

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

A Study on Developers' Challenges and Struggles with Verification-Aware Languages

Francisco Pinto de Oliveira



Master in Informatics and Computing Engineering

Supervisor: Prof. Alexandra Mendes

Second Supervisor: Prof. João F. Ferreira

September 30, 2024

A Study on Developers' Challenges and Struggles with Verification-Aware Languages

Francisco Pinto de Oliveira

Master in Informatics and Computing Engineering

Approved in oral examination by the committee:

President: Prof. João Bispo

External Examiner: Prof. José Campos

Supervisor: Prof. Alexandra Mendes

September 30, 2024

Resumo

Garantir a fiabilidade do software é crucial numa era de crescente digitalização, em que o software está espalhado por várias áreas do nosso quotidiano. Ao longo dos anos, temos assistido a esforços para melhorar a fiabilidade do software, incluindo a conceção e implementação de conjuntos de testes no seu ciclo de vida de desenvolvimento.

No entanto, os testes não são suficientes para sistemas críticos e altamente complexos, uma vez que, por exemplo, muitas vezes não conseguem detetar erros raramente desencadeados. Para resolver estes problemas, outras técnicas podem ser utilizadas.

O *Design-by-contract* é uma dessas técnicas. Nesta abordagem, os engenheiros de software definem especificações de interface formais, precisas e verificáveis para os componentes do software a ser desenvolvido. Estas especificações expandem as definições de tipos, fornecendo uma lista de pré-condições, pós-condições e invariantes que devem ser respeitadas em todo o programa. As linguagens que suportam nativamente esta abordagem e que vêm equipadas com um verificador, permitindo assim a verificação formal automatizada, são designadas *Verification-Aware languages*. O processo de verificação pode frequentemente ser dividido em duas categorias: verificação em tempo de execução e verificação em tempo de compilação. No primeiro tipo, o programa é normalmente interrompido se o contrato for violado, como acontece em Eiffel. No segundo tipo, os programas, verificados em tempo de compilação, não correm o risco de ter um comportamento inesperado devido a violações do contrato. Alguns exemplos deste tipo de linguagens são a Dafny e Frama-C (Plugin WP). Embora esta abordagem ofereça uma maior garantia de correção, a sua adoção ainda não está generalizada.

Neste trabalho, utilizamos *topic modeling* para extrair dificuldades encontradas por utilizadores de linguagens que suportam a segunda abordagem (verificação em tempo de compilação) de mensagens de fóruns online. Depois, aplicamos um inquérito para validar os desafios encontrados e as melhorias que podem ser aplicadas nas linguagens para melhorar a sua adoção e a experiência do utilizador. Além disso, apresentamos informações sobre a forma mais comum de conhecer e aprender *Verification-Aware languages*, demonstrando que a maioria dos programadores as conhece como uma opção de carreira ou de formação.

Por fim, com a informação recolhida, propomos um conjunto de melhorias que podem ser utilizadas por criadores de *Verification-Aware languages* com o potencial de contribuir para a adoção mais alargada da verificação de contratos em tempo de compilação.

Palavras-chave: Design-by-contract, Fiabilidade, Verificação Dedutiva, Verification-Aware Language

Classificação ACM: Software and its engineering → Software notations and tools

Abstract

Ensuring software reliability is crucial in an era of increasing digitization, where software is spread across multiple areas of our daily lives. Over the years, we have seen efforts to improve software reliability, including designing and implementing test suites in its development life-cycle.

However, testing is not enough for critical and highly intricate systems as, for example, it often fails to detect rarely triggered errors. To solve these issues, other techniques are used.

Design by Contract is one such technique. In this approach, software designers define formal, precise, and verifiable interface specifications for software components. These specifications extend the type definitions by providing a list of preconditions, post-conditions, and invariants that must be respected across the program. Languages that natively support this approach and that come equipped with a program verifier, thus enabling automated formal verification, are called *Verification-Aware languages*. The verification process can often be divided into two categories: run-time verification and compile-time verification. The program is usually interrupted if the contract is violated in languages that implement the former, like Eiffel. Alternatively, programs verified at compile-time have no risk of unexpected behavior due to contract violations. Some examples of such languages are Dafny and Frama-C (WP Plugin). Although this approach offers a stronger guarantee of correctness, its adoption is still not widespread.

In this work, we use topic modeling to extract challenges from online forum posts. Then, we apply a survey to validate the found challenges and improvements that can be made to Verification-Aware languages to improve their adoption and user experience. Additionally, we present insights into the most common way developers know about and learn Verification-Aware languages, demonstrating that most developers know about Verification-Aware languages as a career or education choice.

In the end, with the information gathered, we propose a set of improvements that aim to support language and tool-support designers for verification-aware languages that have the potential to contribute to the broader adoption of compile-time contract verification.

Keywords: Design-by-contract, Reliability, Deductive Verification, Verification-Aware Language

ACM Classification: Software and its engineering → Software notations and tools

Acknowledgements

Firstly, I want to express my gratitude to my supervisors, Professor Alexandra Mendes and Professor João F. Ferreira for their guidance throughout the elaboration of this work.

Additionally, I also want to express my gratitude to Carolina Carreira, IST, for guiding me through the elaboration of the survey and its analysis.

Lastly, I want to thank my family, especially my parents and sister, for all the support they have provided me throughout my life.

Francisco Pinto de Oliveira

Contents

Resumo	i
Abstract	ii
Acknowledgements	iii
1 Introduction	1
1.1 Context	1
1.2 Motivation	2
1.3 Problem Statement	3
1.3.1 Research Questions	3
1.3.2 Methodology Overview	3
1.3.3 Contributions	4
1.4 Document Structure	4
2 Topic Modeling	5
2.1 Introduction	5
2.2 Latent Dirichlet Allocation	5
2.3 Topic modeling to analyze forum posts	6
2.3.1 Model Evaluation	11
2.4 Conclusion	12
3 Verification-Aware Languages	14
3.1 Introduction	14
3.1.1 Databases and Question	14
3.2 Exclusion Criteria	15
3.3 Context of Application and Challenges	16
3.4 Conclusion	21
4 Methodology	23
4.1 Dataset	23
4.1.1 Dataset Creation	23
4.2 Pre-processing	26
4.2.1 Pre-processing steps	26
4.2.2 Pre-processing pipelines	28
4.3 Model Training	29
4.3.1 Training Methodology	29
4.3.2 Hyperparameter Tuning	29
4.3.3 Model Storage	30

4.4	Topic Labeling	31
4.5	Conclusion	32
5	Results	35
5.1	First Model Training Performance Analysis	35
5.2	Second Model Training Performance Analysis	45
5.3	Identified tags	55
5.4	Topic Labeling	59
5.5	Challenge Identification	60
5.5.1	Memory Model Understanding	60
5.5.2	Language Understanding	61
5.5.3	Environment Setup	62
5.5.4	Proof Elaboration	62
5.6	Conclusion	63
6	Validation	64
6.1	Validation Methodology	64
6.2	Survey Design	64
6.2.1	Consent	65
6.2.2	Developer Context	65
6.2.3	Proof Elaboration	67
6.2.4	Environment Setup	68
6.2.5	Feature Knowledge	69
6.2.6	Dafny Knowledge	69
6.2.7	Frama-C Knowledge	73
6.2.8	Additional Challenges	74
6.2.9	Overcoming Challenges	75
6.2.10	Demographics	76
6.3	Survey Distribution	78
6.4	Result Analysis	78
6.4.1	Demographic Analysis	80
6.4.2	Developer Background	80
6.4.3	Proof Elaboration Challenges	81
6.4.4	Environment Setup Challenges	89
6.4.5	Feature Knowledge Challenges	93
6.4.6	Dafny Knowledge	96
6.4.7	Frama-C WP Plugin Challenges	104
6.4.8	Additional Challenges	107
6.4.9	Overcoming Challenges	109
6.5	Conclusion	115
7	Conclusion	117
7.1	Summary	117
7.2	Answers to Research Questions	117
7.3	Limitations and Threats to Validity	118
7.3.1	Topic Modeling	118
7.3.2	Survey	119
7.4	Future Work	119

References	120
A Details about Stack Overflow Question Analysis	125
B First Model Training Tags by Topics	127
C Second Model Training Tags by Topics	133
D Mentioned Questions	137

List of Figures

4.1	Database model diagram	24
4.2	Sequence of tasks for the <i>baseline</i> pipeline	28
4.3	Sequence of tasks for the <i>baseline_bigram</i> pipeline	28
4.4	Sequence of tasks for the <i>rem_words_baseline</i> pipeline	29
4.5	Sequence of tasks for the <i>rem_words_bigram</i> pipeline	29
4.6	Model storage diagram	30
4.7	Prompt showing the models for the configuration <i>e1673342</i>	32
4.9	Example of post visualization	32
4.8	Comparison between mixed and topic views	33
5.1	Average coherence score by number of topics for the <i>baseline</i> pipeline (Highest value)	36
5.2	Average perplexity score by number of topics for the <i>baseline</i> pipeline (Highest value)	36
5.3	Average coherence score by number of topics for the <i>baseline</i> pipeline (Lowest deviation)	37
5.4	Average perplexity score by number of topics for the <i>baseline</i> pipeline (Lowest deviation)	37
5.5	Average coherence score by number of topics for the <i>rem_words_baseline</i> pipeline (Highest value)	38
5.6	Average perplexity score by number of topics for the <i>rem_words_baseline</i> pipeline (Highest value)	38
5.7	Average coherence score by number of topics for the <i>rem_words_baseline</i> pipeline (Lowest deviation)	39
5.8	Average perplexity score by number of topics for the <i>rem_words_baseline</i> pipeline (Lowest deviation)	39
5.9	Average coherence score by number of topics for the <i>baseline_bigram</i> pipeline (Highest value)	40
5.10	Average perplexity score by number of topics for the <i>baseline_bigram</i> pipeline (Highest value)	40
5.11	Average coherence score by number of topics for the <i>baseline_bigram</i> pipeline (Lowest deviation)	41
5.12	Average perplexity score by number of topics for the <i>baseline_bigram</i> pipeline (Lowest deviation)	41
5.13	Average coherence score by number of topics for the <i>rem_words_bigram</i> pipeline (Highest value)	42
5.14	Average perplexity score by number of topics for the <i>rem_words_bigram</i> pipeline (Highest value)	42

5.15	Average coherence score by number of topics for the <i>rem_words_bigram</i> pipeline (Lowest deviation)	43
5.16	Average perplexity score by number of topics for the <i>rem_words_bigram</i> pipeline (Lowest deviation)	43
5.17	Distribution of document frequencies for configuration cb938bf0	44
5.18	Frequency of each tag	45
5.19	Average coherence score by number of topics for the <i>baseline</i> pipeline (Highest value)	46
5.20	Average perplexity score by number of topics for the <i>baseline</i> pipeline (Highest value)	46
5.21	Average coherence score by number of topics for the <i>baseline</i> pipeline (Lowest deviation)	47
5.22	Average perplexity score by number of topics for the <i>baseline</i> pipeline (Lowest deviation)	47
5.23	Average coherence score by number of topics for the <i>rem_words_baseline</i> pipeline (Highest value)	48
5.24	Average perplexity score by number of topics for the <i>rem_words_baseline</i> pipeline (Highest value)	48
5.25	Average coherence score by number of topics for the <i>rem_words_baseline</i> pipeline (Lowest deviation)	49
5.26	Average perplexity score by number of topics for the <i>rem_words_baseline</i> pipeline (Lowest deviation)	49
5.27	Average coherence score by number of topics for the <i>baseline_bigram</i> pipeline (Highest value)	50
5.28	Average perplexity score by number of topics for the <i>baseline_bigram</i> pipeline (Highest value)	50
5.29	Average coherence score by number of topics for the <i>baseline_bigram</i> pipeline (Lowest deviation)	51
5.30	Average perplexity score by number of topics for the <i>baseline_bigram</i> pipeline (Lowest deviation)	51
5.31	Average coherence score by number of topics for the <i>rem_words_bigram</i> pipeline (Highest value)	52
5.32	Average perplexity score by number of topics for the <i>rem_words_bigram</i> pipeline (Highest value)	52
5.33	Average coherence score by number of topics for the <i>rem_words_bigram</i> pipeline (Lowest deviation)	53
5.34	Average perplexity score by number of topics for the <i>rem_words_bigram</i> pipeline (Lowest deviation)	53
5.35	Distribution of document frequencies	54
5.36	Frequency of each tag	55
6.1	Distribution of responses to question DK01	97
6.2	Distribution of responses to question WPK01	105

