie `find_entry_line_number` wordt gebruikt om het regelnummer van een BibTeX-entry in een gegeven tekst te vinden op basis van de entry-sleutel.

```

1 def find_entry_line_number(text: str, key: str) -> (int, int):
2     r"""Find the file line number of a pybtex entry.
3
4     :param text: BibTeX file string
5     :param key: Entry key
6     :return: Line number (based on \n characters) of first
7             occurrence of the
8             entry key, offset of the first occurrence of the key in the line
9             """
10    pattern = r'\s*@[a-zA-Z]+\s*{\s*(' + key + r')\s*,'
11    return find_match_line_number(text, pattern, 1)

```

**Listing 2.8.16:** bibl: tex\_utils.py - Definieert de functie `find_entry_line_number` die het regelnummer en de offset van een BibTeX entry key binnen een tekst retourneert.

### Functie `format_rules_markdown_tables`

De functie `format_rules_markdown_tables` wordt gebruikt om een lijst van lintingregels als een Markdown-tabel te formatteren.

```

1 def format_rules_markdown_tables(rules: List[Rule]) -> str:
2     """Format a list of bibl rules as a Markdown table.
3
4     :param: rules: a list of Rule instances
5     :return: A string containing a markdown table of human readable
6             rules
7     """
8     result = "# bibl rules\n"
9     headers = ["Rule ID", "Rule description"]
10    matrix = []
11    for i, rule in enumerate(rules):
12        if i != 0 and rule.rule_id[0] != rules[i - 1].rule_id[0]:
13            result += "## " + rules[i - 1].rule_id[0]
14            result += "\n"
15            result += markdown_table.render(headers, matrix)
16            result += "\n\n"
17            matrix = []
18            matrix.append([f'`{rule.rule_id}`', rule.description])
19    result += "## " + rule.rule_id[0]
20    result += "\n"
21    result += markdown_table.render(headers, matrix)
22    return result

```

**Listing 2.8.17:** `bibl: tex_utils.py` - Definieert de functie `format_rules_markdown_tables` die een lijst van regels formatteert als een Markdown tabel.

### Beschrijving van de Functies

- `find_match_line_number`: Deze functie zoekt naar een regex-match in de tekst en retourneert het regelnummer en de offset van de eerste match. Dit is nuttig voor het lokaliseren van specifieke patronen binnen een tekst.
- `find_entry_line_number`: Deze functie zoekt naar het regelnummer van een BibTeX-entry op basis van de gegeven entry-sleutel. Dit helpt bij het vinden van specifieke bibliografische items binnen een BibTeX-bestand.
- `format_rules_markdown_tables`: Deze functie neemt een lijst van `Rule` objecten en formatteert deze als een Markdown-tabel. Dit maakt het mogelijk om regels op een leesbare en georganiseerde manier weer te geven<sup>12</sup>.

<sup>12</sup>[https://arnevdkgitlab.io/-/bibl/-/jobs/952978205/artifacts/all\\_rules.html](https://arnevdkgitlab.io/-/bibl/-/jobs/952978205/artifacts/all_rules.html)

## Gebruik van Externe Bibliotheken

De functies maken gebruik van verschillende externe bibliotheken:

- `re`: Voor het werken met reguliere expressies.
- `markdown_table`: Voor het genereren van Markdown-tabellen.
- `pybtex`: Voor het werken met bibliografische gegevens.

Deze functies zijn cruciale hulpmiddelen voor het verwerken en manipuleren van de inhoud uit een bibTeX-bestand. Ze zorgen ervoor dat de regels en de bijbehorende bibliografische gegevens consistent en leesbaar blijven.

### 2.8.8. Regels

Zoals weergegeven in Figuur 2.3, is er een `rules` map aanwezig. Deze map bevat naast een lege `__init__.py` ook andere Python-bestanden die de regels van `bibl` definiëren. De regels zijn onderverdeeld in verschillende categorieën: regels voor de gehele database, regels per entry, regels per veld, regels voor onbekende velden of entrytypes, en tot slot tekstuele regels om de *stijl* consistent te houden en te controleren of er ASCII-tekenen worden gebruikt.

In de volgende secties worden enkele regels van elke categorie toegelicht.

#### database\_rules.py - databaseregels

De code in dit bestand implementeert linterregels die de consistentie van een compleet BibTeX-bestand controleren. Deze regels worden gebruikt om bepaalde voorwaarden te valideren, zoals de alfabetische volgorde van de entries, de aanwezigheid van de *preamble* (preambule in het Nederlands) op de eerste regel, en mogelijke duplicaten op basis van titels.

#### Regel voor Preamble op de Eerste Regel

De regel `D01`, gedefinieerd door de methode `line_length` controleert of de preamble begint op de eerste regel van het document. Indien dit niet het geval is, wordt een linterwaarschuwing gegenereerd.

```

1 @register_text_rule('D01', 'Preamble should begin at first line of
   document')
2 def line_length(line_number, line, text):
3     """Raise a linter warning when the preamble is not on the first
   line.
4
5     :param line_number: The number of the current line in the
   bibliography
6     :param line: The content of the current line in the bibliography
7     :param text: The entire bibliography

```

```

8     :return: True if no preamble is present or the preamble starts at
          line 1 of
9     the BibTeX file, False otherwise.
10    """
11    regex = re.compile(r'^\s*@preamble')
12    return not regex.match(line.lower()) or line_number == 0

```

**Listing 2.8.18:** bibl: database\_rules.py - Definieert de regel line\_length die een waarschuwing geeft als de preamble niet op de eerste regel van het document begint.

### Regel voor Mogelijke Duplicaten op Basis van Titels

De regel D02, gedefinieerd door de methode title\_duplicate controleert of er mogelijke duplicaten zijn op basis van titels die sterk op elkaar lijken. Indien dit het geval is, wordt ook hiervoor een linterwaarschuwing gegenereerd.

```

1 @register_entry_rule('D02', 'Possible duplicate entry based on
   similar titles')
2 def title_duplicate(key, entry, database):
3     """Raise a linter warning when entries with similar titles are
         present.
4
5     :param key: The key of the current bibliography entry
6     :param entry: The current bibliography entry
7     :param database: All bibliography entries
8     :return: True if the fuzzy match partial ratio of the title of
           the current
9     entry with any other entry exceeds 90%, False otherwise.
10    """
11    if 'title' not in entry.fields:
12        return True
13    for e in database.entries.values():
14        if 'title' not in e.fields:
15            continue
16        t1 = unicode(entry.fields['title']).lower()
17        t2 = unicode(e.fields['title']).lower()
18        if e != entry and fuzz.partial_ratio(t1, t2) > 90:
19            return False
20    return True

```

**Listing 2.8.19:** bibl: database\_rules.py - Definieert de regel title\_duplicate die een waarschuwing geeft wanneer invoer met vergelijkbare titels aanwezig is.

### entry\_rules.py - entryregels

In dit bestand bevinden zich de regels die per entry gecontroleerd worden. Eén van deze wordt hier meer in detail bekeken.

#### Regel voor Gebruik van ët al. in het Auteursveld

De regel `author_et_al` controleert of het auteursveld "ët al." bevat en genereert een waarschuwing indien dit het geval is. Deze regel vereist dat auteurs en redacteurs volledig worden gespecificeerd en dat er geen worden weggelaten door het expliciete gebruik van "ët al."

```

1 @register_entry_rule(
2     'E03',
3     'The usage of `et al.` in the author field should be replaced by
4     a list '
5     'of all authors')
6 def author_et_al(key, entry, database):
7     """Raise a linter warning when the authors or editors contains
8     'et al.'.
9
10    Authors and editors should be specifed as a complete list of all
11    names.
12
13    :param key: The key of the current bibliography entry
14    :param entry: The current bibliography entry
15    :param database: All bibliography entries
16    :return: True if none of the author or editor names of the
17            current entry
18            contains the substring 'et al' (case insensitive), False
19            otherwise.
20    """
21     if 'author' not in entry.fields:
22         return True
23     return 'et al' not in entry.fields['author'].lower()

```

**Listing 2.8.20:** `bibl: entry_rules.py` - Definieert de regel `author_et_al` die een waarschuwing geeft wanneer 'et al.' wordt gebruikt in het auteursveld en moet worden vervangen door een lijst van alle auteurs.

Net zoals bij de database regels, is het duidelijk dat er regels op een consistente en duidelijke manier gedefinieerd kunnen worden.

### field\_rules.py - Veldregels

In deze sectie wordt een codefragment besproken dat dynamisch regels genereert voor het controleren van vereiste velden in verschillende typen BibTeX-entries.

Deze regels worden geregistreerd en gebruikt door de linter om te controleren of alle vereiste velden aanwezig zijn in elke entry.

### Dynamische generatie van regels

De volgende code genereert dynamisch regels op basis van de configuratie die is gespecificeerd in de `type_spec`-sectie van de configuratie. Voor elk entrytype en elk vereist veld wordt een regel aangemaakt en geregistreerd.

```

1  for entry_type, spec in get_config()['type_spec'].items():
2      for req_field in spec['required']:
3          rule_id = 'M01{}'.format(entry_type.capitalize(),
4                                   req_field.capitalize())
5          message = 'Missing required field `{}` for entry type `{}`'
6          message = message.format(req_field, entry_type)
7
8          @register_entry_rule(rule_id, message)
9          def check_required_field_present(key, entry, database,
10                                         entry_type=entry_type,
11                                         req_field=req_field):
12              """Raise a linter warning when not all required fields
13                 are present.
14
15                 Required fields for an entry type are defined in the
16                 configuration
17                 with the `required` list of field types for a specific
18                 entry
19                 type in the `type_spec` dictionary.
20
21                 :param key: The key of the current bibliography entry
22                 :param entry: The current bibliography entry
23                 :param database: All bibliography entries
24                 :param entry_type: Anchor variable to pass the local
25                 variable `entry_type` from outer scope
26                 :param req_field: Anchor variable to pass the local
27                 variable
28                 `req_field` from outer scope
29                 :return: True if the current entry contains all required
30                        fields,
31                        False otherwise
32              """
33              if entry.type == entry_type:
34                  return req_field in entry.fields

```

```
30         else:
31             return True
```

**Listing 2.8.21:** `bibl: field_rules.py` - Definieert regels om een linterwaarschuwing te geven wanneer niet alle verplichte velden aanwezig zijn voor een specifiek invoertype, zoals gespecificeerd in de configuratie.

### Beschrijving van de regel

De gegenereerde regel `check_required_field_present` controleert of een BibTeX-entry alle vereiste velden bevat zoals gedefinieerd in de configuratie. Indien een vereist veld ontbreekt, wordt er een linterwaarschuwing gegenereerd.

- `key`: De sleutel van de huidige BibTeX-entry.
- `entry`: De huidige BibTeX-entry.
- `database`: Alle BibTeX-entries.
- `entry_type`: De variabele om het lokale `entry_type` door te geven vanuit de buitenste scope.
- `req_field`: De variabele om het lokale `req_field` door te geven vanuit de buitenste scope.
- `return`: `True` als de huidige entry alle vereiste velden bevat, anders `False`.

Deze dynamische benadering maakt het mogelijk om op een flexibele manier regels te definiëren en te controleren of alle noodzakelijke informatie aanwezig is in de BibTeX-entries. Dit helpt bij het handhaven van de volledigheid en correctheid van bibliografische gegevens.

### 2.8.9. Conclusie

`bibl` biedt een uitgebreide en flexibele oplossing voor het waarborgen van de consistentie en volledigheid van BibTeX-bibliografische gegevens. Door gebruik te maken van dynamisch gegenereerde linterregels kan het programma verschillende aspecten van een BibTeX-bestand controleren, waaronder de alfabetische volgorde van entries, de correcte opmaak van velden, en de aanwezigheid van alle vereiste velden voor verschillende entrytypes.

De architectuur van `bibl` maakt het mogelijk om eenvoudig nieuwe regels toe te voegen en bestaande regels aan te passen. Dit wordt bereikt door het gebruik van decorators voor het registreren van regels en hulpfuncties voor specifieke controlemechanismen. Bovendien zorgt de integratie met externe bibliotheken zoals `fuzzywuzzy` en `unidecode` voor robuuste en efficiënte tekstverwerkingsmogelijkheden.

De gestructureerde opzet van de configuratiebestanden en de modulariteit van de regels dragen bij aan de onderhoudbaarheid en uitbreidbaarheid van het systeem. Door het volgen van best practices in softwareontwikkeling, zoals het gebruik van duidelijke en gedocumenteerde functies, wordt de leesbaarheid en het begrip van de code vergroot.

Naast het bezitten van een goed doordachte structuur en opbouw, is `bibl` een waardevol hulpmiddel voor iedereen die werkt met BibTeX-bestanden, en draagt het bij aan de professionalisering en kwaliteit van academisch schrijven en onderzoek.

Gezien al deze eigenschappen en in vergelijking met wat er onder een algemene linteropbouw (zie 2.7) verstaan wordt, is `bibl` zeker een geschikt voorbeeld voor deze [Proof of Concept](#).

## 2.9. Open-source

Net als `bibl`, heeft deze [Proof of Concept](#) als visie om open source te zijn zodat iedereen er gebruik van kan maken en ook iedereen in staat is om een bijdrage te leveren. Open-source software voldoet aan specifieke criteria die het een waardevolle en veelzijdige keuze maken. Onder deze criteria wordt er iets als open-source erkend:

- **Vrij verspreid** kan worden, waardoor het voor iedereen toegankelijk is.
- De **broncode beschikbaar stelt**, zodat ontwikkelaars het kunnen begrijpen, aanpassen en verbeteren.
- **Modificaties en afgeleide werken** toestaat, waardoor een levendige gemeenschap van bijdragers ontstaat.
- De **integriteit van de oorspronkelijke broncode waarborgt**, zodat de software betrouwbaar blijft.
- **Niet discrimineert** op basis van personen of groepen, en voor iedereen toegankelijk is.
- **Niet beperkt is tot specifieke vakgebieden**, waardoor het breed inzetbaar is.
- **Rechten toepast op alle partijen** die de software verspreiden, wat eerlijkheid bevordert.
- **Niet afhankelijk is van een specifieke software distributie**, waardoor het flexibel blijft.
- **Geen beperkingen oplegt aan andere software** die ermee wordt verspreid, wat samenwerking stimuleert.



- **Technologie-neutraal** is, zodat het zich kan aanpassen aan veranderende omstandigheden.

Kortom, open-source is toegankelijk, krachtig en bevordert innovatie in de digitale wereld voor iedereen (open source initiative, 2006).

## 2.10. Pipelines

Pipelines spelen een cruciale rol in de automatisering van het bouwen, testen en implementeren van applicaties. Ze maken het mogelijk om verschillende processen te orkestreren, zodat code consistent en betrouwbaar van ontwikkeling naar productie kan worden verplaatst. Ook voor deze **Proof of Concept** kan een pipeline als een essentieel onderdeel beschouwd worden.

### Functie van Pipelines

Pipelines automatiseren en stroomlijnen het proces van softwareontwikkeling door de workflow op te delen in afzonderlijke fasen. Elke fase voert een specifieke taak uit, zoals het bouwen van de code, het uitvoeren van tests, het voorbereiden en distribueren van de software, alsook het implementeren ervan in verschillende omgevingen. Deze automatisering vermindert de handmatige inspanning, zorgt voor consistentie en versnelt de feedbackloop voor ontwikkelaars. Hoewel **Continuous Integration (CI)** en **Continuous Deployment (CD)** pipelines vaak de eerste soorten zijn die opkomen, zijn pipelines ook voor vele andere zaken in te schakelen. Denk maar aan verschillende operationele taken, zoals het vernieuwen van certificaten, het detecteren van infrastructuurdrift en het uitvoeren van gezondheidscontroles op geïmplementeerde applicaties (van Merode, 2023).

### Structuur van Pipelines

De structuur van een pipeline omvat doorgaans meerdere fasen en taken, die sequentieel of parallel kunnen worden uitgevoerd. van Merode (2023) leert ons dat een goed gestructureerde pipeline de volgende fasen kan omvatten:

- **Valideer ingangscriteria:** Zorgt ervoor dat aan alle vereisten is voldaan voordat het buildproces begint.
- **Voer buildproces uit:** Compileert de broncode en bereidt de build-artefacten voor.
- **Voer unittests uit:** Voert geautomatiseerde tests uit om de functionaliteit van individuele eenheden code te verifiëren.
- **Analyseer code:** Controleert de codekwaliteit en veiligheid op kwetsbaarheden.
- **Verpak artefact:** Bundelt de gecompileerde code in implementeerbare eenheden.

- **Publiceer artefact:** Uploadt de build-artefacten naar een repository.
- **Implementeer artefact naar testomgeving:** Implementeert de build in een testomgeving.
- **Voer tests uit:** Voert integratie- en systeemtests uit in de testomgeving.
- **Valideer uitgangscriteria:** Zorgt ervoor dat alle tests zijn geslaagd en de build gereed is voor productie.
- **Implementeer naar productie:** Verplaatst de definitieve build naar de productieomgeving.

### Toepassingen van Pipeline Resultaten

De resultaten van pipeline-uitvoeringen zijn cruciaal voor verschillende belanghebbenden in de softwareontwikkelingscyclus:

- **Developers:** Krijgen directe feedback over de wijzigingen die ze doorvoeren, wat helpt om snel problemen te identificeren en op te lossen.
- **Operations Teams:** Gebruiken de pipeline-resultaten om de gezondheid en prestaties van applicaties in productie te monitoren, waardoor betrouwbaarheid en stabiliteit worden gegarandeerd.
- **Management:** Verkrijgt inzicht in het ontwikkelingsproces via statistieken en KPI's, waardoor data-gedreven besluitvorming mogelijk wordt.

Pipelines kunnen ook worden ontworpen om complexe workflows te ondersteunen, zoals multi-branch en multi-stage configuraties, die verschillende takken van code en verschillende fasen van ontwikkeling tegelijkertijd verwerken (van Merode, 2023).

Samengevat zijn pipelines een essentieel onderdeel van moderne softwareontwikkeling. Ze bieden automatisering, consistentie en efficiëntie in de continue integratie- en leveringsprocessen. Ze helpen bij het handhaven van de codekwaliteit, verminderen handmatige interventie en versnellen de implementatiecycli, wat uiteindelijk leidt tot meer betrouwbare en onderhoudbare softwaresystemen. Binnen deze **Proof of Concept** zouden een **CI** en een **CD** pipeline goed benut kunnen worden.

## 2.11. Kanban

Overzichtelijk te werk gaan is voor elk project van belang. Ook bij het ontwikkelen van deze **Proof of Concept** was er geen uitzondering. Om een simplistisch maar effectief overzicht te behouden, werd er een kanbanbord gebruikt.

Een Kanban-bord is een visueel hulpmiddel dat teams gebruikt voor het beheren van projecttaken.

De belangrijkste onderdelen van een Kanban-bord zijn de kaarten en kolommen. Kaarten vertegenwoordigen individuele taken en bevatten details zoals beschrijvingen en deadlines. Kolommen vertegenwoordigen verschillende stadia van de workflow, zoals 'te doen', ' bezig', en 'gedaan'. Het beperken van het aantal taken in uitvoering zorgt voor efficiënter werken door de focus te behouden en afleiding te verminderen.

Kanban-borden maken gebruik van zes kernpraktijken: de workflow visualiseren, het werk in uitvoering beperken, workflows beheren, expliciete beleidsregels implementeren, ruimte voor feedback bieden, en voortdurend zoeken naar verbetering. Elk van deze praktijken draagt bij aan het verhogen van flexibiliteit, het reduceren van downtime en het verhogen van de efficiëntie binnen teams (Hennigan, 2024). Zoals eerder vermeld, werd er binnen deze proef ook een simpele vorm van een kanbanbord gebruikt om een overzicht te bewaren van welke functionaliteiten er al dan niet afgewerkt waren. Dit was handig om te weten of de **Minimum Viable Product (MVP)** gehaald kon worden tegen de deadline.

# 3

## Methodologie

Deze **Proof of Concept** is opgesplitst in meerdere fasen. Voor het onderzoeken en effectief opstellen hiervan waren alleen toegang tot een computer, technisch inzicht en logisch redeneren vereist. Het doel was om een open-source **Proof of Concept** te ontwikkelen die als basis kon dienen voor een volledig functionele linter. Voor het gehele proces waren 14 weken beschikbaar. De fasen zijn verdeeld over deze periode met een buffer van twee weken om eventuele fouten in de schattingen op te vangen. De benodigde tijd werd ingeschat op basis van het aantal vereiste onderzoeken en het verwachte resultaat aan het einde van elke fase.

### 3.1. Fase 1: Literatuurstudie

In deze fase werd onderzoek gedaan naar bestaande linters, met en zonder betrekking tot  $\text{\LaTeX}$ . Er werd gekeken naar de programmeertalen waarin deze linters waren geschreven, hun werking, specificaties en functionaliteiten. Daarnaast werd onderzocht hoe een **CI**-pipeline effectief werkt om de linter efficiënt te kunnen integreren in workflows. Ook werd het verschil tussen BibLaTeX en BibTeX onderzocht om de oorzaak van hun incompatibiliteit te achterhalen. Eventuele andere relevante zaken die tijdens het onderzoek naar voren kwamen, werden ook meegenomen.

Het resultaat van deze fase was een informeel document met alle bevindingen, onderzoeken en nuttige informatiebronnen over bestaande linters en andere relevante onderwerpen. Dit document diende als kennisbron voor de volgende fasen en werd uiteindelijk verwerkt in de literatuurstudie.

### 3.2. Fase 2: Technische Analyse en Experimenten

In deze fase was het doel om de lijst van functionele en niet-functionele requirements aan te vullen en te prioriteren. De keuze van de programmeertaal, libraries

en andere softwaretools werd hier gemaakt. Deze keuzes werden gebaseerd op experimenten met kleine prototypes in elke kandidaat-programmeertaal, waarbij zowel de snelheid als andere voor- en nadelen werden geëvalueerd.

### 3.3. Fase 3: Software Ontwikkeling | Proof of Concept

In deze fase werd de **Proof of Concept** ontwikkeld op basis van de verzamelde bronnen. Een werkende versie werd opgesteld, inclusief testen om de werking te garanderen en de uitbreidbaarheid te vergemakkelijken. Er werd ook documentatie geschreven om duidelijkheid te verschaffen aan iedereen die bijdroeg aan het project. Een kanbanbord werd opgezet om het overzicht over de voortgang te behouden en de haalbaarheid van de MVP te bewaken.

Aan het einde van deze fase was een werkende **Proof of Concept** beschikbaar op een GitHub-repository<sup>1</sup> van de auteur van dit onderzoek, Tristan Cuvelier. Deze fase nam aanzienlijke tijd in beslag vanwege het verwachte resultaat, met een ruime marge voor eventuele problemen tijdens de ontwikkeling.

### 3.4. Conclusie

Tot slot wordt er een conclusie getrokken waarin de mate van succes van deze **Proof of Concept** wordt bepaald. De opgeleverde functionaliteiten worden geëvalueerd, evenals de mate waarin aan de niet-functionele requirements is voldaan. Daarnaast worden de resterende werkpunten en de toekomstvisie van deze **Proof of Concept** besproken.

---

<sup>1</sup><https://github.com/MrClassicT/bibla>

# 4

## Technische Analyse en Experimenten

Net zoals bij alle software, was het ook hier van belang om eerst grondig onderzoek te doen naar wat er effectief nodig was en naar welke technologieën er best gebruikt konden worden. Deze fase was van groot belang voor het verdere verloop van het project. Het was de bedoeling om een duidelijk beeld te krijgen van wat er nodig was en hoe dit het best kon worden aangepakt. Een duidelijk overzicht van de functionele en niet-functionele requirements was hierbij van groot belang.

Als eerste werd er onderzocht wat een linter nu eigenlijk was en hoe deze werkte. Daarnaast werd er ook gekeken naar de verschillende soorten linters die er bestonden en naar de programmeertalen waarin deze gemaakt waren. Zodra duidelijk was wat er verwacht werd, werd er een lijst opgesteld van de gewenste functionele en niet-functionele requirements. Deze werden dan gestructureerd naargelang hun prioriteit zodat er bepaald kon worden wat er al dan niet tot de **Minimum Viable Product (MVP)** zou behoren.

Met deze lijst kon er dan gekeken worden naar de keuze van de programmeertaal die gebruikt kon en zou worden voor het maken en opzetten van de proof of concept. De keuze van de programmeertaal werd bereikt door het opstellen van kleine prototypes in elke kandidaat-programmeertaal. Deze werden dan getimed en daarnaast werden ook andere voor- en nadelen van elk bekeken om zo tot de beste optie te komen.

Uit de lijst van functionaliteiten werden er twee gekozen die als eerste uitgewerkt zouden worden om zo een eerste versie van de linter te bekomen. Deze functionaliteiten waren: het detecteren van duplicaten en de detectie om te zien of alle required fields wel aanwezig waren.

## 4.1. Functionele en Niet-Functionele Requirements

Nadat de volledige reeks linters was geanalyseerd, kon een lijst van zowel functionele als niet-functionele vereisten worden opgesteld. Dankzij de promotor van deze thesis werd dat proces versneld. Een informatieve lijst werd verstrekt, waarin onder andere een opsomming van de gewenste regels stond. Aangezien de promotor uitgebreide technische kennis van het onderwerp heeft en beschouwd kan worden als de *klant* bij de ontwikkeling van deze tool, werd gefocust op de uitwerking van de specifieke regels volgens zijn wensen.

Een overzicht van de **functionele requirements** kan in tabel 4.1 worden bezichtigd. Merk op dat deze lijst slechts een richtlijn is voor tijdens de effectieve uitwerking.

Wat de **niet-functionele** requirements betreft:

- Er wordt gewenst dat de linter gebruikt kan worden vanuit de **CLI** zodat deze in pipelines gebruikt kan worden.
- De linter dient **vlot** te werken, zodat de tool strikt als handig beschouwd kan worden en zeker niet als iets storends.
- De linter dient **uitbreidbaar** te zijn eens deze proef-periode is afgelopen. Er wordt duidelijke, gestructureerde en gedocumenteerde code verwacht.
- Er wordt verwacht dat het **eenvoudig** te gebruiken is, begrijp hieronder dat het beschikbaar zal zijn via een pip-installatie gelijk andere Python modules.