List of Tables

2.1	Comparisson between the data sources used by each article	13
2.2	Comparisson between the processing steps of each article	13
4.1	Description of columns of the <i>posts</i> table	24
4.2	Description of columns of the <i>tags</i> table	24
4.3	Properties returned by the StackOverflow API	25
4.4	Number of questions for each SE tag	26
4.5	Description of columns of the <i>model_runs</i> table	30
4.6	Description of columns of the <i>model_configs</i> table	31
4.7	Description of columns of the <i>models</i> table	31
6.1	Sections of the survey and respective abbreviated names	65
6.2	Details about question C01	65
6.3	Details about question C02	65
6.4	Details about question DC01	66
6.5	Details about question DC02	66
6.6	Details about question DC03	67
6.7	Details about question DC04	67
6.8	Details about question DC05	67
6.9	Details about question PE	68
6.10	Details about question ES	68
6.11	Details about question FK	69
6.12	Details about question DK01	69
6.13	Details about question DK02	70
6.14	Details about question DK03	70
6.15	Details about question DK04	70
6.16	Details about question DK05	71
6.17	Details about question DK06	71
6.18	Details about question DK07	71
6.19	Details about question DK08	72
6.20	Details about question DK09	72
6.21	Details about question DK10	72
6.22	Details about question DK11	72
6.23	Details about question WPK01	73
6.24	Details about question WPK02	73
6.25	Details about question WPK03	74
6.26	Details about question WPK04	74
6.27	Details about question AC	74

6.28	Details about question OC01	75
6.29	Details about question OC02	76
6.30	Details about question DM01	76
6.31	Details about question DM02	76
6.32	Details about question DM03	77
6.33	Details about question DM04	77
6.34	Details about question DM05	77
6.35	Details about question DM06	78
6.36	Details about question DM07	78
6.37	Demographics of the survey participants	80
6.38	Background and context of survey participants	81
6.39	Frequency of each response to challenge PE ₁	82
6.40	Frequency of each response to challenge PE ₁ from beginner developers	82
6.41	Frequency of each response to challenge PE ₁ from experienced developers	82
6.42	Frequency of each response to challenge PE ₂	83
6.43	Frequency of each response to challenge PE ₂ from beginner developers	83
6.44	Frequency of each response to challenge PE ₂ from experienced developers	84
6.45	Frequency of each response to challenge PE ₃	84
6.46	Frequency of each response to challenge PE ₃ from beginner developers	85
6.47	Frequency of each response to challenge PE ₃ from experienced developers	85
6.48	Frequency of each response to challenge PE ₄	86
6.49	Frequency of each response to challenge PE ₅	86
6.50	Frequency of each response to challenge PE ₄ from beginner developers	86
6.51	Frequency of each response to challenge PE ₅ from beginner developers	86
6.52	Frequency of each response to challenge PE ₄ from experienced developers	87
6.53	Frequency of each response to challenge PE ₅ from experienced developers	87
6.54	Frequency of each response to challenge PE ₆	88
6.55	Frequency of each response to challenge PE ₆ from beginner developers	88
6.56	Frequency of each response to challenge PE ₆ from experienced developers	88
6.57	Frequency of each response to challenge PE ₇	89
6.58	Frequency of each response to challenge PE ₇ from beginner developers	89
6.59	Frequency of each response to challenge PE ₇ from experienced developers	89
6.60	Frequency of each response to challenge ES ₁	90
6.61	Frequency of each response to challenge ES ₁ from beginner developers	90
6.62	Frequency of each response to challenge ES ₁ from experienced developers	91
6.63	Frequency of each response to challenge ES ₂	91
6.64	Frequency of each response to challenge ES ₂ from beginner developers	92
6.65	Frequency of each response to challenge ES ₂ from experienced developers	92
6.66	Frequency of each response to challenge ES ₃	92
6.67	Frequency of each response to challenge ES ₃ from beginner developers	93
6.68	Frequency of each response to challenge ES ₃ from experienced developers	93
6.69	Frequency of each response to challenge FK ₁	94
6.70	Frequency of each response to challenge FK ₁ from beginner developers	94
6.71	Frequency of each response to challenge FK ₁ from experienced developers	94
6.72	Frequency of each response to challenge FK ₂	95
6.73	Frequency of each response to challenge FK ₂ from beginner developers	95
6.74	Frequency of each response to challenge FK ₂ from experienced developers	96
6.75	Frequency of each response to challenge DK _{3OP1}	98

6.76	Frequency of each response to challenge DK _{3OP1} from beginner developers . . .	98
6.77	Frequency of each response to challenge DK _{3OP1} from experienced developers . .	98
6.78	Frequency of each response to challenge DK _{3OP2}	99
6.79	Frequency of each response to challenge DK _{3OP2} from beginner developers . . .	99
6.80	Frequency of each response to challenge DK _{3OP2} from experienced developers . .	99
6.81	Frequency of each response to challenge DK _{3OP3}	100
6.82	Frequency of each response to challenge DK _{3OP3} from beginner developers . . .	100
6.83	Frequency of each response to challenge DK _{3OP3} from experienced developers . .	100
6.84	Frequency of each response to challenge DK ₅	101
6.85	Frequency of each response to challenge DK ₅ from beginner developers	101
6.86	Frequency of each response to challenge DK ₅ from experienced developers . . .	102
6.87	Frequency of each response to challenge DK ₇	102
6.88	Frequency of each response to challenge DK ₉	103
6.89	Frequency of each response to challenge DK ₉ from beginner developers	103
6.90	Frequency of each response to challenge DK ₉ from experienced developers . . .	104
6.91	Frequency of each response to challenge DK ₁₁	104
6.92	Frequency of each response to challenge WPK _{3OP1}	106
6.93	Frequency of each response to challenge WPK _{3OP2}	106
6.94	Frequency of each response to challenge WPK _{3OP3}	106
6.95	Frequency of each response to challenge WPK _{4OP1}	107
6.96	Frequency of each response to challenge WPK _{4OP2}	107
6.97	Frequency of each response to challenge OC ₁	111
6.98	Frequency of each response to challenge OC ₁ from beginner developers	111
6.99	Frequency of each response to challenge OC ₁ from experienced developers . . .	111
6.100	Frequency of each response to challenge OC ₂	112
6.101	Frequency of each response to challenge OC ₂ from beginner developers	112
6.102	Frequency of each response to challenge OC ₂ from experienced developers . . .	112
6.103	Frequency of each response to challenge OC ₃ from experienced developers . . .	113
6.104	Frequency of each response to challenge OC ₃ from beginner developers	113
6.105	Frequency of each response to challenge OC ₃ from experienced developers . . .	113
6.106	Frequency of each response to challenge OC ₄	114
6.107	Frequency of each response to challenge OC ₄ from beginner developers	114
6.108	Frequency of each response to challenge OC ₄ from experienced developers . . .	114
6.109	Frequency of each response to challenge OC ₅	115
6.110	Frequency of each response to challenge OC ₅ from beginner developers	115
6.111	Frequency of each response to challenge OC ₅ from experienced developers . . .	115
A.1	Mapping between Verification-Aware language names and StackExchange tag names	125
A.2	List of removed domain-specific words	126
B.1	Frequency of each tag regarding topic 0	127
B.2	Frequency of each tag regarding topic 1	127
B.3	Frequency of each tag regarding topic 2	128
B.4	Frequency of each tag regarding topic 3	128
B.5	Frequency of each tag regarding topic 4	128
B.6	Frequency of each tag regarding topic 5	129
B.7	Frequency of each tag regarding topic 6	129
B.8	Frequency of each tag regarding topic 7	129
B.9	Frequency of each tag regarding topic 8	130

B.10	Frequency of each tag regarding topic 9	130
B.11	Frequency of each tag regarding topic 10	130
B.12	Frequency of each tag regarding topic 11	131
B.13	Frequency of each tag regarding topic 12	131
B.14	Frequency of each tag regarding topic 13	131
B.15	Frequency of each tag regarding topic 14	132
B.16	Frequency of each tag regarding topic 15	132
C.1	Frequency of each tag regarding topic 0	133
C.2	Frequency of each tag regarding topic 1	134
C.3	Frequency of each tag regarding topic 2	134
C.4	Frequency of each tag regarding topic 3	134
C.5	Frequency of each tag regarding topic 4	135
C.6	Frequency of each tag regarding topic 5	135
C.7	Frequency of each tag regarding topic 6	136
C.8	Frequency of each tag regarding topic 7	136

Abbreviations

AC	Additional Challenges
C	Consent
DC	Developer Context
DK	Dafny Knowledge
DM	Demographics
ES	Environment Setup
FK	Feature Knowledge
LDA	Latent Dirichlet Allocation
OC	Overcoming Challenges
PE	Proof Elaboration
SO	StackOverflow
SE	StackExchange
VA	Verification-Aware
WPK	Frama-C Knowledge

Chapter 1

Introduction

1.1 Context

In recent years, software has become crucial in our daily lives, from smartphones, which we rely on for communication and leisure, to business applications that drive our economy. From home appliances to critical systems like air traffic control, drive-by-wire, and IoT systems, software is ubiquitous in today's world.

Ensuring reliability is crucial, considering both the spread and depth of software use. But what is reliable software?

According to the standard IEEE 610.12-1990, reliability is "The ability of a system or component to perform its required functions under stated conditions for a specified period of time."

Efforts addressing software reliability go back to the beginning of computing, with authors acknowledging that some errors could escape detection for a long time (Myers (1976)). In his work, Myers (1976) defines software reliability and discusses the principal causes of software errors. The author then describes a set of good practices when building a software system, such as starting with the definition of requirements. In the last part of his work, Myers (1976) addresses software tests that, at the time, were considered to consume vast percentages of data processing budgets and to be unknown by most people.

In his work Floyd (1967) developed a formal definition for the meaning of programs expressed in flowcharts. It expresses the notion of pre-conditions and post-conditions and the guarantees the program can ensure, given that the pre-conditions of a given routine are met.

Later, Hoare (1969) extends the formal definitions of Floyd (1967) to routines expressed in programming languages creating Hoare logic.

With the advancements in formalization provided by Floyd (1967) and Hoare (1969), Meyer (1986) develops a strategy called Design-By-Contract (DbC) to ensure software reliability. According to the author, reliable software is correct and robust software. Although the author does not define correctness and robustness, we can find both terms' definitions in the standard IEEE 610.12-1990.

The standard defines correctness as evaluating the degree to which a system or component is free from faults in its specification, design, and implementation. In the context of Design by Contract, we are evaluating the degree to which software complies with the specifications expressed in the form of contracts. The same standard defines robustness as the degree to which a system or component can function correctly in invalid inputs or stressful environmental conditions.

Meyer (1986) defined contracts in the context of an object-oriented language called Eiffel, so contracts can be defined at the method level with pre-conditions and post-conditions, at the class level called class invariants and finally loop invariants.

According to Meyer (1986), it is up to the developer to choose if contracts are verified at runtime. If the developer decides to do so, Eiffel will throw an exception whenever a contract is violated.

On top of the previously mentioned improvements to reliability, Design-by-Contract poses several other advantages, such as facilitating defensive programming, which directs developers to protect every software module against unexpected behavior and aiding the documentation of code by stating what a function expects from the callers and what it can guarantee about its output (B. Meyer (1992)). Additionally, the specification of software behavior using contracts enables formal verification.