## 4.2. Keuze van de Programmeertaal

Om de gepaste programmeertaal te vinden, werden diverse kandidaat-programmeertalen overwogen. De kandidaat-programmeertalen waren: Rust, JavaScript en Python.

Door het analyseren van reeds bestaande linters en het afgaan van een hele lijst aan programmeertalen, werd er besloten om met deze drie programmeertalen te experimenteren. Om ervoor te zorgen dat de testprogramma's niet te uitgebreid werden en zowel gelijk als eerlijk konden worden vergeleken, zouden er slechts twee functionaliteiten worden uitgewerkt. Deze twee functionaliteiten waren: detecteren van duplicaten en detecteren of alle required fields aanwezig waren. Om dit onderzoek niet nodeloos lang te maken, werd er besloten om de broncode van deze programma's niet op te nemen in dit document. De broncode is echter wel te vinden op GitHub voor zowel de kleine **Proof of Concept (PoC)**'s<sup>1</sup> als voor het uiteindelijk resultaat<sup>2</sup>.

<sup>1</sup><https://github.com/MrClassicT/bibLaTeX-linter-pocs>

<sup>2</sup><https://github.com/MrClassicT/bibla>

Het test .bib bestand dat gebruik werd, zag er als volgt uit:

```
1 % Encoding: UTF-8
2
3 @Online{Arnedo2021,
4   date   = {2021-10-05},
5   title  = {Why you should store custom logs of your data pipelines
6             and how to build a Data Catalog with them},
7   url    =
8           {https://towardsdatascience.com/why-you-should-store-custom-logs
9             -of-your-data-pipelines-and-how-to-build-a-data-catalog-with-
10            them-ee96a99a1c96},
11 }
12 @Comment{jabref-meta: databaseType:biblatex;}
```

**Listing 4.2.1:** Mini-PoC: Test bestand.

### 4.2.1. JavaScript

JavaScript is een programmeertaal die zeer toegankelijk en gekend is onder de meeste developers. Dit was ook de eerste programmeertaal waarin een testversie werd opgesteld, deze werd dan achteraf vertaald naar zowel Python als Rust om te kijken welke van de drie talen het meest geschikt was voor deze case.

De JavaScript versie was van een persoonlijk perspectief de beste om mee te beginnen gezien dit de meest vertrouwde taal was. Direct kan er al opgemerkt worden dat er gebruik gemaakt werd van modules, dit was noodzakelijk om de linter overzichtelijk, uitbreidbaar en onderhoudbaar te maken. Moest alles in éénzelfde bestand zitten, zou het veel te groot en onoverzichtelijk worden.

```
1 // Import submodules
2 const { exit } = require('process');
3 const { readFromFile } = require('./helper/readFromFileAsync.js');
4 const { checkForMissingFields } =
5     require('./checks/missingfields.js');
6 const { checkForDuplicates } = require('./checks/duplicates.js');
7 const { entryPattern } = require('./components/regex.js');
8 const { toFileUrl } = require('./helper/getFileUrl.js')
```

**Listing 4.2.2:** Mini-JS-PoC - Importeert submodules voor het uitvoeren van controles op ontbrekende velden en duplicaten, en definieert een reguliere expressie voor het patroon van BibLaTeX vermeldingen.

Als eerste wordt er een bestand ingelezen. Eens dat gebeurt is, dient er een onderscheid gemaakt te worden tussen alle verschillende bronnen binnen dit bestand.



De bronnen in dit bestand, worden ook wel 'entries' genoemd. Dit werd op de volgende manier gedaan:

```
1 // Extract all entries
2 const entries = [ ... fileContent.matchAll(entryPattern)].map(match =>
  ({
3     type: match[1],
4     citationName: match[2].trim(),
5     content: match[3],
6     position: match.index
7   }));
```

**Listing 4.2.3:** Mini-JS-PoC - Haalt alle vermeldingen uit het bestand op, waarbij elk item wordt geformatteerd als een object met type, citationName, inhoud en positie.

Op deze wijze werd niet alleen een 'entry' ontdekt, maar werd ook meteen onderscheid gemaakt tussen de bronsoort, de sleutel, de inhoud en de positie van waar deze voorkomt in het bestand. Om dit te doen, werd er gebruik gemaakt van een reguliere expressie. Met een reguliere expressie is het mogelijk om patronen binnen teksten te vinden. Gezien de structurele vorm van een .bib bestand, is het mogelijk om reguliere expressies op een eenvoudige manier te benutten om zo een gedetailleerde analyse uit te voeren op de input.

In detail zal er niet ingegaan worden op hoe deze reguliere expressies werken, maar de twee reguliere expressies die nodig waren binnen deze mini-PoC waren de volgende:

```
1 // Extract each entry from the .bib file.
2 const entryPattern = /@(\w+)\{([^\,]+),\s*(.*?)\},\s*\}/sg;
3 // Extract each field from the entry.
4 const fieldPattern = /(\w+)\s*=\s*(?:\{(.*?)\}|(\S+))/sg;
```

**Listing 4.2.4:** Mini-JS-PoC - Definieert reguliere expressies voor het patroon van BibLaTeX vermeldingen en velden.

De eerste zorgt ervoor dat elke bron van elkaar onderscheden kan worden. De tweede gaat dan weer van toepassing zijn op een iets lager niveau. Daarmee kan er binnen een bron gekeken worden naar de individuele velden data die ervan bijgehouden worden, of juist om te kijken naar welke velden er missen bijvoorbeeld. Om te kijken welke velden er missen, werd volgende methode gebruikt:

```
1 function checkForMissingFields(entry) {
2
3     let fields = {};
4
5     while ((fieldMatch = fieldPattern.exec(entry.content)) !== null)
6         {
7             fields[fieldMatch[1]] = fieldMatch[2] || fieldMatch[3];
8         }
9
10    let missingFields = [];
11    if (entryRequirements[entry.type]) {
12        entryRequirements[entry.type].required.forEach(field => {
13            if (!fields[field]) {
14                missingFields.push(field);
15            }
16        });
17    }
18    return missingFields;
19 }
```

**Listing 4.2.5:** Mini-JS-PoC - Controleert op ontbrekende velden in een BibLaTeX-vermelding en retourneert een lijst met ontbrekende velden.

Hier wordt er een lijst gebruikt van velden die binnen de entry voor dienen te komen. Indien er gemerkt wordt dat er een bepaald veld niet in voorkomt, wordt de naam van het ontbrekende veld toegevoegd aan een lijst die teruggegeven wordt eens alles gecontroleerd is. Deze lijst wordt dan getoond aan de user.

Vóór het onderzoek naar ontbrekende velden wordt eerst gecontroleerd op eventuele duplicate entries. Ook wordt onderzocht of er mogelijk entries zijn die licht verschillen maar toch dezelfde verwijzingsleutel bevatten. Gezien het feit dat een duplicaat sleutel niet is toegestaan, wordt deze controle als eerste uitgevoerd.

Deze controle werd als volgt geïmplementeerd:

```

1 function checkForDuplicates(entries) {
2     const citationNames = entries.map(entry => entry.citationName);
3     const duplicates = citationNames.filter((citationName, index, self) =>
4         self.indexOf(citationName) !== index
5         && self.lastIndexOf(citationName) === index
6     );
7
8     if (duplicates.length > 0) {
9         const plural = duplicates.length > 1;
10        console.error(`Caution: ${plural ? "" : "A "}duplicate key
11        ${plural ? "s" : ""} ${duplicates} ha${plural ? "ve" : "s"}
12        been found!`);
13        exit(1);
14    }
15 }

```

**Listing 4.2.6:** Mini-JS-PoC - Controleert op duplicaten in de citatienamen van BibLaTeX-vermeldingen en geeft een waarschuwing als duplicaten worden gevonden.

In deze controle valt op dat er een gebrek aan consistentie is in de manier waarop informatie aan de gebruiker wordt gepresenteerd. In tegenstelling tot de andere controle, waarbij dit pas achteraf gebeurde, gebeurt dit hier onmiddellijk zodra we het detecteren. Dit kan worden toegeschreven aan het feit dat dit een testversie is, ontworpen om een eerste indruk te krijgen. Bovendien zou een directe presentatie van de informatie mogelijk efficiënter kunnen zijn dan wanneer deze eerst wordt teruggestuurd naar een hoger niveau in het programma.

#### 4.2.2. Python

Gezien het aanzienlijke aanbod aan linters die in Python geschreven zijn en de brede bekendheid ervan, werd besloten om ook Python te gebruiken. Omdat de Python-kennis verzwakt was, werd gebruikgemaakt van de mogelijkheden van AI. De linter **Proof of Concept (PoC)**, geschreven in JavaScript, werd aan ChatGPT-4 voorgelegd en stap voor stap werd gevraagd deze code om te zetten naar een werkend Python-alternatief. Binnen enkele uren werd een eerste werkende versie gerealiseerd. Het was indrukwekkend om te zien hoe ChatGPT omging met de Python-syntaxis. Ook de reguliere expressies die werden gebruikt, hebben op een bepaald moment in de handen van ChatGPT of GitHub Copilot gezeten. AI-tools zijn zeer nuttig om complexe zaken te verduidelijken en te assisteren bij het doen van suggesties voor het aanpakken van specifieke problemen. Het potentieel van AI was hier duidelijk merkbaar en werd optimaal benut, zowel door code te vertalen als door bugs op te lossen.

```
1 entries = [match.groups() for match in
             entry_pattern.finditer(file_content)]
2 formatted_entries = [{
3     'type': entry[0],
4     'citationName': entry[1].strip(),
5     'content': entry[2],
6     'position': match.start()
7 } for match, entry in zip(entry_pattern.finditer(file_content),
                           entries)]
```

**Listing 4.2.7:** Mini-Python-PoC - Verwerkt BibLaTeX vermeldingen uit een bestand, waarbij elk item wordt geformatteerd als een dict met type, citationName, inhoud en positie.

Als deze pythoncode vergeleken wordt met de javascript-code, kan er direct opgemerkt worden dat er gelijkaardige logica achter zit. Dit is uiteindelijk ook het geval want de javascript-code is simpelweg vertaald naar Python code.

De twee functionaliteiten werden op een gelijke wijze vertaald en geïmplementeerd in de kleine Python [Proof of Concept \(PoC\)](#).

```
1 def check_for_missing_fields(entry):
2     fields = {}
3     for match in field_pattern.finditer(entry['content']):
4         key, value1, value2 = match.groups()
5         fields[key] = value1 or value2
6
7     missing_fields = []
8     if entry['type'] in entry_requirements:
9         for field in entry_requirements[entry['type']]['required']:
10            if field not in fields:
11                missing_fields.append(field)
12
13     return missing_fields
14
15 def check_for_duplicates(entries):
16     citation_names = [entry['citationName'] for entry in entries]
17     duplicates = {name for name in citation_names if
18                  citation_names.count(name) > 1}
19     if duplicates:
20         plural = len(duplicates) > 1
21         print(f'Caution: {"A duplicate key" if not plural else
22              "Duplicate keys"} {"has" if not plural else "have"} been
23              found: {", ".join(duplicates)}!')
24     exit(1)
```

**Listing 4.2.8:** Mini-Python-PoC - Controleert op ontbrekende velden in een BibLaTeX-vermelding en op duplicaten in de citatienamen van BibLaTeX-vermeldingen.

Nadelen zijn er natuurlijk ook aan het vertalen van bestaande code naar een andere programmeertaal met behulp van AI. Zo zullen fouten die in de ene taal bestaan, hoogstwaarschijnlijk ook gewoon mee vertaald worden. Dit was bijvoorbeeld het geval wat de reguliere expressie betrefde. Bij een eerste iteratie van deze **Proof of Concept (PoC)**, werkte één van de twee controles niet. Dit bleek achteraf te komen doordat de reguliere expressie iets te gevoelig was aan de hoeveelheid witruimte die er was. Gelukkig kon deze fout relatief snel achterhaald en opgelost worden. Ook is het interessant om te weten dat bij deze **Proof of Concept (PoC)** het programma minder modulair was. Dit kwam door een kennisbarrière waarbij niet gekend was hoe modules echt goed werken in Python. Echter was dit geen dramatisch probleem gezien het maar een kleine **PoC** was en de zekerheid er was dat dit met wat grondiger onderzoek op een later tijdstip opgelost zou kunnen worden. De eerste signalen dat een goede syntax-kennis niet onbelangrijk is, doken hier al op.

### 4.2.3. Rust

Hoewel Rust een veelbelovende kandidaat leek om de programmeertaal te worden waarin de linter zou worden geschreven, werd de ontwikkeling ervan ijverig gestart. Snelheid is echter niet de enige belangrijke factor; ook kennis speelt een cruciale rol, zoals bleek bij de Python **PoC**. Een taal kan nog zo snel zijn, maar als er niemand is die ermee kan werken, is het nutteloos. Rust, geïntroduceerd in 2015, wordt ondanks zijn plaats in de top 20 van de TIOBE Index<sup>3</sup> nog niet door veel mensen gebruikt.

Zoals eerder meerdere malen vermeld, was de keuze van de programmeertaal van groot belang om ervoor te zorgen dat er ook na deze proef verder gewerkt kon worden aan de linter. Rust had hierbij de minst gunstige positie. Het was essentieel om te onderzoeken of het potentieel in snelheid voldoende merkbaar was om de keuze te rechtvaardigen.

Ondanks dat er kennis werd opgedaan over de programmeertaal zelf en dat de capaciteiten van AI werden ingezet, was het niet mogelijk om beide functionaliteiten werkend te krijgen binnen de Rust-**PoC**. De kennis was nog steeds te beperkt, ook voor AI. Aangezien Rust recenter is en minder populair dan Python, hebben de AI-modellen (ChatGPT-4 en Microsoft's Copilot) veel minder referenties om zich op te baseren voor het oplossen van de problemen die optreden. Dit benadrukt het belang van goede programmeerkennis en toont aan dat **Artificiële Intelligentie (AI)** niet zomaar elke programmeur kan vervangen.

Gezien de beperkte tijd werd besloten om de enige werkende functionaliteit te

<sup>3</sup><https://www.tiobe.com/tiobe-index/>

testen en te vergelijken met de snelste andere PoC, waarbij ook daar slechts één controle werd gebruikt. De enige functionaliteit die werkend was in Rust, was de controle op duplicaten.

Net zoals bij de andere twee PoC's, werd er een soortgelijke werkwijze gehanteerd. Eerst worden de verschillende entries van elkaar gescheiden en vervolgens worden er de controles, of in dit geval de controle, op uitgevoerd.

```

1 let entry_pattern = &patterns :: patterns :: ENTRY_PATTERN;
2
3 let entries: Vec<_> = entry_pattern
4   .captures_iter(&file_content)
5   .map(|cap| {
6       let entry_type = cap[1].to_owned();
7       let citation_name = cap[2].trim().to_owned();
8       let content = cap[3].to_owned();
9       let line_number =
10          file_content[.. cap.get(0).unwrap().start()].lines().count();
11          (entry_type, citation_name, content, line_number)
12      })
13      .collect();
14 checker :: checker :: check_for_duplicates(
15     &entries
16     .iter()
17     .map(|(a, b, c, d)| (a.as_str(), b.as_str(), c.as_str(), *d))
18     .collect :: <Vec<_>>(),
19     &file_path,
20 );

```

**Listing 4.2.9:** Mini-Rust-PoC - Verwerkt BibLaTeX vermeldingen uit een bestand, waarbij elk item wordt geformatteerd als een tuple met type, citationName, inhoud en regelnummer, en controleert op duplicaten in de citatienamen van BibLaTeX-vermeldingen.

Elke programmeur kan zelf ook inzien dat deze code ver van optimaal is. Maar opnieuw, de kennisbarrière was eenmaal aanwezig en met een beperkte tijd, diende er compromis gemaakt te worden. Merk op dat er veel lussen gebeuren, lussen die hoogstwaarschijnlijk overbodig of toch zeker weg te werken zijn. Er werden veel transformaties gemaakt met de data waarbij het leek dat er van het éne type naar het andere gegaan werd om dan vervolgens terug naar het éne type te gaan. Duidelijk veel overbodigheden. Voorgaande code bevatte niet eens de volledige check op duplicaten, dat was slechts het begin-bestand waaruit de applicatie start. Hieronder volgt de effectieve controle. Merk ook daar op dat er twee lussen in de methode zitten. Hoewel het grotendeels is voor zaken op een mooie manier te tonen is, blijft het merkwaardig dat er geen efficiëntere manier gevonden kon wor-

den. ChatGPT of Copilot konden hier beide niet beter mee helpen dan ze tot dit punt al gedaan hebben.

```
1 pub fn check_for_duplicates<'a>(
2     entries: &[(&'a str, &'a str, &'a str, usize)],
3     file_path: &str,
4 ) {
5     let mut duplicates: Vec<(&str, usize)> = Vec::new();
6     let mut seen: Vec<&str> = Vec::new();
7
8     for entry in entries.iter() {
9         let citation_name = entry.1;
10        if seen.contains(&citation_name) {
11            duplicates.push((citation_name, entry.3));
12        } else {
13            seen.push(citation_name);
14        }
15    }
16
17    for duplicate in duplicates {
18        println!("All entry names should be unique.");
19        let (entry_name, zerobased_line_number) = duplicate;
20        let line_number = zerobased_line_number + 1;
21        println!(
22            "Duplicate field found in entry '{}{}' at line {}",
23            entry_name, line_number
24        );
25
26        println!("External file path: {}:{}", file_path,
27                line_number);
28    }
```

**Listing 4.2.10:** Mini-Rust-PoC - Controleert op duplicaten in de citatienamen van BibLaTeX-vermeldingen en geeft een waarschuwing als duplicaten worden gevonden, inclusief het bestands-pad en het regelnummer waar de duplicaten zijn gevonden.

#### 4.2.4. Keuze

Na het opstellen van kleine prototypes in elke kandidaat-programmeertaal, werd er gekeken naar de voor- en nadelen van elk, en alsook naar de uitvoeringstijden. Hierbij werd er aan het begin van elk scriptje een timer gestart in de code en werd deze op het einde van de uitvoering gestopt waarna er nog een printfunctie gebeurde om het resultaat te tonen. De resultaten worden weergegeven in bijhorende tabel 4.2.

De linters werden allemaal op eenzelfde bestand gebruikt. Hierbij werd er eerst driemaal het Python commando achter elkaar uitgevoerd en daarna driemaal de JavaScript versie. Het concept van caching, opslaan van bepaalde data in snel beschikbaar geheugen, is hier zichtbaar. De eerste keer dat er iets gebeurde, duurde het langer dan de eerstvolgende keer dat het plaatsvindt. Opmerkelijk is hier dat de JavaScript versie tot wel 12 keer *trager* is dan de Python versie.

In een poging om het caching aspect zoveel mogelijk te vermijden en elke taal een zo gelijk mogelijke kans te geven, werd het bestand meerdere keren gecontroleerd, maar dit keer afwisselend met JavaScript en Python. Eerst werd er tweemaal JavaScript als eerste uitgevoerd en vervolgens werd hetzelfde gedaan met Python als eerste. Wat hier extra opmerkelijk is, is dat JavaScript deze keer uitzonderlijk lang duurde in vergelijking met eerdere keren. 27 milliseconden in vergelijking met de Python versie die het in minder dan 2 milliseconden kon. Hoewel het bij de tweede poging niet veel verschil maakte, kan er opnieuw gemerkt worden dat er een vorm van caching zal plaatsvinden waarbij het voorgaande resultaat nog altijd (deels) opgeslagen bleef. Hierdoor was de JavaScript linter in staat om het bestand meer dan dubbel zo snel uit te voeren. Wat wel nog altijd meer dan vijf keer trager was dan de Python versie. Python is een duidelijke winnaar in vergelijking met JavaScript op vlak van snelheid. Gezien Python daarnaast ook op de *eerste* plaats staat in de TIOBE Index<sup>4</sup> (mei 2024), is het veilig om ervanuit te gaan dat er genoeg mensen de taal kennen om zeker te zijn dat de linter onderhoudbaar blijft nadat dit onderzoek afgelopen is.

Tot slot rest er nog Rust. Zoals eerder vermeld werd er niet in geslaagd om beide functionaliteiten werkende te krijgen. Daarom werd er besloten om enkel de controle op duplicaten uit te voeren. De snelheid van deze uitvoeringstijd, werd dan vergeleken met die van de Python versie, gezien deze de snelste van de twee was. Hoewel het internet deed geloven dat Rust mogelijk sneller zou zijn, bleek dit bij deze test verre van de waarheid te zijn. Python was ook in vergelijking met Rust sneller. Zoals eerder aangehaald, is de implementatiewijze van groot belang en mogelijks de oorzaak van deze resultaten. Echter geeft het nog altijd een realistisch beeld van de verwachtingen, moest er een linter in gemaakt worden. Door het gebrek aan kennis, zou de taal nooit tot het volle potentiële benut kunnen worden en zou de linter nooit zo snel kunnen werken, als dat in theorie mogelijk zou

<sup>4</sup><https://www.tiobe.com/tiobe-index/>



moeten zijn.

Het is in principe wel nog perfect mogelijk dat Rust zich beter schaalbaar naar grotere bestanden toe. Het testbestand bevatte slechts één bron, maar wat als er honderden bronnen in zouden staan? Zou Python's uitvoertijd lineair vergroten of eerder exponentieel? Zou Rust sub-lineair evolueren? Deze vragen zullen binnen dit onderzoek onbeantwoord blijven. Kwestie dat er simpelweg onvoldoende tijd is om in deze scope op te nemen.

Python is de winnaar uit dit onderzoek en zal gebruikt worden om de rest van deze proof of concept mee uit te werken.

### 4.3. Gebruik van bibl als Basis

Met Python als programmeertaal was het mogelijk om terug te denken aan bibl, dat uitgebreid werd bestudeerd (2.8). Zou het mogelijk zijn om bibl om te vormen tot een BibLaTeX-compatibele linter? Gezien de eerdere positieve ervaringen met bibl, werd besloten dit te onderzoeken. Een geldig BibLaTeX-bestand werd door bibl gehaald om de resultaten te analyseren. bibl crashte niet; het genereerde enkele linterwaarschuwingen op het bestand. Dit was een positieve uitkomst. Aangezien bibl open-source is en een goed gestructureerde codebasis heeft, was het eenvoudig om regels aan te passen, te verwijderen of nieuwe toe te voegen.

Nadat een geldig BibLaTeX-bestand door bibl was gelind, werd het duidelijk welke regels voor BibLaTeX niet langer van toepassing waren. De volgende regels werden als overtreding gedetecteerd:

- **D00** - Entry not in alphabetical order by key
- **M01** - Missing required field
- **M02** - Missing optional field
- **T00** - Non-ascii character
- **T01** - Non-standard whitespace
- **T02** - Whitespace end of line
- **T03** - Line length exceeding limit
- **U00** - Unknown entry type
- **U01** - Unrecognized field
- **E05** - Missing file

Regel D00 kon direct worden uitgeschakeld. De volgorde van de bronnen is in deze use case niet relevant; daarom staat deze regel op de lijst van te negeren regels. Het was logisch dat de regels M01 en M02 in eerste instantie als overtreding werden

beschouwd. BibTeX-bestanden gebruiken namelijk andere velden die niet langer nodig zijn in BibLaTeX, waardoor deze velden als ontbrekend werden gemarkeerd. De lijst van entry types en hun velden moeten zorgvuldig worden nagekeken. Dit geldt ook voor regel U00, aangezien BibLaTeX meer entry types kent dan BibTeX. Regel T00 kan worden verwijderd, aangezien BibLaTeX non-ascii characters ondersteunt. Regel T01 vereist meer aandacht; het aantal karakters dat als tab worden gebruikt, verschilt namelijk per omgeving. Voor deze proef werd de tab ingesteld op twee spaties. Omdat dit niet altijd het geval is, biedt bibl gelukkig een optie om dit eenvoudig in te stellen in de configuratie. Regel T02 verwijst naar spaties aan het einde van de regel. Nadat deze spaties waren verwijderd, was dit probleem opgelost. Deze regel kan blijven bestaan, afhankelijk van persoonlijke voorkeur. Regel T03 controleert het aantal karakters per regel. In deze use case werd hier minder aandacht aan besteed, aangezien het niet van groot belang was. Regel E05 is handig voor persoonlijke organisatie. Het verwijst naar een ontbrekend bestand dat aan een bron is gekoppeld. Aangezien de lijst met geldige referenties door de promotor werd geleverd, is het logisch dat sommige bestanden niet beschikbaar waren.

#### **4.3.1. Overbodige regels**

Zoals eerder vermeld, is regel T00 overbodig omdat deze controleert op non-ascii characters. In BibTeX mochten alleen ascii-characters worden gebruikt, maar in BibLaTeX zijn non-ascii characters toegestaan. Deze regel kan worden verwijderd. De overige regels kunnen wel nog als relevant beschouwd worden afhankelijk van de persoonlijke voorkeur.

Met deze veranderingen kon de uitwerkingsfase officieel van start gaan.