Leveraging both Hoare Logic and Design-by-Contract, a new class of languages was created called Verification-Aware Languages. These languages provide mechanisms to formally verify contract compliance at compile-time, avoiding exceptions thrown at runtime and ensuring the software is reliable even without running it. Contract verification is enabled through the added expressiveness of these languages, which allows for the definition of pre and post conditions, as well as loop invariants and class invariants, when suitable.

1.2 Motivation

Ensuring software reliability is of utmost importance to guarantee the trust of its users. The importance of reliability is even greater when considering critical applications where failures can be life-threatening and result in extensive damage Parnas et al. (1990). However, life-threatening failures are far from the unique risk of critical systems; software failures can also cause financial damage and risk users' privacy.

With the increasing interest in critical software applications like autonomous driving, where the number of articles indexed by Scopus from 2019 onwards is five times higher than those indexed before 2018, and Internet of Things, where the number is almost four times higher, ensuring reliability is even more challenging and relevant.

Contracts have been used in multiple contexts, namely the manual (Link (2003)) and automatic (Dai et al. (2007)) derivation of unit tests. However, the application of design-by-contract that interests us the most is formal software verification, whose adoption has been slow in common industrial practice. (Nyberg et al. (2018))

Having both the relevance of writing reliable software and the slow adoption of formal verification using contracts for general applications, the present work provides relevant insights into the challenges of writing said contracts.

We have a broad range of feedback about the challenges of writing and verifying contracts, from beginners to experienced users, from the behavior specification to understanding why verification fails.

Gathering insights into improving said tools, we provide a source of knowledge to drive and aid the development of verification tools and how software verification is taught to beginner programmers.

1.3 Problem Statement

Since its definition in the 1980s in Meyer (1986), Design-by-Contract has experienced a slow adoption in general software development (Nyberg et al. (2018)), in contrast with unit testing that is seen as the de facto standard to ensure reliability in agile methodologies (Ilieva et al. (2004); Aziz Butt (2016); Tripp and Armstrong (2014))

Still, works that approach the challenges of formal verification of Design-by-Contract are often delimited by specific implementation experiences or limited by only observing one language.

Therefore, this work provides insights into the challenges and improvements that can be made to Verification-Aware Language to improve their adoption, not limited by language. The relevance of each improvement can then be used by Verification-Aware Language and verifier developers to improve the developer experience.

1.3.1 Research Questions

In this work, we address the following research questions:

RQ1: What is the context of first contact with Verification-Aware languages?

RQ2: What challenges do Verification-Aware language practitioners face?

RQ3: What improvements can be made to the existing tools to overcome those challenges?

1.3.2 Methodology Overview

We used a two-step approach to answer the previously formulated questions.

In the first step, we used topic modeling to extract challenges from the body of StackOverflow questions, answering RQ2.

In the second step, we elaborated a survey to answer RQ1 and RQ3 and validate the answers to RQ2. The validation of the challenges of RQ2 was done by evaluating statements that translated each challenge found in the first step, using Likert scales. A similar approach was taken to evaluate improvements, derived from the challenges, proposed as an answer to RQ3.

1.3.3 Contributions

The main contributions of this work increase and condense the knowledge about the challenges of adopting VA languages that hinder their widespread use. This work builds on previous methodologies used to analyze the challenges of different areas and applies them to an area that is disregarded by developers and stakeholders.

In summary, the contributions of this dissertation are:

- Collection of challenges that affect VA language developers validated by data gathered from users of those languages.
- Collection of improvements that can be applied to VA languages validated by beginner and experienced practitioners
- An overview of when developers first contact VA languages

1.4 Document Structure

This document follows the structure presented below:

- Chapter 2 introduces the reader to topic modeling terms relevant to future chapters. It also summarizes the methodology previous literature used when applying topic modeling to discover topics in forum questions.
- Chapter 3 summarizes what challenges were found in previous literature when applying VA languages.
- Chapter 4 explains the methodology that will be followed to store, process, and analyze the forum questions that we collected.
- Chapter 5 presents the results of topic modeling and answers the research question RQ1.
- Chapter 6 presents the survey methodology, as well as its questions, the reasoning behind them, and the results of the survey. In the end, questions RQ2 and RQ3 are answered.
- Chapter 7 presents the final remarks of the work, summarizing our findings and proposing the direction of future research.

Chapter 2

Topic Modeling

2.1 Introduction

Online forums are a valuable source of unfiltered data that can be used to understand the challenges faced by verification-aware language developers. It is, therefore, of utmost importance to study posts made on these forums. However, the question of how to deal with the large amounts of data that can be gathered from these sources remains.

One approach is to analyze each question, extracting the desired information manually. However, depending on the size of the dataset, it might be nearly impossible.

In this chapter, we will explore an approach that uses topic modeling, an unsupervised machine-learning technique that identifies groups of similar words in a body of text.

2.2 Latent Dirichlet Allocation

Latent Dirichlet Allocation (LDA) is an unsupervised machine learning algorithm that models automatically extracted topics in corpora. It was first described in the context by [Blei et al. \(2003\)](#).

LDA is based on the idea that each document is a mixture of topics, and each topic is a mixture of words. LDA aims to discover how much of each topic is in each document and how much of each word belongs to a particular topic. The output of the algorithm is percentages that represent the previously mentioned proportions. ([Silge and Robinson \(2024\)](#))

Similar to other machine learning algorithms, several hyperparameters might be tuned. Hyperparameters are the configuration parameters of machine learning algorithms set by the researcher. Below is a list of the most common hyperparameters for LDA:

- **Number of topics, K :** is the number of topics we believe will be needed to describe the corpora.
- **α :** Guess on how the documents mix the topics. High values of α mean that the documents cover a wide range of topics; lower values indicate more specific documents with fewer topics.

- β : Guess on how topics mix words. High values of β mean that every word in the vocabulary has a greater chance of being included in a topic; lower values indicate that the topic has only a few specific words.
- **Iterations, I :** The algorithm is trained through iterative guesses on document-topic and topic-word proportions. This hyperparameter determines how many iterations will be performed to train the model.

In the next sections, we will use the defined terms to refer to the hyperparameters used by each reviewed article.

2.3 Topic modeling to analyze forum posts

Over the years, several research papers have been published on categorizing forum posts using unsupervised machine learning.

Trying to understand what are the main discussion topics of Stack Overflow, Barua et al. (2014) analyzed posts from a data dump of Stack Overflow containing data from July 2008 to September 2010. The authors do not specify the font of the data dump but cite a blog article from Stack Overflow referring to the licensing of their official data dumps (Atwood (2009)). These data dumps can be found on the Internet Archive from January 2014 onward (Fullerton (2014)). The authors took a three-step approach to analyze the data.

Firstly, the authors pre-processed the posts by discarding code snippets, HTML tags, and common English stop words. The last step was the application of the Porter stemming algorithm.

Secondly, they modeled the topics using LDA implemented by a library called MALLET. As is typical with machine learning algorithms, several parameters must be tweaked to get the best results. The authors settled on $I = 500$ and $K = 40$. To improve the quality of text analysis, as claimed by Tan et al. (2002), the authors used a mix of uni-grams and bi-grams to create the topics.

Lastly, they analyzed the resulting document-topic and topic-word distribution. The first step was to define a threshold to indicate whether a document belongs to a topic. This is necessary because, due to the probabilistic nature of LDA, sometimes documents are assigned a small but non-zero chance of belonging to a topic. The authors settled with likelihoods of 0.10 or higher citing Blei et al. (2003). The authors proceed to define metrics to measure several properties of the topics. These metrics were considered specific to their research, so they will not be analyzed.

Following prior research from Barua et al. (2014), Bagherzadeh and Khatchadourian (2019) also used the official Stack Overflow data dump to study the topics of questions about big data.

After gathering the data, the authors identified tags that contained questions that matched their research topic, i.e., big data. Starting with an initial set of tags, the authors extracted questions from the dataset containing tags in the initial set. Then, the authors calculate the significance and relevance of newly found tags. These heuristics were based on previous work by Yang et al. (2016). After testing multiple thresholds for significance and relevance values, the authors settled with 0.2

for significance and 0.005 for relevance, values they claim are in line with previous research of Yang et al. (2016) and Ahmed and Bagherzadeh (2018).

The authors extracted and pre-processed the questions with the identified relevant tags, removing code snippets, HTML tags, stop words from MALLET, numbers, punctuation marks, and nonalphabetical characters. After all these steps they applied Porter stemming to reduce words to their stemmed form.

Multiple models were trained with the previously processed posts using MALLET's implementation of LDA with $K \in \{5, 10, 15, 20, 25, 30, 35, 40, 45, 50\}$ and $I \in \{500, 1000, 1500, 2000\}$. After multiple experiments, the authors used $K = 30$ and $I = 1000$, claiming that those values resulted in sufficiently granular topics for their research. For α and β the following values were used: $\alpha = 50/K$ and $\beta = 0.01$. The authors claimed these values to be standard from previous literature.

Next, the authors labeled the topic using open card sort. To come up with a name, the authors looked at the top 20 most relevant words and 15 random posts for that topic for each topic. Then, they discussed the names each author gave and reiterated until there was a consensus on the names of the topics. A third non-author collaborator reviewed the topic labels and offered suggestions on how to improve the naming. Lastly, the collaborator approved the names after the suggestions had been applied.

The additional steps were considered specific to their study and will not be analyzed.

Han et al. (2020) had a similar approach to previous articles when studying what issues deep learning framework practitioners discuss. The authors gathered data from StackOverflow and GitHub. Data from StackOverflow was obtained from the official publicly available data dumps, and the official API was used for GitHub.

The first step, data pre-processing, is similar to the corresponding step in Barua et al. (2014) and Yang et al. (2016) research, although with two changes from the previously mentioned approaches. First, instead of standardizing words with Porter stemming, they used lemmatization. The second is that stop words were only removed after word standardization.

In the second step, they used an implementation of LDA from the gensim library¹ to perform topic modeling. The authors chose 5 as the number of passes, but no reasoning was provided for choosing this value. The value of K was chosen based on the value that resulted in models with the highest value of topic coherence (Röder et al. (2015)). The range of K was set from 5 to 50 with a step of 5. Due to the probabilistic nature of LDA, three models were trained for each configuration, comparing their performance according to topic coherence. The values of α and β were also tweaked;. However, neither the reasoning for the values chosen in each configuration nor the interval of values used is disclosed; the authors provided a table with the ideal values for each value of K . The best model had the parameters: $K = 20$, $\alpha = 0.05$ and $\beta = 0.05$.

The last step is the aggregation of generated LDA topics into topic categories. The authors proposed a generic deep-learning framework workflow and wanted to associate the questions with

¹<https://radimrehurek.com/gensim/>

each step and then derive the aggregated topic categories. To derive the topic categories, the authors used card sorting derived from [Spencer \(2009\)](#) and [Wan et al. \(2017\)](#).

In line with previous work, [Abdellatif et al. \(2020\)](#) used the official StackOverflow data dump as the dataset to answer the question: "What topics are chatbot developers asking about?". Then, similarly to [Bagherzadeh and Khatchadourian \(2019\)](#), they analyzed two heuristics to discover relevant tags in the dataset related to their research questions, the tag *tag significance threshold* (*TST*) and the *tag relevance threshold* (*TRT*).

Regarding the threshold for each metric, [Bagherzadeh and Khatchadourian \(2019\)](#) mentions that 36 experiments with different values of significance and relevance were performed. Still, the criteria for choosing the threshold values for the heuristics are not described. [Abdellatif et al. \(2020\)](#) independently examined the tags with different *TST* and *TRT* values. For each tag, the first three authors inspected a randomly selected sample of posts to identify when the tags become less relevant and less specific to chatbots, identifying the appropriate heuristic thresholds. Finally, the first three authors independently evaluated the optimal values for the heuristics that yielded the best results and discussed between them to reach a consensus. They finally settled with a *TRT* value higher than 11% and a *TST* value higher than 0.14%, which the authors claim aligns with previous research.