Prioriteit	Requirement Beschrijving
<b>Musts</b>	<ul style="list-style-type: none"> <li>- Elk item moet een sleutel hebben.</li> <li>- Elke BibLaTeX-sleutel moet uniek zijn.</li> <li>- Alle vereiste velden moeten aanwezig en niet leeg zijn.</li> <li>- Auteursnamen moeten in het juiste formaat zijn.</li> <li>- Data moeten in het formaat 'YYYY-MM-DD' zijn.</li> <li>- Speciale tekens te vervangen door hun LaTeX-commando's.</li> <li>- Paginabereiken moeten een ëm-dash"gebruiken, d.w.z. '–'.</li> <li>- article - Vereist: author, title, journaltitle, date.</li> <li>- book - Vereist: author editor, title, date, publisher.</li> <li>- inbook - Vereist: author editor, title, booktitle, date, publisher.</li> <li>- dataset - Vereist: author editor, title, date, url, urldate.</li> <li>- manual - Vereist: author editor, title, date.</li> <li>- misc (software) - Vereist: author editor, title, date.</li> <li>- online - Vereist: author editor, title, date, url, urldate.</li> <li>- inproceedings (conference) - Vereist: author, title, booktitle, date.</li> <li>- report (techreport) - Vereist: author, title, date, type, institution.</li> <li>- thesis (mastersthesis, phdthesis) - Vereist: author, title, date, type, institution.</li> </ul>
<b>Shoulds</b>	<ul style="list-style-type: none"> <li>- Geef de voorkeur aan "date"boven ijearën/of "month".</li> <li>- Aanbevolen velden ('<i>WARN_FIELDS</i>') moeten aanwezig en niet leeg zijn.</li> <li>- article - Aanbevolen: doi, volume, number, pages.</li> <li>- book - Aanbevolen: isbn.</li> <li>- inbook - Aanbevolen: isbn doi, pages.</li> <li>- manual - Aanbevolen: organization publisher, isbn doi url.</li> <li>- inproceedings - Aanbevolen: editor, eventtitle, isbn doi url.</li> <li>- report - Aanbevolen: doi url.</li> <li>- thesis - Aanbevolen: url.</li> </ul>
<b>Coulds</b>	<ul style="list-style-type: none"> <li>- BibLaTeX-keys moeten overeenkomen met de naam van de auteur.</li> <li>- Geef de voorkeur aan originele types, niet aan aliassen.</li> <li>- Verwijder tekstmarkering van URL.</li> <li>- Vermijd het citeren van tertiaire bronnen.</li> <li>- Citeer geen startpagina van een website.</li> <li>- Geef de voorkeur aan "journaltitle"boven "journal".</li> <li>- Geef de voorkeur aan institution"boven "school".</li> </ul>

**Tabel 4.1:** Geprioriteerde Functionele Vereisten met behulp van de MoSCoW-methode

JavaScript (Node) (ms)	Python (ms)	Rust (ms)	Notities
22,635	1.83867	-	Eerst 3 maal Python, vervolgens 3 maal JavaScript achter elkaar.
12,008	1.56621	-	
13,735	1.57471	-	
<b>Commando</b>			
27,095	1.76967	-	<code>node js/linter.js testData/test.bib &amp;&amp;</code>
10,8	1.44262	-	<code>python3 py/linter.py testData/test.bib</code>
9,386	1.23654	-	<code>python3 py/linter.py testData/test.bib</code>
11,311	1.36937	-	<code>&amp;&amp; node js/linter.js testData/test.bib</code>
<b>Notities</b>			
-	1.27763	9.669958	Enkel controle op duplicaten.
-	1.141	7.747833	
-	1.17904	8.652	

**Tabel 4.2:** De tabel met uitvoeringstijden van elke PoC in vergelijking met een andere. Eerst werden JavaScript en Python met elkaar vergeleken omdat deze als eerste waren opgesteld. Achteraf werd een laatst PoC in Rust opgezet. Wegens gebrek aan kennis en de tijd om deze kennis op te doen, kon er slechts één van beide controles geïmplementeerd worden. Vandaar dat bij de Rust testen genoteerd staat dat er enkel op duplicaten gecontroleerd werd. Gezien snelheid van belang was, werd JavaScript al uitgeschakeld door Python en was het niet nodig om deze nog eens te vergelijken met Rust.

# 5

## Uitwerkingsfase

Deze fase vond iets later plaats dan initiëel verwacht, maar verliep ook vlotter dan verwacht. Dat allemaal dankzij het ontdekken van `bibl`. Een gepaste *rebrand* naam vinden voor `bibl` was misschien geen prioriteit, maar er werd toch besloten er eentje te vinden. Na wat brainstormen werd de naam `bibla` bedacht. Waar `bibl` wellicht stond voor `bibTeX-Linter` of iets soortgelijk, zal `bibla` simpelweg refereren naar `Bi-bLaTeX`. Gezien `BibLa` minder leuk klonk, werd er gekozen voor `bibla`.

Nu deze lastige zaak afgehandeld is, werd er eerst een hele *rebrand* uitgevoerd op de fork van `bibl`; `bibla` was geboren.

In het begin was het zeer lastig om eigen regels toe te voegen, dit omdat het ondanks de grondige analyse die erop plaats vond, nog altijd een *vreemd* project was. Daarom werd er besloten om eerst bestaande regels te controleren en deze te vergelijken met de eigen lijst van functionele requirements. Zo konden er enerzijds taken worden geschrapt van zaken die reeds geïmplementeerd waren en kon er een overzicht gemaakt worden van de regels die aangepast diende te worden.

Op deze manier werd er toch al nuttige vooruitgang geboekt en schepte dit een vertrouwensband met de bestaande codebase. Eens enkele waren regels aangepast, werden er ook eigen regels opgesteld die nodig waren.

### 5.1. `bibla` - 2.0.5

Gezien `bibl` de voorganger is van `bibla`, werd er besloten om `bibla` vanaf versie 2.0.0 te laten starten. Dit met de insteek dat de grote basis van `bibl` zeker niet vergeten mag worden. Naast het uitschrijven van deze proef, werd er ook nog rustig verdergewerkt aan `bibla` zelf, waardoor sommige bugs die in deze codeblocks nog te zien zijn, mogelijks al een verandering ondergaan zijn in latere versies. Onder de

GitHub-issues<sup>1</sup> kunnen de gekende bugs geraadpleegt worden en kunnen er ook nieuwe bugs gemeld worden door onder andere de gebruikers. Gezien code altijd blijft evolueren, was het niet mogelijk om altijd de meest recente versie van de code te gebruiken. Dit is waarom er expliciet vermeld wordt dat er in deze proef gekeken wordt naar versie 2.0.5 omdat veranderingen zullen blijven gebeuren; wat ook het doel is van deze PoC.

### 5.1.1. Behouden regels

Deze sectie bevat de regels die volledig doorgekomen zijn vanuit bibl en dus geen veranderingen ondergaan zijn. Wel is het mogelijk dat hier regels bij vermeld staan waarvan er veranderingen zijn doordat het configuratiebestand veranderingen ondergaan heeft.

Zo zijn regels voor wat de Entries (E) betreffen: E01 (voornaam auteur mag niet worden afgekort), E02 (middennamen die al dan niet mogen worden afgekort), E03 (vermijd het gebruik van 'Et Al' in auteursveld), E04 (bestanden moeten relatief pad gebruiken), E05 (gelinkte bestanden die afwezig zijn), E06 (onjuist DOI-formaat) en E07 (onjuist ISBN-formaat) gelijk gebleven.

De Database (D) regels: D00 (Alfabetische volgorde van de bronnen), D01 (Preamble is de start van het document), D02 (Mogelijke duplicaten op basis van titel) zijn ook ongewijzigd gebleven.

De Missing (M) regels: M00 (Geen auteurs of redacteurs), M01 (Missend verplicht veld), M02 (Missend optioneel veld) zijn ook ongewijzigd gebleven wat de code betreft. Echter is hier wel de configuratie grondig aangepakt geweest voor M01 en M02, gezien HOGENT<sup>2</sup> zelf eigen voorkeuren heeft over welke bibliografische gegevens van een bron bijgehouden dienen te worden. Ook voor de Unrecognized (U) regels, zijn er enkel wijzigingen gebeurt in de configuratie.

De Unrecognized (U) regels zijn ook totaal onaangepast gebleven. Hetzelfde kan gezegd worden voor de Textual (T) regels; opnieuw met uitzondering voor T01 (aantal spaties die als indentatie gebruikt dienen te worden), waar de configuratie van werd aangepast; uiteraard met uitzondering voor regel T00 (enkel ASCII karakters) die weggehaald werd gezien deze niet meer relevant is.

### 5.1.2. Aangepaste regels

#### E00 - Sleutelformaat

Eén van de regels die aangepast diende te worden, was regel E00. Deze regel controleert het formaat van entry keys. Dit is vooral belangrijk om de consistentie te garanderen en om ervoor te zorgen dat er niet plots door inconsistenties een wijziging niet gevonden kan worden. De regel ziet er als volgt uit:

<sup>1</sup><https://github.com/MrClassicT/bibla/issues>

<sup>2</sup>[www.hogent.be](http://www.hogent.be)

```

1 @register_entry_rule('E00', 'Keys of published works should have
   format `AuthorYEARa`')
2 def key_format(key, entry, database):
3     """Raise a linter warning when entry key is not of format
4         `AuthorYEARa`.
5
6     E.g. an entry with values
7     ```
8     author = {Arthur B Cummings and David Eftekhary and Frank G
9               House},
10    date    = {2003-02-02}
11    ```
12    should have key `Cummings2003`.
13    If another entry with the same year and main author is present,
14    their keys should have formats `Cummings2003a` and
15    `Cummings2003b`.
16    This rule only applies when `date` and at least one author are
17    set.
18
19    :param key: The key of the current bibliography entry
20    :param entry: The current bibliography entry
21    :param database: All bibliography entries
22    :return: True if the current entry's key has the specified format
23            or
24            year or author are not specified, False otherwise.
25    """
26    if 'date' not in entry.fields or len(entry.persons.get('author',
27        [])) = 0:
28        return True
29    try:
30        author = entry.persons['author'][0]
31    except KeyError:
32        try:
33            author = entry.persons['editor'][0]
34        except KeyError:
35            return True
36    names = author.rich_prelast_names + [name for last_name in
37        author.rich_last_names for name in last_name.split('-')]
38    date = entry.fields['date']
39    year = re.search(r'\d{4}', date).group()
40

```

```

34     # Check for 'EtAl' in the key when there are more than 3 authors
35     if len(entry.persons.get('author', [])) ≥ 3 and 'EtAl' in
        key.lower():
36         correct_key_unicode = names[0] + 'EtAl' + year
37     # Check for two names in the key when there are exactly 2 authors
38     elif len(entry.persons.get('author', [])) = 2:
39         correct_key_unicode = "".join([str(name) for name in
        names[:2]]) + year
40     else:
41         correct_key_unicode = "".join([str(name) for name in names])
        + year
42
43     correct_key_ascii = unidecode(str(correct_key_unicode))
44     regex = re.compile(correct_key_ascii + r'[a-zA-Z]?')
45     return bool(regex.match(key))

```

**Listing 5.1.1:** bibla: E00 - Sleutels moeten het juiste formaat hebben. De aangepaste versie. Aanpassingen waren nodig vanwege het gebruik van het 'date' field in plaats van het voormalige 'year' field.

Gezien bij bibTeX de datum opgesplitst werd in year, month en day velden, diende deze regel aangepast te worden om rekening te houden met het date veld in de plaats (zie regel 21). Alsook diende er een **Reguliere Expressie (Regex)** toegevoegd te worden om rekening te houden met het veranderde datum formaat. Het jaar moest immers uit het date field gehaald worden gezien het geen veld op zich meer was, zie regel 32. Daarnaast werd er ook een controle toegevoegd voor wanneer er geen author aanwezig was en bijvoorbeeld enkel een editor. bibl hield er namelijk geen rekening mee dat er enkel editors aanwezig zouden kunnen zijn in een bepaalde bron. Bij het uitwerken van deze proef is er een bron in de bibliografie geraakt waarbij dit het geval was. Dit had als gevolg dat de linter crashte bij het linten van dit bibliografisch bestand. Daarnaast moest het ook mogelijk zijn om achternamen van twee auteurs te gebruiken in een sleutel of zelf 'EtAl' indien het er drie of meer waren. Ook hiervoor werden er enkele controles op punt gezet om hier rekening mee te houden, zie code-regel 35 tot en met 41.

### E08 - pages veld formaat

Deze regel leek niet te werken vanwege het veld waarop gecontroleerd werd. Er werd naar het page veld gezocht, maar het veld waar effectief naar gezocht diende te worden staat gekend als pages in BibLaTeX. Uiteindelijk werd deze regel een samenvoeging van de voormalige regel E08 (pagina formaat moet xx-yy zijn) en E09 (eindpagina dient groter te zijn dan de startpagina) uit bibl.



**E09 - Correct date formaat**

Zoals bij regel E00 (5.1.2) reeds besproken was, is het gebruik van data wat veranderd. Nu er geen apart veld meer gebruikt wordt voor year, month en day, dient er een extra controle te gebeuren om te zien of de datum van een bron wel goed geformatteerd is. Waar de originele regel van bibl enkel keek of de maand weldegeelijk in 'MMM' formaat was, dient er nu gekeken te worden dat het formaat gelijk is aan 'YYYY-MM-DD'. Deze regel vervangt de in bibl gekende regel E10 waarbij de maand in mmm-formaat diende te zijn.

```

1  if 'date' not in entry.fields:
2      return True
3  date = entry.fields['date']
4  regex = re.compile(r'^(\d{4})-(\d{2})-(\d{2})$')
5  match = regex.match(date)
6  if not match:
7      regex = re.compile(r'^(\d{4})-(\d{2})$')
8      match = regex.match(date)
9      if not match:
10         regex = re.compile(r'^(\d{4})$')
11         match = regex.match(date)
12         if not match:
13             return False
14         return True
15     year, month = map(int, match.groups())
16     if not (1 ≤ month ≤ 12):
17         return False
18     return True
19 year, month, day = map(int, match.groups())
20 if not (1 ≤ month ≤ 12):
21     return False
22 if not (1 ≤ day ≤ 31):
23     return False
24 return True

```

**Listing 5.1.2:** bibla: E09 - Correct date formaat. Deze regel werd gewijzigd gezien het vermelden van data gewijzigd is. Gebruikt nu date veld in plaats van de year, month en day velden. Dit is de logica. Ook hier is er op meerdere plaatsen weer een **Reguliere Expressie (Regex)** te zien. **Regex** zijn krachtig voor het controleren van bepaalde vaste tekstuele structuren. In deze use case zijn ze bijvoorbeeld zeer handig om het date veld op te splitsen. Bij de eerste check, zie regel 4, kan al direct gezien worden of het een '4-2-2' formaat is, of toch eerder een '2-2-4' formaat. Gezien zowel de maand als dag met twee getallen voorgesteld worden, is het belangrijk om hier een minimale vorm

van controle op uit te voeren. Zo kan het maandnummer nooit groter zijn dan 12 en zijn er maar maximaal 31 dagen tijdens de langste dagen. Hoewel dit niet elke fout kan tegengaan, is het wel handig om deze minimale controles uit te voeren om zoveel mogelijk garantie te bieden dat er geen grove fouten optreden op vlak van consistentie in de date velden. Regels 6 tot en met 18 voeren de controle uit om te zien wanneer de dag en/of maand moest ontbreken, om te zien dat ook dan het formaat zo goed mogelijk gerespecteerd wordt.

### 5.1.3. Nieuwe regels

#### E10 - Gebruik van Velden

Zoals bij de aanpassingen gemerkt kon worden, was het gebruik van andere velden toch wel een aanpassing. Echter is het niet te voorkomen dat er nooit nog de *oudere* velden gebruikt worden om data in op te slaan. Om de gebruikers zo goed mogelijk te ondersteunen, werd er een regel opgesteld waardoor de voorkeursvelden en hun aliassen konden gecontroleerd worden. De regel gaat op zoek naar oudere varianten/naamgevingen en suggereert dan het nieuwere veld aan in de plaats.

Er kan terug gedacht worden aan de `year`, `month` en `day` velden, waarvoor nu de voorkeur gaat om `date` te gebruiken.

De regel die dit controleert zag er als volgt uit:

```

1 def register_variant_rule(entry_type, field, variant):
2     rule_id = 'E10{}}}'.format(entry_type.capitalize(),
3                               field.capitalize())
4     message = "Entry type {} - use {} instead of {}!".format(entry_type,
5                                                               variant, field)
6
7     @register_entry_rule(rule_id, message)
8     def check_variant_field(key, entry, database, entry_type=entry_type,
9                             field=field, variant=variant):
10        """Raise a linter warning when a specific field is used instead of
11           its variant.
12
13           This function checks if the variant field is required and if the
14           specific field is present.
15           If these conditions are met, it suggests to use the variant field
16           instead of the specific field.
17
18           :param key: The key of the current bibliography entry
19           :param entry: The current bibliography entry
20           :param database: All bibliography entries

```

```

15 :param entry_type: Anchor variable to pass the local variable
    `entry_type` from outer scope
16 :param field: The field that is being checked
17 :param variant: The field that should be used instead
18 :return: False if the specific field is used instead of its variant,
    True otherwise
19 """
20 if entry.type == entry_type and field in entry.fields:
21     return False
22 else:
23     return True

```

**Listing 5.1.3:** bibla: Nieuwe Regel | E10 - Gebruik aanbevolen velden in plaats van gelijkaardigen.

Net zoals bij de originele regels van bibl, wordt er ook bij de nieuwe aandacht besteed aan een correcte documentatie van de code. Het is namelijk dankzij deze zorgvuldige documentatie dat gebruikers en bijdragers (Engels: Contributors) in staat zijn om te begrijpen wat er gaande is en hun bijdrage te leveren zonder onnodig tijd te moeten verspillen aan het ontcijferen van bestaande code.

Bij deze regel kan ook duidelijk gezien worden dat er inspiratie gehaald werd bij een reeds bestaande regels die op een zeer gelijkaardige manier te werk gaan. De regels M01 en M02 zijn een grote inspiratie geweest. De regels zijn simpel maar zeer efficiënt. Er wordt een lijst *dictionary* bijgehouden waarbij de 'oude' velden als *key* dienen en de 'nieuwe' als *value*.

### E11 - Gebruik van Aliassen

Binnen BibLaTeX zijn er vaak vele varianten beschikbaar voor bepaalde types bronnen, deze varianten vereisen dan dezelfde fields als het *originele* type. Om consistentie te garanderen, gaat de voorkeur uit om altijd het *originele* type te gebruiken in plaats van één van de aliassen. Deze methode werd op een gelijkaardige manier uitgewerkt als E10, M01 en M02 waar er gekeken wordt naar de bepaalde velden die al dan niet gebruikt worden. Het enige verschil is dat er hier niet naar de velden gekeken worden van een *entry* maar wel naar het type van de *entry*.

### E12 - Geen startpagina's in URL veld

Startpagina's zijn te algemeen en verwijzen meestal niet naar een geldige bron, daarom is het van groot belang dat de correcte URL opgeslagen wordt, zodat ook na het raadplegen van een bron voor de eerste keer, dezelfde bron eenvoudig achterhaald kan worden. Ook voor in de bibliografie is het van groot belang dat dit correct opgeslagen wordt. Deze regel wordt nagekeken door opnieuw gebruik te maken van [Regex](#). De code binnen deze regel ziet er als volgt uit:

```

1 if 'url' not in entry.fields:
2     return True
3 url = entry.fields['url']
4 regex = re.compile(r'^(https?://)?[^\s/]+/?$')
5 if regex.match(url):
6     return False
7 return True

```

**Listing 5.1.4:** bibla: Nieuwe Regel | E12 - Gebruik geen startpagina's in het URL veld, logica.

Op regel 4, in de **Regex** kan er gezien worden dat er gekeken wordt naar hoe een typische URL eruit kan zien, rekeninghoudende met al dan niet optionele componenten uit een URL.

### E13 - Enkel kritische pad in URL veld

Naast dat er niet enkel startpagina's gebruikt mogen worden, is het ook ongewenst dat de URL onnodig lang is. Om deze voorvallen te voorkomen, wordt er een check gedaan op enkele karakters die typisch voorkomen in *te lange* URLs. Hieronder worden bijvoorbeeld URL's verstaan waarin er een directe verwijzing gebeurt naar een specifieke zin uit een webpagina. Het wordt met behulp van een opsomming van deze karakters gecontroleerd zoals hieronder te zien is.

```

1 if ('#' or '?' or '%') in url:

```

**Listing 5.1.5:** bibla: Nieuwe Regel | E13 - Gebruik enkel het kritische pad in het URL veld, logica.

### M03 - Speciale karakters

Deze regel zal mogelijks later van naam veranderen, gezien het niet direct onder de meest intuïtieve categorie staat (M). 'M' wordt doorgaans gebruikt voor 'Missing' of ontbrekende zaken aan te duiden. Al kan er gediscussieerd worden of dit dan niet net toch de juiste categorie zou zijn, gezien het om een ontbrekend karakter gaat. Speciale karakters dienen namelijk vervangen te worden door het commando om ze te genereren. Het commando is doorgaans gewoon een `\` voor het karakter plaatsen. Dit zorgt ervoor dat LaTeX het speciale karakter niet gaat zien als een commando dat het moet uitvoeren.

```

1 @register_entry_rule('M03', 'Special characters should be replaced
   by the command to generate them: %, &, $, #, _, \, ~, ^, |')
2 def check_special_characters(key, entry, database):
3     """Raise a linter warning when a field contains special
   characters that should be replaced.
4     :param key: The key of the current bibliography entry
5     :param entry: The current bibliography entry

```

```

6     :param database: All bibliography entries
7     :return: True if no field contains special characters, False
           otherwise.
8     """
9     special_chars = ['%', '&', '$', '#', '_', '\\', '~', '^', '|']
10
11    for field in entry.fields.values():
12        if any(char in field for char in special_chars):
13            for char in special_chars:
14                if char in field and field[field.index(char) - 1] ≠
                    '\\':
15                    return False
16    return True

```

**Listing 5.1.6:** bibla: Nieuwe Regel | M03 - Speciale karakters dienen met commando gegenereerd te worden.

Hoewel deze versie gebruikt wordt in versie 2.0.5 van bibla, zit hier nog een kleine *bug* in. Het is namelijk zo dat deze regel elk veld zal controleren op speciale karakters. Hoewel dit over het algemeen goed is, gooit het hierdoor ook foutieve waarschuwingen op het url veld.

#### 5.1.4. Diverse aanpassingen

Ondanks dat `bibl` een uitstekende start was voor deze [Proof of Concept](#), waren er nog een paar zaken ter verbetering vatbaar. Foutafhandelingen waren hier een deel van. Ze dienen niet direct als een regel gezien te worden, maar ze kunnen wel voorkomen door regels die overtreden worden. Zo was bijvoorbeeld het bevatten van duplicate entry keys een reden waardoor de applicatie het soms begaf. Dit werd opgelost door een 'exception handler', ook wel foutafhandelaar genoemd, toe te voegen. Hoewel dan niet het hele bestand gelind werd, vanwege de vroegtijdige fout in het programma, wordt er nu tenminste wel op een minder schokkerende wijze getoond wat er mis is en gewijzigd dient te worden. Hetzelfde werd voorzien voor wanneer er gewoon lege entries waren.

```

1  try:
2      bib_data = parse_file(bibliography, macros=MONTH_NAMES)
3  except BibliographyDataError as e:
4      # Extract the key from the error message
5      match = re.search(r'repeated bibliograhpy entry: (.*)', str(e))
6      if match:
7          duplicate_key = match.group(1)
8          print(f"{bibliography} D03: Duplicate entry with key
                '{duplicate_key}'")

```

```
9     else:
10         print(f"Warning: {e}")
11     return []
12 except pybtex.scanner.TokenRequired as e:
13     print(f"E00: {e}")
14     return []
15 except Exception as e:
16     print(f"Error: {e}")
17     return []
```

**Listing 5.1.7:** bibla: Diverse aanpassing | Foutafhandelingen om de tool properder te laten werken. In dit codeblock, genomen uit `lint.py`, kan er gezien worden dat er aan extra foutafhandelingen gedaan wordt. De regels die hier overtreden worden, staan op dit ogenblik vast gecodeerd en zijn minder dynamisch dan doorheen de rest van de applicatie gewend is, maar ze zijn afgestemd geweest en als de testen als betrouwbaar gezien kunnen worden, zou dit kloppen met wat de oorzaak achterliggend effectief is. Dit was de meest eenvoudige en duidelijke manier om de gebruiker te laten weten wat er mis is zonder kostbare tijd te verliezen voor het verder ontwikkelen van andere delen van de applicatie.

### 5.1.5. Configuratie

De standaardconfiguratie van bibl was een zeer goede basis om van verder te gaan voor bibla. Maar zoals al vaker aangegeven, zijn er verschillen tussen bibTeX en bibLaTeX die ervoor zorgen dat er aanpassingen dienen te gebeuren. Op de configuratie van bibl<sup>3</sup> zijn er zaken veranderd, maar ook extra toegevoegd. Zoals voor bijvoorbeeld regel E10 (5.1.3) waar er gekeken werd naar bepaalde velden waarbij er een voorkeur is om er een synoniem van te gebruiken. Waarvan hieronder het stukje configuratie in YAML-formaat.

```
1 alternate_fields:
2   # preferred: [other]
3   date: [year] # Month and day will not be used alone, so when we
4     just check for the year, that'll be fine.
5   journaltitle: [journal]
6   institution: [school]
```

**Listing 5.1.8:** bibla: Configuratie | E10 - Gebruik aanbevolen velden in plaats van gelijkaardigen. De overige configuratie kan geraadpleegd worden op de bibla repository in `bibla.yml`<sup>4</sup>, wat de algemene configuratie voor bibla bevat.

<sup>3</sup><https://gitlab.com/arnevdk/bibl/-/blob/master/bibl/bibl.yml>

<sup>4</sup><https://github.com/MrClassicT/bibla/blob/master/bibla/bibla.yml>

### 5.1.6. Pipelines

`bibl` werkte met GitLab, `bibla` werkt met GitHub. Het is daarom niet onlogisch dat er nog enkele andere zaken aangepast dienen te worden om alles vlot te laten verlopen. Hierbij denken we vooral aan de pipelines die gebruikt worden. `bibl` bevat slechts één pipeline bestand<sup>5</sup>, een continuous integration. Al gebeurt er in deze pipeline ook een *deploy* van een statische website die alle linterregels bevat. Op GitHub worden pipelines gekend onder *workflows*, wat reflecteert naar wat een pipeline effectief inhoudt. Gezien de *workflows* iets anders zijn dan de gewone pipelines op GitLab, diende er enkele wijzigingen te gebeuren. Zo diende de pipelines te komen in een folder `.github/workflows` om herkend te worden. Hierin werd er een omgevormde CI-pipeline geplaatst, gebaseerd op de originele van `bibl`.