After extracting all the posts with the tags identified in the previous step, [Abdellatif et al. \(2020\)](#) moved on to the pre-processing phase. Unlike [Barua et al. \(2014\)](#), [Bagherzadeh and Khatchadourian \(2019\)](#) and [Han et al. \(2020\)](#), [Abdellatif et al. \(2020\)](#) only used post titles, claiming that previous works show that it is representative of the body of the post and that the inclusion of a post's body can introduce noise. To prepare the titles to be used as training data for the LDA algorithm, the authors removed stopwords using the NLTK² stopwords corpus. Next, they built a bigram³ model using Gensim to reflect pairs of words that naturally occur together. Finally, they lemmatized words to map each word to their origin.

To identify the topics discussed by chatbot developers on StackOverflow, the authors used MALLET's implementation of LDA. The only documented hyperparameter tuning is to the value of K . To discover the best value, the authors ran the model with values of K ranging between 5 to 20 in steps of 1 and calculated the coherence metric ([Röder et al. \(2015\)](#)). The first two authors ran LDA and found that the value of the coherence metric stabilized with K from 10 to 14. To find the ideal value of K , the first two authors examined a randomly selected sample of 30 posts from each topic for K values ranging between 10 and 14. From their examination, $K = 12$ was the ideal value.

The last step, topic labeling, was performed by the first three authors, who looked at the top 20 keywords and a random sample of at least 30 posts from each topic in order to label it. The authors then discussed what the labels should be for the 12 topics.

With a different theme but similar approach, [Openja et al. \(2020\)](#) analyzed StackOverflow questions to answer the question: "What topics are discussed around Release Engineering?"

²Python library to work with human language data. <https://www.nltk.org/> [last access 28th of August 2024]

³Sequence of two adjacent elements from a string

Unlike the other works analyzed, [Openja et al. \(2020\)](#) used a dataset called SOTorrent created by [Baltes et al. \(2018\)](#). This dataset is based on the official StackOverflow data dump released on 2019-09-04.

Similarly to [Bagherzadeh and Khatchadourian \(2019\)](#) and [Abdellatif et al. \(2020\)](#), the authors performed topic extraction. They used the same heuristics, i.e., significance and relevance. The authors had six tags in the initial tag set and applied a strategy similar to [Abdellatif et al. \(2020\)](#) to each tag, identifying individual thresholds. The final set of tags is the union of the six resulting sets. The authors then extracted all the questions containing tags in the final set.

Then, like the other analyzed works, [Openja et al. \(2020\)](#) perform a pre-processing step removing HTML tags, code snippets, numbers, punctuation marks, non-alphabetical characters, and stop words using MALLET's list of stop words. Finally, they applied Porter stemmer to reduce the number of words.

To train MALLET's LDA implementation, the authors chose to use bi-grams and uni-grams, stating that previous studies have shown that this approach improved the quality of the resulting model. Experimenting with values of K between 5 to 60 with a step of 5 and values for I ranging from 500 to 3000 with a step of 500. The authors settled with $K = 40$ and $I = 1000$ and concluded that the values were in line with previous research. Due to the probabilistic nature of LDA, the authors trained four models with the final configuration and compared the 40 topics generated to search for significant differences.

Finally, to label the topics, the authors sorted the documents by probability value from highest to lowest and randomly read through the top 15 to 20 documents representing each group, also analyzing the top 30 keywords. Each author named the topics and discussed them with each other until a consensus was reached.

Due to the similarities between [Al Alamin et al. \(2021\)](#)'s work with the other analyzed works, here, we will only analyze a specific innovation in pre-processing, the hyperparameters chosen, the criteria to select the best model configuration, and the strategy to label the resulting topics.

Regarding the pre-processing step, the authors remove the stop-words using existing lists of words but also introduce domain-specific stop-words, such as platform names.

With respect to the hyperparameters chosen, the authors tested values of K from 5 to 30 with a step of 1, with $I = 1000$, $\alpha = 50/K$, and $\beta = 0.01$, similarly to [Bagherzadeh and Khatchadourian \(2019\)](#). The best model was considered the one with the highest coherence measure ([Röder et al. \(2015\)](#)). There were two models with tied measures with $K = 7$ and $K = 13$; the authors chose $K = 13$, claiming that it captured their underlying topics better.

Finally, to label the resulting topics, similarly to [Abdellatif et al. \(2020\)](#) and [Openja et al. \(2020\)](#), the authors performed open-card sorting using the top 30 words and a random sample of at least 20 questions. The authors reached an agreement after 15 iterations.

[El aoun et al. \(2021\)](#) used two different sources GitHub⁴ and StackOverflow data gathered from StackExchange Data Explorer⁵. The methodology of processing both datasets is similar to

⁴<https://github.com/> [last access 10th of June 2024]

⁵<https://data.stackexchange.com/> [last access 10th of June 2024]

the ones previously analyzed. Considering the hyperparameters, the authors picked values for α and β between 0.01 to 1 with a step of 0.01. The number of topics K was picked from 5 to 30 with a step of 1. Unlike most works analyzed, the number of iterations is not mentioned. Topic coherence (Röder et al. (2015)) is used to measure model quality.

When labeling the topics, the authors used hybrid card sorting, where two authors of the paper labeled different halves of the data in a two-round process. In the first round, each author labels half of the questions and discusses the labels. Then, in the second round, the authors label the other half of the questions with the categories discovered in the first round. Finally, the authors discuss the results of the labeling process and settle on an agreement with a third author.

The last article to be analyzed is from Peruma et al. (2022). Like Openja et al. (2020), Peruma et al. (2022) used the SOTorrent dataset to answer the question: "What are the Topics Around Software Refactoring that are Being Asked by Developers?".

Given the broad nature of Stack Overflow posts, first, the authors filtered them, extracting only posts tagged with refactor or that contained variations of refactor in the title. With this strategy, the authors ensured that all the posts were related to their research topic.

The authors then performed both a qualitative and quantitative analysis of the dataset; for the purpose of this chapter, only quantitative analysis using topic modeling is relevant. Starting with post pre-processing, the authors expanded word contractions (e.g., "I'm" -> "I am"); removed URLs, code blocks, stop-words, alphanumeric words, and punctuation, retaining only nouns, verbs, adjectives, and adverbs. Finally, they lemmatized the words. The authors claim to have used lemmas over stemming because the lemma of a word is a valid English word. Apart from using NLTK stop-words, the authors identified frequently occurring words that have irrelevant meanings; the examples given were: "thanks," "question," and "answer," and removed them.

Regarding hyperparameter tuning, the authors extracted topics from 2 to 50 with a step of 1. In each cycle of training, the authors measured the coherence (Röder et al. (2015)), perplexity score (Blei et al. (2003)), and performed topic visualization (Sievert and Shirley (2014)). The authors also used manual examination to discover patterns such as similarities between topics, overlapping topics, and what topics are discovered by every training. Finally, to derive the topic labels, the authors only looked at the terms representing each topic, unlike the other works.

In conclusion, the state-of-the-art methodology to analyze forum posts using topic modeling can be divided into four steps.

The first, *data collection*, usually from StackOverflow, but articles using other sources were also found. Table 2.1 compares the data sources used by the articles that were analyzed. To identify relevant tags the authors Bagherzadeh and Khatchadourian (2019), Abdellatif et al. (2020), Openja et al. (2020), Al Alamin et al. (2021) and El aoun et al. (2021) used heuristics to find new relevant tags from an initial set of previously identified tags.

The second is *data pre-processing*, where most authors implemented a variation of the following steps: removal of all non-alphanumeric characters and markup language tags, including the ones that envelop code snippets and stop words. The final step is usually the standardization of

words through Porter stemming or lemmatization. Table 2.2 compares the ordering in which each author performed the post pre-processing.

The third is called *model training*, where LDA is trained using the dataset pre-processed in the previous step.

The last step called *tag labeling*, is where the authors label the tags, looking at words and topics belonging to each tag.

2.3.1 Model Evaluation

The articles analyzed in the previous section mention three strategies to evaluate the performance of a given model, i.e., topic coherence (Röder et al. (2015)), perplexity score (Blei et al. (2003)), and topic visualization (Sievert and Shirley (2014)).

When Röder et al. (2015) defined topic coherence as the degree to which statements support each other, they did not define a single metric but a framework to calculate coherence metrics using existing formulas. However, none of the analyzed articles acknowledged this and explicitly stated the strategy used.

In this work, we used the libraries of analyzed works to hint at which strategy the authors might have used. Most of the articles used MALLET's implementation of LDA (Barua et al. (2014), Bagherzadeh and Khatchadourian (2019), Abdellatif et al. (2020), Openja et al. (2020), Al Alamin et al. (2021), El aoun et al. (2021)). Function 2.1 defines the coherence definition for MALLET.

$$C(w_i, w_j) = \sum_i \sum_{j < i} \log \frac{P(w_j, w_i) + \beta}{P(w_i)} \quad (2.1)$$

As the authors define in McCallum (2002): "The score is the log of the probability that a document containing at least one instance of the higher-ranked word also contains at least one instance of the lower-ranked word." In this case, β is a smoothing parameter to avoid the value of the logarithm being zero.

This formulation is very similar to the one Röder et al. (2015) called C_{UMass} that is based on a work by Mimno et al. (2011), and is defined by Function 2.2

$$C(w_i, w_j) = \frac{2}{N \times (N - 1)} \sum_{i=2}^N \sum_{j=1}^{i-1} \log \frac{P(w_i, w_j) + \epsilon}{P(w_j)} \quad (2.2)$$

Where $P(w_j)$ is the number of documents with least one token of type w_j and where $P(w_i, w_j)$ is the number of documents containing one or more tokens of type w_i and at least one token of type w_j . (Mimno et al. (2011))

We can also conclude that the higher the value the better the coherence because when the probabilities tend to 1, meaning in almost all documents a given pair of words appear together, the

logarithm tends to 0 approaching 0 by negative values so the sum of lower negative values gives an higher overall coherence value.

We decided to use C_{UMass} to evaluate our models since it is a standard metric that can be quickly calculated.

We also analyze the logarithm of the perplexity, which measures how likely an outside observer is to predict the values drawn from a model. The values used are the output of the function `log_perplexity`⁶ in Gensim. Higher values indicate that the more likely an observer can guess the value drawn from the model.

2.4 Conclusion

In this chapter, we analyze eight articles in detail that use topic modeling, particularly with LDA, to analyze the challenges of various areas of Software Engineering.

From these articles, we discovered four datasets with forum questions, nine pre-processing steps for the body of the retrieved questions, and the ways to combine them.

Finally, we explored two metrics for model evaluation that will be used to compare the multiple models analyzed in chapter 5.

⁶https://radimrehurek.com/gensim/models/ldamodel.html#gensim.models.ldamodel.LdaModel.log_perplexity [last access 31st of August 2024]

Table 2.1: Comparisson between the data sources used by each article

	SO Data Dump Fullerton (2014)	SOTorrent Baltes et al. (2018)	SE Data Explorer	GitHub
Barua et al. (2014)	X			
Bagherzadeh and Khatchadourian (2019)	X			
Han et al. (2020)	X			X
Abdellatif et al. (2020)	X			
Openja et al. (2020)		X		
Al Alamin et al. (2021)	X			
El aoun et al. (2021)			X	
Peruma et al. (2022)		X		

Table 2.2: Comparisson between the processing steps of each article

	Removal of HTML tags	Removal of non-alphabetical characters	Removal of code snippets	Removal of stop words	Build a bigram model	Join title with body	Expansion of word contractions	Porter stemming	Lemmatization
Barua et al. (2014)	2nd		1st	3rd				4th	
Bagherzadeh and Khatchadourian (2019)	2nd	4th	1st	3rd				5th	
Han et al. (2020)	2nd	3rd	1st	5th					4th
Abdellatif et al. (2020)				1st	2nd				3rd
Openja et al. (2020)	2nd	3rd	2nd	3rd		1st		4th	
Al Alamin et al. (2021)	1st	2nd	1st	2nd				3rd	
El aoun et al. (2021)	3rd	4th	2nd	4th		1st		5th	
Peruma et al. (2022)	2nd	5th	3rd	4th			1st		6th

Chapter 3

Verification-Aware Languages

3.1 Introduction

This dissertation aims to identify the practices and challenges usually encountered when using Verification-Aware (VA) languages. These languages natively support constructs to enable automated program verification, i.e., pre-conditions, post-conditions, and invariants.