#### CI-pipeline

```
1 name: CI Workflow
2
3 on:
4   push:
5     branches:
6       - master
7
8 jobs:
9   build:
10    runs-on: ubuntu-latest
11    strategy:
12      matrix:
13        python-version: ["3.12"]
14    steps:
15      - uses: actions/checkout@v4
16
17      - name: Set up Python ${ matrix.python-version }
18        uses: actions/setup-python@v4
19        with:
20          python-version: ${ matrix.python-version }
21
22      - name: Install dependencies
23        run: |
24          python -m pip install --upgrade pip
25          pip install
26          git+https://${ secrets.BIBLA_GITHUB_TOKEN }@github.com/mrclassict/bibl
```

<sup>5</sup><https://gitlab.com/arnevdk/bibl/-/blob/master/gitlab-ci.yml>

```
26     pip install --upgrade setuptools wheel
27     pip install -e .[dev]
28     pip install flake8
29
30 - name: Static Analysis with Flake8
31   run: |
32     mkdir out
33     flake8 --append-config .flake8 --ignore=E501 | tee
34       out/lint_result.txt
35     num_warnings=$(wc -l < out/lint_result.txt)
36     echo "$num_warnings warnings!"
37     echo "Generating badge ..."
38     anybadge -v=$num_warnings -f=out/flake8.svg -l=flake8
39       10=green 1000=orange 10000=red
40     cat out/lint_result.txt
41     if [ $num_warnings -gt 0 ]; then exit 1; fi
42
43 - name: Upload artifacts
44   uses: actions/upload-artifact@v4
45   with:
46     name: flake8-results
47     path: |
48       out/flake8.svg
49       out/lint_result.txt
```

**Listing 5.1.9:** bibla: CI-pipeline | [Continuous Integration \(CI\)](#) pipeline overzicht

Bij het analyseren van deze pipeline, kan er gezien worden dat er vier zaken gebeuren:

1. De juiste Python-versie installeren (in dit voorbeeld 3.12)
2. Dependencies installeren
3. De code linten met flake8
4. Artifacts uploaden

Deze stappen gebeuren op een virtuele machine op de GitHub servers en ze gebeuren elke keer dat er een *push* gebeurt naar de 'master' branch. Hoewel in deze pipeline net gelijk bij `bibla` Flake8 gebruikt wordt om de code te linten, werd er ondertussen geleerd dat Ruff wellicht een interessantere optie is. Naar de toekomst toe kan het interessant zijn om hier eens naar te kijken en eventueel over te schakelen van linting-tool.



### CD-pipeline

Om ervoor te zorgen dat bij elke *release* (of uitgave in het Nederlands) *bibla* eenvoudig en consistent naar de pakketmanager gestuurd kon worden, werd er ook een CD-pipeline voorzien. Het opzetten van deze pipeline ging dankzij de templates die GitHub voorziet in minder dan vijf minuten! Het enige dat zelf diende te gebeuren, was een **API**-token (een vorm van zowel identificatie en authenticatie) voorzien en de CD-pipeline-template deed de rest.

```
1 # This workflow will upload a Python Package using Twine when a
   # release is created
2 # For more information see:
   # https://docs.github.com/en/actions/automating-builds-and-tests/building-and-te
3
4 # This workflow uses actions that are not certified by GitHub.
5 # They are provided by a third-party and are governed by
6 # separate terms of service, privacy policy, and support
7 # documentation.
8
9 name: Upload Python Package
10
11 on:
12   release:
13     types: [published]
14
15 permissions:
16   contents: read
17
18 jobs:
19   deploy:
20
21     runs-on: ubuntu-latest
22
23     steps:
24     - uses: actions/checkout@v4
25
26     - name: Set up Python
27       uses: actions/setup-python@v3
28       with:
29         python-version: '3.x'
30     - name: Install dependencies
31       run: |
32         python -m pip install --upgrade pip
```

```
33     pip install build
34     pip install markdown
35
36     - name: Build package
37       run: python -m build
38
39     - name: Publish package
40       uses:
41         pypa/gh-action-pypi-publish@27b31702a0e7fc50959f5ad993c78deac1bdfc29
42       with:
43         user: __token__
44         password: ${{ secrets.PYPI_API_TOKEN }}
```

**Listing 5.1.10:** bibla: CD-pipeline | [Continuous Deployment \(CD\)](#) pipeline overzicht

### 5.1.7. Testen

Wat de testen betreft bij deze [Proof of Concept](#), werd er zich gebaseerd op de manier waarop `bibl` gebruikt wordt. Er werden `test.bib`-bestanden aangemaakt die elk voor een bepaalde regel verantwoordelijk dienden te zijn. Zo kon er tijdens het ontwikkelen getest worden of de regel al dan niet werkt zoals verwacht. Uiteindelijk bleek wel dat deze `test.bib`-bestanden wat vaker revisies mochten ondergaan om accuraat en representatief te blijven. [AI](#) heeft hier ook deels geholpen om deze te genereren, maar gezien er wel relevante bronnen en input nodig waren, was het niet altijd de beste tactiek. Vooral omdat online heel veel BibTeX-bronnen staan en die dus ook vaker aanbevolen worden door [AI](#).

Testen werden manueel uitgevoerd door het `bibla lint test.bib` commando uit te voeren.

# 6

## Conclusie

Deze bachelorproef richtte zich op het ontwikkelen van een proof of concept voor een BibLaTeX-compatibele linter, gebaseerd op de bestaande `bibl`-tool, een linter voor BibTeX. Door de keuze voor Python als programmeertaal en de solide structuur van `bibl`, is een functionele en effectieve linter gerealiseerd die BibLaTeX-bestanden kan analyseren en valideren.

Tijdens de ontwikkeling en testen van deze linter is gebleken dat de tool over het algemeen betrouwbaar en consistent is en nauwkeurige resultaten levert. Dankzij feedback van de promotor en medestudenten zijn diverse bugfixes doorgevoerd, wat heeft bijgedragen aan de verbetering van de tool en heeft geleid tot minder foutieve detecties. Hoewel er nog enkele mankementen zijn, worden deze actief gemeld via GitHub 'issues' en voortdurend aangepakt. Dit iteratieve proces zorgt ervoor dat de linter steeds verder wordt verbeterd en dat er ruimte blijft voor het toevoegen van extra functionaliteiten in de toekomst. Dit toont de doeltreffendheid van de linter en de waarde van een iteratief ontwikkelingsproces.

### 6.1. Nog uit te werken requirements

Naast alle uitgevoerde functionaliteiten, zijn er enkele die niet afgerond zijn. Een functionaliteit die prioriteit zou mogen krijgen, is het controleren of auteurs/redacteuren hun namen in het correcte formaat hebben geschreven. Gezien de manier waarop Pybtex deze uitleest, bleek dit complexer dan verwacht en werd deze feature daarom als out of scope beschouwd voor deze POC. Daarnaast moet er nog een methode worden gevonden om te controleren of tertiaire of illegale bronnen gebruikt worden, zodat ook deze afgeraden kunnen worden. Een andere kleinere feature die wegens tijdsgebrek niet volledig klaar is, is een online lijst van alle regels. Deze zou tijdens het uitvoeren van de pipelines aangemaakt worden en achteraf

gepubliceerd worden. GitHub Pages<sup>1</sup> is een optie die hiervoor gebruikt zou kunnen worden.

## **6.2. Was de proof of concept een succes?**

Er kan geconcludeerd worden dat het resultaat van dit project een groot succes is. De linter functioneert naar behoren en kan effectief worden ingezet als analysetool voor BibLaTeX-bestanden; zeker binnen de HOGENT-omgeving, aangezien de regels de voorkeuren van HOGENT<sup>2</sup> respecteren. Deze tool stelt gebruikers in staat om consistentie en validiteit in hun bronvermeldingen te waarborgen, wat essentieel is voor academische werken. Dankzij de bekendmaking van de tool door de promotor en docent aan HOGENT aan medestudenten, kon de tool al getest worden. Eén van deze medestudenten heeft een schermafbeelding voorzien waaruit de effectiviteit van `bibla` aangetoond kan worden, zie Figuur 6.1.

Samenvattend biedt `bibla` een meer uitgebreide en gedetailleerde set regels die beter zijn afgestemd op de eisen van BibLaTeX in vergelijking met `bibl`. De aanvullende en aangepaste regels (zie Sectie 5.1) zorgen voor een grotere nauwkeurigheid en consistentie in de verwerking van bibliografische vermeldingen. Door het volgen van deze regels kunnen gebruikers ervoor zorgen dat hun BibLaTeX-bestanden voldoen aan de huidige standaarden en best practices die voor deze use case werden opgelegd door HOGENT.

## **6.3. Toekomstvisie**

Het gebruik van `bibla` als de standaard bronvermeldingsanalysetool binnen HOGENT is een veelbelovende stap. Door de implementatie van `bibla` zullen studenten en docenten profiteren van een uniforme en betrouwbare methode om BibLaTeX-bestanden te controleren. Dit zal de kwaliteit van academische werken verbeteren en de werkdruk verlagen doordat veelvoorkomende fouten automatisch worden gedetecteerd.

De ontwikkeling van `bibla` staat echter niet stil. Nieuwe features en regels zullen blijven worden toegevoegd om de functionaliteit van de linter uit te breiden. Dit omvat bijvoorbeeld het toevoegen van extra types bronvermeldingen en het verbeteren van bestaande regels om nog nauwkeuriger te zijn. Daarnaast zullen eventuele nieuwe bugs worden aangepakt en opgelost om ervoor te zorgen dat de linter zo feilloos mogelijk kan werken.

Met `bibl` als stevige fundering was het mogelijk om `bibla` te ontwikkelen. De modulaire en uitbreidbare architectuur van `bibl` heeft bijgedragen aan een soepel ontwikkelproces en biedt een solide basis voor verdere uitbreidingen. Deze funde-

---

<sup>1</sup><https://pages.github.com/>

<sup>2</sup>[www.hogent.be](http://www.hogent.be)

ring zorgt ervoor dat `bibla` niet alleen nu, maar ook in de toekomst een waardevol hulpmiddel zal blijven voor de academische gemeenschap van HOGENT.

#### 6.4. Aanbevelingen

Voor toekomstige ontwikkelaars en gebruikers van `bibla` zijn de volgende aanbevelingen relevant:

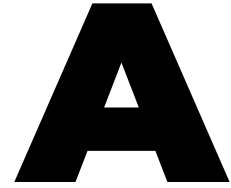
- **Documentatie:** Zorg voor uitgebreide documentatie voor gebruikers om hen te helpen het meeste uit de tool te halen.
- **Feedback Verzamelen:** Blijf feedback verzamelen van gebruikers om de tool verder te verbeteren en aan te passen aan hun behoeften.
- **Community Betrokkenheid:** Moedig betrokkenheid van de academische gemeenschap aan om bij te dragen aan de ontwikkeling van nieuwe regels en eventuele features.
- **Testen:** Besteed extra aandacht aan het ontwikkelen van betere testen. Zo kan iedereen beter ondersteund worden en zal dit ervoor zorgen dat de tool accuraat en representatief blijft werken, ook na de toevoeging van nieuwe features.
- **Migratie naar Ruff?:** Volgens Astral (2024) zou Ruff vele malen sneller zijn dan Flake8. Er kan een pipeline opgezet worden met Ruff om dit effectief te testen om dan zo te bepalen of het eventueel gunstig kan zijn om te migreren naar Ruff, in plaats van Flake8 te gebruiken zoals `bibl` doet.

Met deze punten zal `bibla` zich kunnen blijven ontwikkelen en bijdragen aan de verbetering van academische publicaties zowel binnen als buiten HOGENT.



Figuur (6.1)

Deze afbeelding toont de **capaciteiten** van **bibla**. Een gebruiker heeft dit gedeeld, op de afbeelding kan het rode gezien worden als wat weggehaald werd en het groene als wat er in de plaats bijgekomen of ter vervanging is. Hoewel de afbeelding wat enigszins onduidelijk is, kunnen de aantal veranderingen die dienden te gebeuren wel waargenomen worden. Dit illustreert het **aantal fouten** waarvan docenten - **per student** - bespaard zullen worden om ze iedere keer zelf aan te moeten halen en anderzijds heeft deze gebruiker nu een correctere bibliografie.



# Onderzoeksvoorstel

Het onderwerp van deze bachelorproef is gebaseerd op een onderzoeksvoorstel dat vooraf werd beoordeeld door de promotor. Dat voorstel is opgenomen in deze bijlage.

## A.1. Introductie

LaTeX ( $\text{\LaTeX}$ ) uitspraak: la-tech; is een populair softwaresysteem in de wetenschappelijke wereld omdat het uitblinkt in het zetten van technische documenten, en beschikbaar is voor bijna alle computersystemen (Oetiker e.a., 2023).

Niet alleen in de wereld van de wetenschap wordt LaTeX gebruikt. Ook studenten op hogescholen en universiteiten maken er gebruik van voor het schrijven van bachelor- en/of masterproeven.

Bij het schrijven van, al dan niet wetenschappelijke, teksten is het uitermate belangrijk om aan een correcte vorm van bronvermelding te doen.

Binnen LaTeX zijn er verschillende manieren om dit aan te pakken. Eén van deze manieren is met behulp van BibLaTeX, een package speciaal gebouwd voor deze taak.