To understand what are the contexts in which practitioners apply these languages and the challenges that arise from their use, we defined the following questions:

- What verification-aware languages are used in the industry?
- With which purpose are these languages used in an industrial context?
- What are the issues already identified in adopting and using verification-aware languages?

3.1.1 Databases and Question

We decided to query the following databases: [ACM Digital Library](#) and [IEEE Xplore](#) databases because of their specific nature tailored to Engineering. In addition, we also query [Scopus](#) and [Web of Science](#) because of their wide range of publication themes and curated results.

The research query was developed following an iterative methodology. Firstly, we identified the following articles we would like to find with our research query: [White et al. \(2017\)](#); [Todorov et al. \(2018\)](#); [Nyberg et al. \(2018\)](#); [Schoolderman et al. \(2019\)](#); [Hähnle and Huisman \(2019\)](#). Then, we developed an initial research query based on keywords from the collected articles. The resulting query was then tailored to find as many articles as possible containing the previously identified articles. We decided only to accept articles with ten or fewer years because, considering we are trying to understand developer practices, old articles could contain issues already addressed in the languages concerned or issues that are outdated at the time of writing. We included a set of verification-aware programming languages to expand the results of our search query. These languages resulted from a search of industry used *Verifiable Programming Languages* from [Formal Methods Europe](#) consulted on 20/11/2023. The languages that did not comply with our definition of verification-aware languages were removed.

Finally, the resulting query was chosen because it contained all the articles we defined initially as crucial and included the 10-year limit on publication date. It is worth noting that the query only searches in the field *Abstract* because it is the field that all databases support, and for that, we could find relevant keywords.

The resulting query is as follows:

```
("theorem proving" OR "theorem proof" OR "deductive
  verification" OR "deductive methods" OR "formal
  verification" OR "formal specification" OR "contract
  programming" OR "programming by contract" OR "design by
  contract" OR "Bandera" OR "Dafny" OR "Escher C Verifier"
  OR "Eiffel" OR "Frama-C" OR "Java Pathfinder" OR "JML" OR
  "mbeddr" OR "Spec#" OR "Spark Ada" OR "VeriFast" OR "VCC
  " OR "Whiley" OR "Why3") AND industr* AND (challeng* OR
  issue* OR drawback? OR constraint? OR limitation? OR
  obstacle? OR difficult*)
```

3.2 Exclusion Criteria

The following criteria were used to filter the articles that resulted from the previous search query and duplicate removal. They were elaborated to include the maximum number of valid articles, and when in doubt, the full-text screening was applied directly.

3.2.0.1 Title Screening

- The title is not related to formal verification (or derived terms)

Examples:

- Older Adult Tourism to Older Adult Sojourn: A New Discuss Based on the Life-Span Theory of Control
- Solidification Principle in Large Vertical Steel Casting Under the EMS Effect

- The title mentions formal verification tools that are not verification-aware languages

Example:

- Alloy, SysML

3.2.0.2 Abstract Screening

- The term formal methods or related is used in contexts not related to our research

Example:

- Formal Verification of Startup models

- The tools or the formal verification methods mentioned are related to the area of software engineering, but the area of application is not aligned with the research

Example:

- Formal verification of system models

3.2.0.3 Full-Text Screening

- The article does not mention any challenges
- The challenges are not related to the application of verification-aware languages

Example:

- The article talks about the challenges of integrating Dafny with Solidity

The queries were made to all the databases on 26/12/2023, resulting in 1024 references. After 729 duplicates were removed, 600 articles were screened. Nine of the 600 articles were considered relevant; we analyze them in the next section.

3.3 Context of Application and Challenges

In their experiments to apply formal verification to automotive embedded software, [Todorov et al. \(2018\)](#) applied deductive verification to a function that computed the square root by linear integer interpolating numbers from 0.00 to 100.00. When using Frama-C, the authors state that for the full table of numbers, all the default solvers failed to verify the function's post-condition. The authors then *resorted to help from Frama-C's development team*, which provided a special solver that correctly verified the function. When using SPARK, the proof is completed successfully. SPARK also returned a counter-example when the contract checking failed. Therefore, the success of the proof can depend on the solver, and the quality of the feedback given when an error occurs varies from tool to tool. On top of that, [Todorov et al. \(2018\)](#) stated that *many tools coming from academia are developed by PhD students and postdoctoral researchers, resulting in abandonment after the conclusion of such research*. They also pointed out that there are few algorithms in commercial tools, and the existing ones are badly documented.

[Nyberg et al. \(2018\)](#) tried to apply deductive verification using verification-aware languages, namely VCC and Frama-C, to three components of their automotive software. VCC could not verify one of the three components because it *lacks floating-point arithmetic verification*. To verify these components, the researchers first formalized the existing component requirements. Then, they created contracts for each functional requirement¹. The researchers conclude that *deductive verification requires rigorous formalization and a deep understanding of the tools and processes*. They also noted *automation of large parts of the process is required for widespread*

¹The researchers defined functional requirements as requirements that define output values as a function of the input values. This definition differs from the definition commonly used in software engineering.

adoption. Some tools either *lack fundamental features* (for instance, VCC lacks reasoning about floating points), have *scalability problems*, or require *time-consuming manual work*.

White et al. (2017) state some contexts where deductive verification aided by verification-aware languages can be applied. They analyze the verification of the following systems:

- **SHOLIS:** monitoring system for helicopter landing parameters on top of ships.
- **Global Key Centre:** system that provides cryptographic keys for MULTOS, a smartcard operating system
- **Engine Monitoring Unit:** a system that provides in-flight monitoring of jet engines to inform just-in-time maintenance activities.
- **Tokeneer biometric access control:** system that provides biometric-based access control for a secure room.
- **iFACTS:** set of tools to help UK air traffic controllers to handle increasing volumes of traffic safety.

After presenting multiple success cases, the authors present some possibilities for why automatic verification tools are not widely adopted. Firstly, there is the issue of being locked to a tool. The authors state that tools often have a particular *notation that is not interoperable*. Secondly, the authors identified the *price of commercially available formal verification tools*. When choosing deductive verification, the *cost of changing from one tool to another* is also a concern. The lack of developer knowledge and the often changing industry practices that are misaligned with objectively superior tools and strategies in the long term pose a challenge when *convincing stakeholders* to use verification-aware languages. Finally, the *cost of using formal methods from the beginning of a project* and the fact that they often *lack a drag-and-drop user interface* are also mentioned.

Hähnle and Huisman (2019) was introduced despite not being part of the results of our search query because it is part of the list of relevant articles that serve as the basis for the research query. In this article, the authors describe the technical and non-technical challenges of verification.

On the technical side, the authors start by stating that deductive verification specifications are not as declarative and abstract as desired, polluting the code base with annotations and hints to the verification tools. This challenge is aggravated by the lack of verified, widely-used APIs that must be verified per project. The absence of verified APIs prevents verification from being scalable. Additionally, the authors mention the integration with tools like IDE, debuggers, and invariant generators. Finally, the authors mention the limitations of the tools related to verifying floating-point arithmetic, concurrent software, and programs where reflection is necessary.

On the non-technical side, the authors mention the challenges of convincing industrial stakeholders that it is valid and can fit into the existing development environment. The authors point out that the development of such tools is expensive and often abandoned because the academy does not reward practical achievements and long-term success, leading to the abandonment of the development of tools. Finally, the authors mention the societal context where the timely delivery

of features is often more important than quality and that the development of new language features is not made with verification in mind, making coverage unnecessarily difficult.

Dordowsky (2015) used Frama-C with the WP Plugin to verify a set of functions that read data from two redundant temperature sensors and return the mean, as long as the readings do not differ more than a certain amount.

The *first* obstacle encountered by the authors is related to the visibility of internal module variables, i.e., a static variable of file scope. Since ACSL notations are placed in header files and static variables are placed in implementation files, the authors had to create a ghost variable that replicated the value stored in the original variable, uncovering the internal state of the module.

The *second* challenge is related to the first one and is about the specification of behavior across multiple calls. In their example, the authors wanted to tolerate two erroneous readings before reporting a reading error. To accomplish that, the authors stored the number of erroneous readings in a static variable. Again, in this case, the program's internal logic needs to be disclosed in the header file through a ghost variable to allow verification. The authors also state that an alternative language and plugin could be used to specify the behavior; that plugin would then create the ACSL annotations to be verified by the WP Plugin; however, due to time constraints, the authors did not try that approach.

The *third* challenge is related to the pattern of obtaining input values from calls to subroutines. In the example below, the authors created a function `acq_measure_temp` that returns the temperature readings through the arguments `temp1` and `temp2`. The authors created two ghost variables that represented the return values of the subroutines and then defined predicates since simple assertions could not be used due to the value of the temperatures only being determined at runtime.

```
acq_measure_temp(&temp1 , &temp2 );
```

The *fourth* challenge is related to the inability to use *complete* and *disjoint* clauses since, because of the previous challenge, the behavior is expressed with predicates instead of *assumptions* through the *assumes* clause.

The *fifth* challenge is related to specification proliferation, i.e., the repetition of lower-level function specifications by higher-level functions.

The *sixth* challenge is related to the fact that, in the version used for the study, the WP Plugin did not support math functions, making it hard to verify the function that converted voltages to temperature readings.

The *seventh* challenge is related to the lack of compiler-specific language extensions to access hardware registers, which, since they are not standard, are unknown by the WP Plugin.

The *eighth* and last challenge is related to the modeling of hardware access functions that require the hardware to be modeled in ACSL to be verified due to the side effects of hardware interaction. The authors state that future versions of Frama-C support volatile ghost variables used to store values where the reading might result in a side effect. However, the temporary solution was to create small functions that interact with the hardware and manually verify them.

Gurov et al. (2017) describe the verification of automotive embedded C-code written at SCANIA. The authors describe the formalization process of requirements and software verification and the obstacles encountered. The formalization of requirements allowed the authors to discover two contradictions. Still, the authors consider the formalization of requirements positive since it allows the resolution of ambiguities and contradictions.

The *first* obstacle is related to type safety. The authors stated that they had to remove implicit type conversions and aliasing of distinct memory objects so that VCC could verify the properties of memory handling and management.

Another obstacle that was encountered was related to the verification of behaviors that depend on previous executions. The state of previous executions is often saved in static variables outside the scope of verification. Similarly to Dordowsky (2015), the authors suggest creating ghost variables that replicate the value of the static variables or avoid them altogether.

Finally, the authors stated that boolean types. In C, there is no native boolean type, so the value is usually stored inside an integer. However, the verifier assumes that the integer can take any value in the range and not only two, making verification more challenging.

Knüppel et al. (2018) used KeY to verify parts of the Collections API of OpenJDK 6 specified using JML. The authors divided the challenges into two categories: lessons learned from formalizing parts of open JDK and experiences drawn from verifying OpenJDK.

In the first category, the authors identified two challenges. The *first* challenge was that the informal specifications of Java Collection API were often imprecise or even incorrect. The *second* challenge is related to the scope of verification. They claimed that most currently active verification systems only focus on the method under verification and do not consider callers of a given method.

In the second category, the authors identified four challenges.

In the *first* challenge, the authors state that the *ArrayList* constructor behavior in case of invalid arguments, i.e., capacity lower than zero, is impossible to verify because that flow would result in a property not being instantiated when the class invariant states that it is always different from null.

The *second* challenge is related to the verification of *List.indexOf(Object)* a method that uses Java's *Object.equals(Object)* because the later method is not verified, the specification phase can not be completed.

The *third* challenge is related to unspecified Java semantics. Specifically, when verifying the method *ArrayList.toArray()*, the authors could not decide the maximum size for an array since it depends on the virtual machine configuration and is not part of the Java language specification.

The *fourth* and last challenge is related to the parameterization of KeY. According to the authors, KeY possesses a variety of options related to the behavior of the verifier. The issue arises from the need to change the parameters so that verification is successful, which often leads to trial and error until the verification is successful.

Djouidi et al. (2021) used Frama-C with WP and MetAcsI plugins to verify a large-scale application, a JavaCard Virtual Machine, with over 7000 lines of C code.

The *first* challenge the authors encountered was related to the encoding of multiple flags into a single integer variable. The authors stated that automatic proof failed for many properties when manipulating multiple bits. The authors maintained an equivalence between the flag-storing variable and a boolean ghost variable to overcome this challenge.

In the *second* challenge, the authors state that rewriting type casts and introducing ghost variables to store the value of casted values was necessary due to the difficulty of proving properties among heterogeneous casts, i.e., casts between different types.

In the *third* challenge, the authors show that there is still a manual process of consulting the interactive proof editor when the verification fails. Still, the feedback seemed helpful in allowing them to proceed with the verification.

The *fourth* challenge was related to scalability issues with the simplification steps of Qed taking 40 minutes per property. The solution the authors found was to turn off some simplifications.

In the *fifth* challenge, the authors state that ACSL annotations may become pervasive and difficult to track and maintain, requiring careful organization to ensure specification traceability and maintainability.

Overall, the authors consider that their work represents a successful industrial application of deductive verification. However, the effort in creating and updating WP proof scripts was used to unfold predicate definitions, split proof goals, and instantiate quantified goals with specific values. The authors also suggest adding lemmas to prove cast properties about casts from different integer types.

In their work, [Schoolderman et al. \(2019\)](#) documented students' experience in a Master program verifying a non-trivial C function using Why3. The student's challenges can be split into two groups: the ecosystem challenges and the tooling challenges.

Regarding the ecosystem challenges, the students complained about the lack of an online community for Why3 users on StackExchange.

Regarding the tooling challenges, the students mentioned that the error messages provided, namely regarding syntax and type errors, were challenging to understand. The students also mentioned challenges related to understanding why proof is failing and that the counter-examples provided were only helpful for very simple proofs. The students also mentioned that the lack of a debugger prevented exploring the program state in depth. Finally, the students reported challenges related to the discovery of loop invariants.

Similarly to the work of [Schoolderman et al. \(2019\)](#), [Güdemann \(2021\)](#) reports on teaching formal methods in the form of program verification for Master students. The part that is relevant to our work is related to deductive verification, and for that, the authors used Frama-C with the WP plugin. A big challenge for students was understanding why proofs did not work. One reason for this challenge presented by the authors was that the proof obligations were reported using first-order logic and were not trivially mapped to the source code written in C. Finally, the authors mentioned that it is related to creating loop invariants with limited automated tool support.

In their work, [Selvaraj et al. \(2019\)](#) compares multiple verification methods to prove that: "If changing lane, the lane change shall always be to the same side as indicated." One of the tools

evaluated was SPARK Ada. The authors state that the tool needs stronger annotations to verify a functional property and could not complete the proof.

3.4 Conclusion

In conclusion, the purpose of the application of Verification-Aware languages can be split into three categories. The first is the verification of large-scale applications (White et al. (2017); Djoudi et al. (2021)). The second is as a tool to teach formal methods (Schoolderman et al. (2019); GÜdemann (2021)). The last category contains all the remaining seven articles that only considered verifying small components of larger systems.

Regarding the challenges, we identified 21 groups from 11 articles.

The main challenges identified by the authors are:

- **Behavior Dependent on Previous Executions:** Includes challenges related to the verification of properties where the behavior is influenced by previous executions (Dordowsky (2015); Gurov et al. (2017)). This challenge was only observed with C programs.
- **Behaviour Inconsistency:** In this category, the authors verified that the verification success changed depending on the solver (Todorov et al. (2018)).
- **Cost of Applying VA Languages (White et al. (2017))**
- **Feedback Quality:** This category includes the lack of counter-examples by some tools (Todorov et al. (2018)). It also includes the lack of clarity of the error messages (Schoolderman et al. (2019)) and a general difficulty in understanding why proofs fail (GÜdemann (2021)).
- **Handling Boolean Values:** This category's challenges are related to verifying properties about Boolean values. These challenges were only verified in the C language that does not contain a native boolean type, so integers are often used instead (Gurov et al. (2017); Djoudi et al. (2021)).
- **Imprecise Specs:** Includes challenges related to ambiguous, incomplete, or contradictory specs. Still, resolving these challenges often results in stronger, more correct specs (Gurov et al. (2017); Knüppel et al. (2018)).
- **Increase the Scale of Verification:** Includes challenges related to the integration of multiple verified functions (Knüppel et al. (2018)), the scaling of verification time (Djoudi et al. (2021)), the difficulty of tracking and maintaining annotations (Djoudi et al. (2021); Hähnle and Huisman (2019)) and general claims about scalability (Nyberg et al. (2018)).
- **Integration with Hardware:** Includes challenges related to the verification of functions that interact with hardware, often resulting in side effects (Dordowsky (2015)).

- **Integration with Other Tools** (Hähnle and Huisman (2019))
- **Lack of Features:** Includes challenges verifying some behaviors due to the lack of features (Knüppel et al. (2018); Selvaraj et al. (2019))
- **Lack of Verified APIs:** Includes challenges related to the general lack of verified APIs (Hähnle and Huisman (2019)) and specific to math functions (Dordowsky (2015)).
- **Lacking Support for Specific Tasks:** This category includes lack of support for floating-point arithmetic (Nyberg et al. (2018); Hähnle and Huisman (2019)), concurrent software (Hähnle and Huisman (2019)) and reflection (Hähnle and Huisman (2019)) verification.
- **Loop Invariant Discovery** (Schoolderman et al. (2019); Güdemann (2021))
- **Need for Automation:** Includes authors that felt the need for automation so that deductive verification can be widespread Nyberg et al. (2018) or authors that applied it Djoudi et al. (2021).
- **Output as Arguments:** This challenge is related to the verification of C functions that output values through arguments (Dordowsky (2015)).
- **Shortage of Experienced Developers** (White et al. (2017))
- **Support and Documentation:** Includes challenges related to the support of tools being insufficient (Todorov et al. (2018); Schoolderman et al. (2019); Hähnle and Huisman (2019)) and costly (White et al. (2017); Hähnle and Huisman (2019)).
- **Support for Math Functions:** Includes challenges related to the lack of support for floating-point arithmetic (Nyberg et al. (2018); Hähnle and Huisman (2019)).
- **Switching Costs:** Includes authors that considered the lack of interoperability between tools and the cost to switch them an obstacle for the adoption of VA languages (White et al. (2017)).
- **Type Conversions:** The challenges in this category are related to the verification of implicit (Gurov et al. (2017)) or heterogeneous (between different types) (Djoudi et al. (2021)) type conversions.
- **Utility Challenges:** Includes challenges of convincing industrial stakeholders that verification can fit into existing development environments and verifying software where the timely delivery of features is more important than quality. (Hähnle and Huisman (2019))

Chapter 4

Methodology

This chapter presents and describes the methodology to address the questions proposed in Chapter 1, namely RQ2. In section 4.1, we explore existing dataset options identified in Chapter 2 and explain the reasoning for creating our own. Following dataset creation, we explain our pre-processing pipeline in section 4.2. In section 4.3, we explain the methodology to train all our LDA models as well as the measures taken to overcome the algorithm’s non-deterministic nature. Finally, in section 4.4, we explain our strategy to label the topics and identify the challenges.

4.1 Dataset

In Chapter 2, we discovered three StackOverflow and StackExchange post datasets. The first, SO official data dumps (Fullerton (2014)), is a collection of data dumps created periodically and distributed as an XML file with a size of around 100 GiB. This distribution format makes it very hard to process and filter posts.

The second, SOTorrent (Baltes et al. (2018)), is a dataset that was last updated on 31st of December 2020. In the dataset’s website¹, the author states that it will not be updated in the foreseeable future, so its use is not viable due to outdated contents.

The third and last one, SE Data Explorer², seems like a good source for updated SE post data, but the interface seems more human-centered than computer-centered, so its use in a script, for instance, is almost unfeasible.

We decided to use SE official API³. Similarly to SE Data Explorer, the API provides the newest data at the time of querying but in a computer-friendly format.

4.1.1 Dataset Creation

As previously stated, we created our dataset with data from the official SE API in an SQLite database. The database contains two tables: *posts* and *tags*. Table 4.1 contains a description for

¹<https://empirical-software.engineering/sotorrent/> [last access 11th of June 2024]

²<https://data.stackexchange.com/> [last access 10th of June 2024]

³<https://api.stackexchange.com/>

each column of the table *posts*, and the table 4.2 contains a description for each column of the table *tags*. Figure 4.1 depicts the relationship between these two tables.

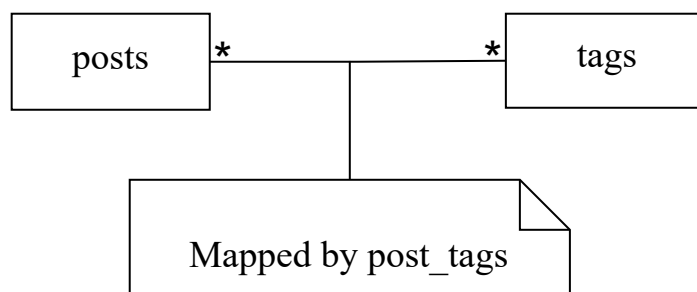
Table 4.1: Description of columns of the *posts* table

Column name	Description
id	Id of each question
question_id	StackOverflow id of each question
title	Question title
body	Question body
body_clean	Question body after the pre-processing step
link	Original link to the question in StackOverflow
up_vote_count	Number of question up votes
creation_date	UNIX timestamp in seconds indicating the question creation date
fetch_date	UNIX timestamp indicating the date in which the API was consulted and the question saved

Table 4.2: Description of columns of the *tags* table

Column name	Description
id	Id of each tag
name	Name of each tag

Figure 4.1: Database model diagram



The StackExchange Official API can be used to retrieve data from multiple websites, such as StackOverflow, SuperUser, and Mathematics. Similarly to previous research, we chose to use data only from StackOverflow (Barua et al. (2014); Bagherzadeh and Khatchadourian (2019); Abdellatif et al. (2020); Al Alamin et al. (2021)) since it is the website where most development discussions occur and has no specific theme.

After identifying the adequate sources, we identified the tags we would use to query the API endpoint. We settled with a variation of the Verification-Aware languages identified in Chapter 3. From our observations of the StackOverflow tag list ⁴, only hyphenated tags are supported, so “Escher C Verifier”, “Java PathFinder”, and “Spark Ada” had to be transformed to accommodate this requirement. Similar to what we observed in Chapter 3, the tool KeY introduced significant

⁴<https://stackoverflow.com/tags>

noise due to the name’s generality. For this reason, the tool was removed from the list of tags. Additionally, we verified that there were no questions with the tags “Escher C Verifier” and “Java PathFinder”, thinking that this was due to the tag’s complexity, we removed the part that referred to the underlying programming language but to no avail. The final list mapping between Verification-Aware language official names and the tag names with all the aforementioned pre-processing can be found in Table A.1.

Additionally, we tweaked the attributes returned by the API upon querying to reduce the amount of data that would be transferred but discarded. After some analysis, we ended up with the following filter:

`!D4p-cX0xzCDrlgjnFwqTQ-Nf_E3-DWR(2 n60tGdKJYOdSU(6UME`

StackExchange API filters are not human readable, but the fields they affect can be consulted at the official API documentation⁵. Summarizing, the filter removes all but the properties described at Table 4.3.