Studenten te HOGENT dienen gebruik te maken van deze combinatie bij het schrijven van hun bachelorproef. Ondanks dat lectoren veel moeite steken in het bondig toelichten van het correcte gebruik, worden er nog veel fouten gemaakt op het correct bijhouden van bronnen. Op deze groep zal deze bachelorproef zich focussen, met behulp van een statische analysetool, ook wel linter genoemd, zouden er al veel van de herhalende fouten voorkomen kunnen worden. Een linter is een programma dat broncode of gestructureerde dataformaten kan controleren op stijl, syntax en logische fouten (Kamunya, 2023).

Het zou dus uitermate geschikt zijn om de studenten een linter te laten gebruiken om hen zo te helpen bij het voorkomen of opsporen van de gemaakte fouten. Zo

dienen lectoren hen niet keer op keer op dezelfde fouten te wijzen.

BibLaTeX is voortkomende uit BibTeX en biedt meer opties om bibliografieën en citaten te configureren (Cassidy, 2013). Hoewel er voor BibTeX reeds een linter bestaat, is deze niet compatibel met BibLaTeX.

Het doel van deze bachelorproef is om een proof of concept, analyse & de software-architectuur uit te werken voor een BibLaTeX linter en er een prototype voor te schrijven in een passende programmeertaal.

Concreet betekent dit:

- De lijst van gewenste functionele en niet-functionele requirements aanvullen en structureren naar prioriteiten
- De werking van bestaande linters bestuderen als inspiratiebron
- Een gemotiveerde keuze maken voor de te gebruiken programmeertaal en eventuele libraries
- Een prototype met een minimale set van linting-regels implementeren
- Unit tests schrijven met zo compleet mogelijke code coverage
- CI-pipeline opzetten voor packaging en testing
- Documentatie schrijven voor het gebruik en uitbreiden van de linting-regels

Als eindresultaat voor deze bachelorproef zal er een open-source prototype opgesteld worden waaraan vrijwilligers verder kunnen werken.

## A.2. State-of-the-art

Op het ogenblik van het schrijven van dit bachelorproefvoorstel, zijn er nog geen *optimale* BibLaTeX-Linters beschikbaar. De enige beschikbare linter die bestaat voor BibLaTeX voor dit moment, staat op een GitHub-repository van Pez Cuckow. Deze zou wel werkende zijn, maar lijkt niet zo optimaal op het eerste zicht. Er is dus duidelijk mogelijkheid tot verbetering. De BibLaTeX-Linter van Pez Cuckow is geschreven in Python en heeft een webinterface (Cuckow, 2022). Tijdens het onderzoek naar de werking hiervan, werd er niet in geslaagd om deze werkende te krijgen. Dit is eventueel een opdracht voor tijdens de effectieve uitvoer. Hierdoor mag ook al de conclusie getrokken worden dat de bestaande Linter voor BibLaTeX dus nog zeker niet optimaal is op vlak van *gebruiksvriendelijkheid*, gezien de nodige tijd om hem werkende te krijgen.

## A.3. Methodologie

Dit onderzoek zal opgesplitst worden in meerdere fasen. Voor het uitvoeren van dit onderzoek zal wellicht een computer het enige zijn dat nodig is, met uitzondering



op technisch inzicht en logisch redeneren. Het eindresultaat voor dit onderzoek zal een open-source prototype zijn van een BibLaTeX-linter waaraan vrijwilligers verder kunnen werken.

Voor het onderzoek zouden 14 weken ter beschikking gesteld worden. Het is de bedoeling om de fasen te verdelen over deze weken en om alsnog een week of twee over te houden ter reserve voor moesten er fouten bij de schattingen ingeslopen zijn. De inschatting van de benodigde tijd zal bepaald worden op basis van het aantal onderzoek dat er moet gebeuren en van het verwachte resultaat aan het einde van deze fase.

### **A.3.1. Fase 1: Literatuurstudie**

Hier wordt er onderzoek gedaan naar reeds bestaande linters, al dan niet gerelateerd aan LaTeX. Er zal gekeken worden naar de programmeertalen waarin deze gemaakt zijn, hun werking, specificaties en functionaliteiten. Daarnaast mag er ook niet vergeten worden om te kijken hoe een CI-pipeline effectief in zijn werk gaat, zodat het mogelijk is om de linter op een efficiënte manier te integreren in workflows. Indien er andere relevante zaken opduiken die belang kunnen hebben aan het onderzoek, zullen deze ook worden opgenomen om te onderzoeken. Op deze manier zal er geprobeerd worden een zo volledig mogelijk onderzoek te voeren. Een kijk naar hoe een linter *echt* werkt. Deze kennis zal gebruikt worden als inspiratiebron voor het uiteindelijke proof of concept dat opgesteld zal worden in een latere fase. Een persoonlijke deadline wordt op 1 mei 2024 geplaatst. Hoewel de focus op deze fase het grootst zal zijn gedurende de eerste twee weken, is het een fase die parallel blijft doorlopen gezien er altijd extra info nodig kan zijn doorheen de andere fasen. Het resultaat van deze fase is een (informeel) document waarin alle ondervindingen, onderzoeken en handige informatiebronnen bijgehouden zal worden over reeds bestaande linters en andere zaken die van pas komen tijdens zowel het onderzoeken als het ontwikkelen van deze proof of concept. Het document zal een bron van kennis zijn die gebruikt kan worden in verdere fasen.

### **A.3.2. Fase 2: Technische Analyse en Experimenten**

In deze fase is het doel om de lijst van gewenste functionele en niet-functionele requirements aan te vullen en deze te structureren naargelang hun prioriteit. Onder andere de keuze van de programmeertaal, eventuele libraries en andere softwaretools die gebruikt kunnen en zullen worden voor het maken en opzetten van de proof of concept zullen hier gebeuren. Een eerste poging rond een simpele CI-pipeline zal hier mogelijk ook plaatsvinden. Dit om de werking en configuratie ervan zo goed mogelijk te begrijpen. Fase 2 zal plaatsvinden gedurende de maand maart. Het resultaat is een overzicht van zowel de functionele als de niet-functionele requirements, op prioriteit gerankschikt. Alsook zal er een soort van

sprint planning opgesteld worden om een overzicht te bewaren in wat er wanneer te verwachten valt tijdens de volgende fase.

### **A.3.3. Fase 3: Software Ontwikkeling (PoC - Proof of Concept)**

In deze fase gebeurt het uitdagende werk. De proof of concept zal ontwikkeld worden. Een werkend prototype zal opgesteld worden, inclusief testen om de werking te garanderen en om het eenvoudig uitbreidbaar te maken. Niettemin zal er ook uitbundige documentatie geschreven worden, zodat alles wat er gebeurt duidelijk is voor elke vrijwilliger die een bijdrage wenst te leveren. Er wordt verwacht dat in deze fase enkel gefocust dient te worden op de effectieve creatie van de linter met behulp van gevonden informatie en bronnen. Echt opzoekwerk zou minimaal moeten zijn in deze fase. Aan het einde van deze fase zal er dus een werkend prototype beschikbaar staan op een GitHub-repository van de auteur van dit onderzoek, Tristan Cuvelier. Er wordt verwacht dat deze fase wel enige tijd in beslag zal nemen gezien het verwachte resultaat en met een marge die ruim genoeg is voor problemen die opduiken tijdens het ontwikkelen van de software. Deze fase begint eind maart en duurt tot 17 mei.

### **A.3.4. Algemeen + Fase 4**

Naast deze fasen is het ook nodig om de bachelorproef zelf uit te schrijven. Het is belangrijk dat dit doorheen de fasen heen ook al gebeurt. Zo kunnen de laatste paar weken gebruikt worden om alles op punt te zetten om de finale versie in te dienen. Deze fase start dus al gelijk met fase 1, op 19 februari waarbij er gelijde-lijk aan doorheen de andere fasen aan het eind dossier gewerkt wordt. Deze fase heeft als deadline 21 mei gekregen, zo zijn er nog enkele reserve dagen voor de effectieve deadline van 24 mei om 12u 's middags. Het resultaat van deze fase is, zoals verwacht, een grondig uitgewerkte bachelorproef.

## **A.4. Verwacht resultaat, conclusie**

Als resultaat voor deze bachelorproef wordt er verwacht dat er een open-source repository van een BibLaTeX-linter ter beschikking komt. Deze zal de studenten en lectoren van HOGENT helpen om minder fouten te maken op vlak van bronvermeldingen binnen het schrijven van hun papers. Ook externe personen buiten HOGENT zullen gebruik kunnen maken van deze linter. Hoewel de linter slechts een prototype zal zijn, is het de bedoeling dat deze als goede basis kan dienen om op relatief eenvoudige wijze aan verder te werken. Dit alles met als ultiem doel de ideale linter te worden voor BibLaTeX. Gezien dat er voorlopig slechts één BibLaTeX-linter online staat die niet snel werkend te krijgen is, zal dit ongetwijfeld een grote meerwaarde bieden aan iedereen die gebruik maakt van BibLaTeX.

Er wordt verwacht dat dit een uitdagende opdracht zal worden waaruit veel geleerd kan worden, 'trial and error' samen met grondig onderzoek zullen de basis

zijn. Als softwaredeveloperstudent is het een interessante uitdaging om zelf een linter te schrijven voor iets dat een student toegepaste informatica slechts enkele keren gebruikt doorheen de schoolcarrière. Pipelines zijn niet ongekend, maar blijven wel iets minder voorkomend in de tak van development in de richting.

# Bibliografie

- Astral. (2024). *Ruff - Lint at lightspeed*. Verkregen maart 23, 2024, van <https://astral.sh/ruff>
- Borges Late, J. (2021, januari 9). *Making your own javascript linter: Part 1*. Verkregen maart 24, 2024, van <https://medium.com/codex/making-your-own-javascript-linter-part-1-ee9f91dc49d8>
- Cassidy, J. (2013, juli). *Getting started with BibLaTeX*. Overleaf. Verkregen december 7, 2023, van [https://www.overleaf.com/learn/latex/Articles/Getting\\_started\\_with\\_BibLaTeX](https://www.overleaf.com/learn/latex/Articles/Getting_started_with_BibLaTeX)
- Cuckow, P. (2022, december 17). *BibLaTeX Linter* (P. Cuckow, Red.). Verkregen december 7, 2023, van <https://github.com/Pezmc/BibLaTeX-Linter>
- Hennigan, L. (2024, februari 1). *What Is A Kanban Board? The Ultimate Guide* (C. Bottorff & R. Watts, Red.). Forbes. Verkregen april 28, 2024, van <https://www.forbes.com/advisor/business/software/what-is-kanban-board/>
- Kamunya, T. (2023, januari 25). *Should You Start Linting Your Code? [+5 Linting Tools]*. Geekflare. Verkregen december 7, 2023, van <https://geekflare.com/linting-tools/>
- Kime, P., Wemheuer, M., & Lehman, P. (2024, maart 21). *The biblatex Package: Programmable Bibliography and Citations*. Versie 3.20. Verkregen mei 4, 2024, van <https://ctan.mines-albi.fr/macros/latex/contrib/biblatex/doc/biblatex.pdf>
- Klabnik, S., & Nichols, C. (2022, december). *The Rust Programming Language* (2de ed.). No Starch Press.
- Knuth, D. E. (1984). *Knuth: Donald Ervin* (2de ed., Deel A). Addison Wesley.
- Lampport, L. (1994). *LATEX: A document preparation system ; user's guide and reference manual ; [updated für LATEX 2E]* (2de ed.). Addison-Wesley.
- Loubser, N. (2021). *Software Engineering for Absolute Beginners: Your Guide to Creating Software Products*. Apress. <https://doi.org/10.1007/978-1-4842-6622-9>
- Møller, A., & Schwartzbach, M. I. (2023, mei 18). *Static Program Analysis*. Verkregen mei 18, 2024, van <https://cs.au.dk/~amoeller/spa/spa.pdf>
- Oetiker, T., Serwin, M., Partl, H., Hyna, I., & Schlegl, E. *The Not So Short Introduction to LaTeX: The Not So Short Introduction to LaTeX*. 2023, 19, 20. 298 p. Verkregen februari 13, 2024, van <https://tobi.oetiker.ch/lshort/lshort.pdf>
- open source initiative. (2006, juli 7). *The Open Source Definition*. Open source Initiative. Verkregen april 28, 2024, van <https://opensource.org/osd>

- Patashnik, O. (1988, februari 8). *BIBTEXing*. Verkregen mei 4, 2024, van <https://ctan.mines-albi.fr/biblio/bibtex/base/btxdoc.pdf>
- Simpson, J. (2023, december 22). *How JavaScript Works: Master the Basics of JavaScript and Modern Web App Development*. Apress Berkeley. <https://doi.org/https://doi.org/10.1007/978-1-4842-9738-4>
- Turner-Trauring, I. (2023, mei 1). *Goodbye to Flake8 and PyLint: faster linting with Ruff*. Verkregen maart 23, 2024, van <https://pythonspeed.com/articles/pylint-flake8-ruff/>
- van Merode, H. (2023). *Continuous Integration (CI) and Continuous Delivery (CD): A Practical Guide to Designing and Developing Pipelines*. Apress. <https://doi.org/10.1007/978-1-4842-9228-0>