Table 4.3: Properties returned by the StackOverflow API

Property	Description
tags	List of question tags
up_vote_count	Number of up votes
last_activity_date	UNIX timestamp of the last change to the question (including edits)
creation_date	UNIX timestamp of the creation date
question_id	StackOverflow question Id
link	Hyperlink to the question page
title	Question title
body	Question body in HTML

Moreover, we decided only to query the */questions* endpoint because we are looking for challenges commonly found in questions but not in the follow-up interactions, i.e., answers and comments.

Finally, we built a Python script that creates the database according to the schema displayed in Figure 4.1, queries the StackExchange API endpoint to gather the questions tag by tag from StackOverflow, and finally stores the questions and tags in an SQLite database. The final queried endpoint is the following, with {0} being replaced by the page index and {1} being replaced by the tag:

`https://api.stackexchange.com/2.3/questions?page={0}&
pagesize=100&tagged={1}&site=stackoverflow&filter=!D4p-
cX0xzCDrlgjnFwqTQ-Nf_E3-DWR(2 n60tGdKJYOdSU(6UME`

We ended up with 1420 questions, most of which were about Dafny and Frama-C. The precise distribution of questions can be found in Table 4.4. When compared to Table A.1, the missing tag names correspond to tags for which no questions were found.

⁵<https://api.stackexchange.com/docs/questions>

Table 4.4: Number of questions for each SE tag

Tag Name	Number of Questions
dafny	530
frama-c	474
eiffel	289
spark-ada	55
why3	52
jml	44
spec#	16

4.2 Pre-processing

Since we gathered data from StackExchange API, the question bodies have the same properties as the ones obtained from Fullerton (2014) and Baltes et al. (2018), i.e., the question bodies contain code snippets and tags, so the knowledge obtained in Chapter 2 is applicable.

With that in mind, we built four pre-processing pipelines tested against each combination of hyperparameters defined in the Chapter 4.3.2. Each pipeline combines a subset of the nine steps we developed based on prior research described in section 4.2.1 whose order is described in section 4.2.2.

4.2.1 Pre-processing steps

The step `__remove_code` expects a string representing the HTML body of a question and returns a string. It works by removing all tags and code snippets from the body of a given question, wrapping it into a *div* tag, and then performing a depth-first search into its children, removing all *code* and *pre* tags. Only the inner text of the other tags, such as *anchor*, *strong*, *paragraph*, etc, and text that is directly a child of a given tag is kept. The resulting text is then trimmed and joined with a white space. This step is based on work by Barua et al. (2014), Bagherzadeh and Khatchadourian (2019), Han et al. (2020), Openja et al. (2020), Al Alamin et al. (2021), El aoun et al. (2021), and Peruma et al. (2022). An example of a question’s HTML body is presented below:

```

<p>I am working on a Dafny project (...) </p>
<p>This is a small example </p>
<pre><code>class A {

    var v : int
    // this is ok
    method M ()
        modifies this
        ensures v == old(v)
    (...)
}</code></pre>

```

The step `__gensim_preprocessing` receives and returns a string, performing common pre-processing tasks using the functions `strip_numeric`, `strip_multiple_whitespaces`, and `strip_punctuation` from the Gensim⁶ library. As the name of the functions implies, this step removes digits from the initial string and multiple consecutive white spaces, leaving just one separating the words, and finally strips all the punctuation from the input string. The removal of digits and punctuation of the input string is based on work by Bagherzadeh and Khatchadourian (2019), Han et al. (2020), Openja et al. (2020), Al Alamin et al. (2021), El aoun et al. (2021), and Peruma et al. (2022).

The step `__lemmatize_body` takes a string and returns a list of strings, representing words. This step starts by tokenizing the input string using NLTK's `word_tokenize` function. After tokenizing, we apply stemming using NLTK's WordNet lemmatizer. This lemmatizer needs to receive the part of speech of each word. Since identifying specific parts of speech is out of the scope of our work, we decided to consider every word as a verb since we found that verbs and plural nouns were transformed into their stems. The choice for this specific lemmatization model is supported by the previous work of Han et al. (2020).

The step `__remove_stop_words` takes a list of words as input and returns a list of words. It filters out stop words using the NLTK stop words corpus, such as “a”, “an”, “the”, “of” and “and”. This is an approach used by Han et al. (2020), Abdellatif et al. (2020), and Peruma et al. (2022).

The step `__transform_bigrams` takes a list of words and returns a list of bigrams using NLTK `bigrams` function. This step was also used by Abdellatif et al. (2020), although the authors used Gensim to build the bigrams.

The step `__make_lower_case` simply takes a list of words and transforms them into their lowercase version.

The step `__remove_handpicked` removes words that, after examination, we considered as not adding value to the trained model. Table A.2 contains the full list of removed words. This step is based on the works by Al Alamin et al. (2021) and Peruma et al. (2022).

The step `__remove_uncommon` removes words with less than three uses since these words can not possibly form patterns.

⁶<https://radimrehurek.com/gensim/>

The step `__remove_small` removes words with less than two characters since we considered that words with that size could not express meaningful value.

4.2.2 Pre-processing pipelines

This section describes the four pipelines we developed to process the questions. All the pipelines are based on Han et al. (2020) because they lemmatize the words before removing the stop words. We followed this work because we determined that the number of stop words identified increased by 257 when the words were already stemmed.

- *baseline*: This pipeline is the simplest and produces a list of individual words. The exact sequence of the tasks is depicted in Figure 4.2.
- *baseline_bigram*: This pipeline creates bigrams from the list of words resulting from the baseline pipeline. It is based on the work by Abdellatif et al. (2020). The exact sequence of the tasks is depicted in Figure 4.3.
- *rem_words_baseline*: This pipeline is similar to the regular baseline pipeline but incorporates the innovation of Al Alamin et al. (2021) and Peruma et al. (2022) of removing domain-specific words as well as our innovation of removing words based on size and frequency. The exact sequence of the tasks is depicted in Figure 4.4.
- *rem_words_bigram*: This pipeline is based on *rem_words_baseline* but, similarly to *baseline_bigram*, ends with a step that transforms the resulting list of words in a list of bigrams. The exact sequence of the tasks is depicted in Figure 4.5.

Figure 4.2: Sequence of tasks for the *baseline* pipeline

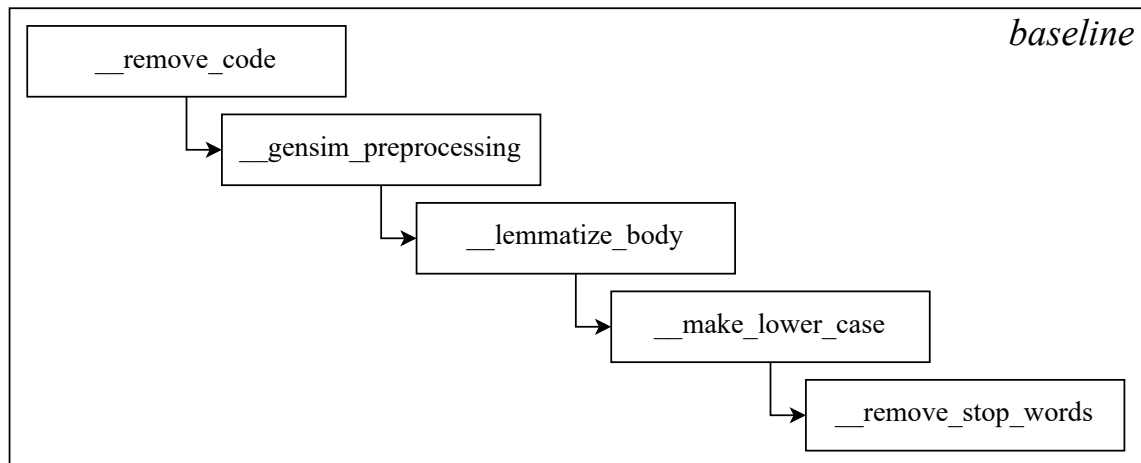


Figure 4.3: Sequence of tasks for the *baseline_bigram* pipeline

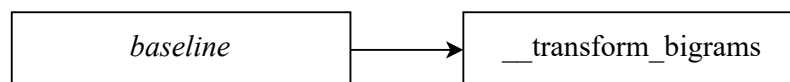
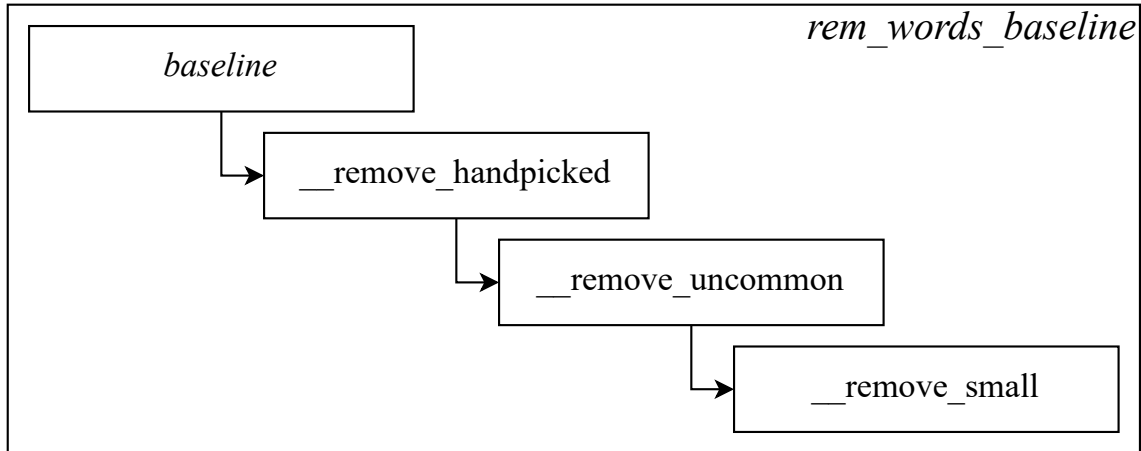
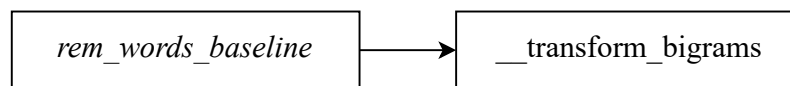


Figure 4.4: Sequence of tasks for the *rem_words_baseline* pipelineFigure 4.5: Sequence of tasks for the *rem_words_bigram* pipeline

4.3 Model Training

This section describes the methodology we used to train the models. In subsection 4.3.1, we describe the overall training methodology, namely how we overcame the non-deterministic nature of the algorithm. In subsection 4.3.2, we describe the hyperparameters chosen to configure the model.

4.3.1 Training Methodology

The specifics about the creation of model configurations are shown in the section 4.3.2. For each configuration, we trained 4 models with the same hyperparameters.

Regarding evaluation, we calculated the average for each configuration as well as its standard deviation.

Taking into consideration LDA's non-deterministic nature, this strategy allows to detect if a score is exceptionally good or if the hyperparameters are able to make LDA consistently output good models.

4.3.2 Hyperparameter Tuning

In regards to hyperparameter tuning, we implemented grid search, training four models for each combination of the parameters enumerated below:

- *K* from 2 to 40 with a step of 2. This value is both in line with previous research analyzed in Chapter 2 and allows us to include all the challenges identified in Chapter 3.

- K from 500 to 3000 with a step of 500 in line with previous research. We also included 50 iterations because it is the default value of Gensim.
- Number of passes: we included 1 as it is the default value from Gensim and 5 from Han et al. (2020).
- β : we included 0.1 and 0.01 from previous literature, and None because it is the default value.
- α : we decided to settle with the formula $\alpha = 50/K$ from the analyzed literature (Bagherzadeh and Khatchadourian (2019); Al Alamin et al. (2021)).

4.3.3 Model Storage

To store the resulting models, we extended the schema defined in section 4.1 by adding three different tables: *model_runs*, *model_configs*, and *models*. *model_runs* was created to aggregate several hyperparameter sets created by grid search. However, we only created one run per database file and assumed only one model run would exist. *model_configs* holds the hyperparameters needed to train a LDA model. Finally, *models* holds all the models trained using the same hyperparameters. Each model configuration was used to train four models, whose performance was compared to understand if it was consistent across each trained model.

The properties for *model_runs*, *model_configs*, and *models* can be found in tables 4.5, 4.6, and 4.7, respectively. The relationship between the tables can be observed in the Figure 4.6.

Figure 4.6: Model storage diagram

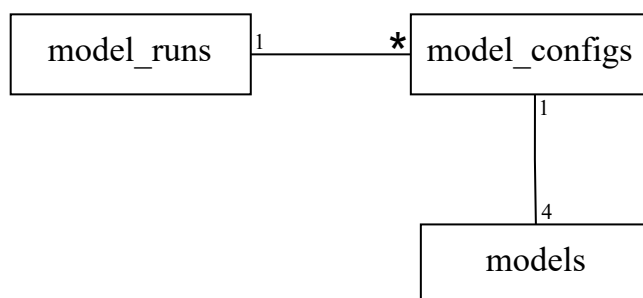


Table 4.5: Description of columns of the *model_runs* table

Column name	Description
id	Id of each model run
dictionary_pickle	This is an object used by Gensim to convert documents to their bag of words representation. Encoded using Python's <i>pickle</i> library
run_date	UNIX timestamp in seconds representing the date of when the model run was created

Table 4.6: Description of columns of the *model_configs* table

Column name	Description
id	Id of each model configuration. Uses a UUID
num_topics	The number of topics, K
iterations	The number of iterations, I
passes	The number of passes
α	The document-topic distribution, α
β	The topic-word distribution, β . Gensim calls this parameter <i>eta</i>
model_run_id	The id of the corresponding model run

Table 4.7: Description of columns of the *models* table

Column name	Description
id	Id of each model. Uses a UUID
model_pickle	The binary corresponding to the trained model Encoded with Python's <i>pickle</i> library
model_config_id	The id of the corresponding model configuration

4.4 Topic Labeling

The final step is topic labeling. The first step was to calculate the document-topic probabilities for each document. Considering t_{prob} as the topic probability and n as the number of topics, we discarded all documents with:

$$t_{prob} < \frac{1}{n}$$

The reasoning is that $\frac{1}{n}$ is the probability of randomly assigning a document to a given topic, so we only accepted probabilities greater than the random assignment.

After filtering the documents with low probabilities, we sorted the documents and performed open card sorting based on the top 30 words and top 20 documents for each topic. The labeling for each topic was discussed among the team members until a consensus was reached.

To aid the labeling of topics, we created a tool that allowed the visualization of questions. Firstly, the user of the tool must input a directory containing a file called *posts.sqlite* corresponding to an SQLite database with the schema described in sections 4.1.1 and 4.3. Below, we present the exact syntax to run the tool, assuming that the user's shell's working directory is the script's root.

```
python .\main.py BASE_PATH CONFIG_ID
```

The argument *BASE_PATH* must be a directory with two files, one called *posts.sqlite* with a database whose schema is according to the one described in sections 4.1.1 and 4.3, and another called *eval.json* with the model's evaluation.

The argument *CONFIG_ID* must be all the characters of the id of a configuration until the first hyphen.

After choosing a configuration, the tool prompts the user to choose the exact model of the four trained for the chosen configuration. The coherence metric is shown near the model IDs to help choose the best model. An illustration of the prompt can be seen in the Figure 4.7

Figure 4.7: Prompt showing the models for the configuration *e1673342*

```
Found models:
1: c7298177-65d0-4920-b79f-30e4cfccee4a (-1.8246387700255642)
2: 1613d2b4-89e4-42de-9307-63375c718ac5 (-1.8253157177534942)
3: fecc82a4-952b-4dda-990a-2a44ce848d26 (-1.8275000440289169)
4: af8e28cf-284d-4fcf-9ee0-0057bb818317 (-1.8244731419438664)
Insert model index: |
```

To avoid bias when classifying the questions, the tool allows viewing the questions grouped by topics and mixed. The view with the questions mixed was used to classify each question, and the view with the questions grouped by tags was used to label the tags. A side-by-side figure comparing both views can be found in Figure 4.8. The tool also provides a view of the questions, which aided us in evaluating the stored version instead of the most recent one in case the questions were edited in the meantime. Figure 4.9 illustrates the view of a post using question 399.

Figure 4.9: Example of post visualization

```
399: Can I find non-spurious counter example if I use different Boogie backend to check translated bpl files by Dafny?

As mentioned in the Wiki on Dafny GitHub, when Dafny cannot prove an assertion in a program, it might be due to either my program is incorrect OR the incompleteness of Dafny. But I figured the counter example from Dafny is spurious after I tried understanding it, so I still have no idea if my program is correct or not.

My question is as follows. If I manage to use another Boogie backend such as Corral to check the translated Boogie program from Dafny using /print, and Corral also returns a model to invalidate the Boogie program, would that guarantee the model disprove my Dafny program and I can use it for debugging? Or is it still likely spurious so don't bother to try it?

From their papers, it seems to me that Corral and Symbooglix should guarantee that a provided model should be a concrete counter example for the translated Boogie program, so maybe my question is more on whether the Dafny program and the translated Boogie program is semantically equivalent.

PS: I tried passing translated bpl files to Corral and check specific Boogie procedure in that bpl, and Corral simply says it cannot find the procedure, so I'm debating if I want to make it work.

Tags: dafny, boogie

Research Tags:

Link: https://stackoverflow.com/questions/52382898/can-i-find-non-spurious-counter-example-if-i-use-different-boogie-backend-to-che
Create date: 2018-09-12 20:25:41
Fetch date: 2024-03-21 15:58:25
```

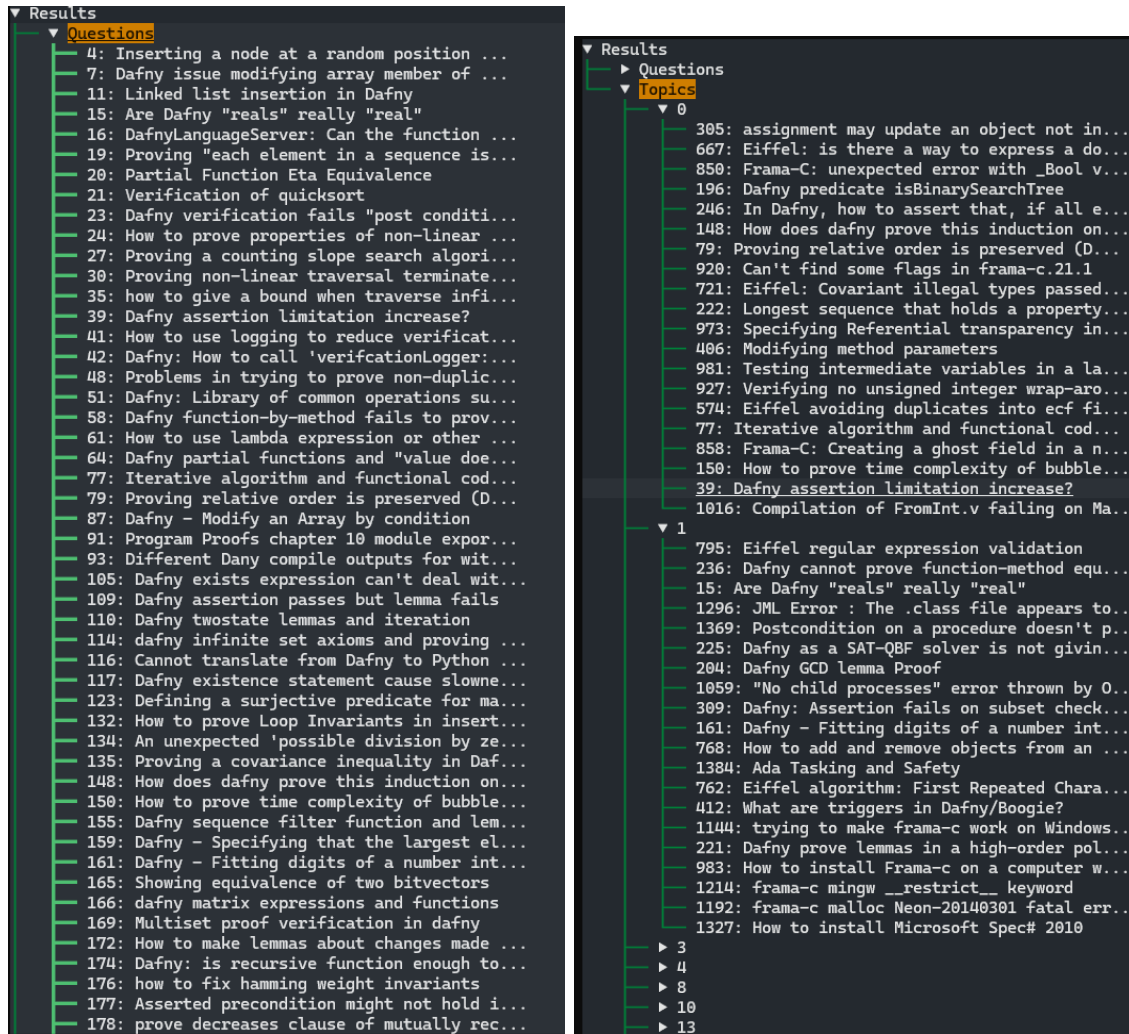
4.5 Conclusion

In conclusion, this chapter presents the details of the methodology that we used to gather, pre-process, model, and categorize the questions.

In Section 4.1, we elaborate on the reasoning behind the choice of StackOverflow's official API to gather the question and the filters that were used so that only wanted data was returned. Additionally, we present the database schema for creating a SQLite 3 database to store the questions and associated metadata.

Section 4.2 presents four pipelines derived from the literature used to pre-process the question's bodies. Their performance was evaluated at Chapter 5

Figure 4.8: Comparison between mixed and topic views



In Section 4.3, we explain what hyperparameters will be used when training the model and the methodology that combines them and stores the trained models.

Finally, in Section 4.4, we explored the methodologies and tools we created to classify the questions and topics.

Chapter 5

Results

As presented in Chapter 4, we used topic modeling to identify topics of similar challenges. This chapter analyzes the results of the model training following the aforementioned methodology.

In sections 5.1 and 5.2, we analyze the model training results with all the SO questions and, secondly, without the questions considered unsuited after the first training.

In sections 5.3 and 5.5, we identify the tags discovered during open card sorting, where we tagged each question without predefined categories. Finally, we identify the challenges from the relevant topics.

In this chapter, we will interchangeably refer to the terms logarithm of perplexity and perplexity when evaluating models. For a deeper understatement of the evaluation metrics used, we refer to Chapter 2, more specifically, Section 2.3.1.

5.1 First Model Training Performance Analysis

The methodology presented in Chapter 4 produced: four model runs, one for each pipeline; 840 model configurations resulting from the combination of LDA hyperparameters; and 3360 models, four for each configuration. This section provides an in-depth look into the performance of some of the model configurations for each of the used number of topics, i.e., a number of topics ranging from 2 to 40 with a step of 2. We analyzed the configurations with the highest average coherence (C_{UMass}) score by number of topics as well as the configurations with the lowest standard deviation. We also analyzed each model configuration found using the aforementioned criteria and analyzed its average perplexity score and standard deviation.

Both *baseline* and *baseline_bigram* will be used to check if the removal of additional words in the *rem_words_baseline* and *rem_words_bigram* improve the model's performance.

When looking at the performance of the configurations with the highest average scores for the *baseline* pipeline, depicted in Figure 5.1, we observe that all the configurations resulted in significantly high standard deviations that resulted in models with different numbers of topics having indistinguishable performance. On top of that, the high standard deviation indicates that the configurations were unstable. The same phenomena occur in the lowest deviation plot depicted

in Figure 5.3. However, the perplexity score for the highest average and lowest error depicted in figures 5.2 and 5.4, respectively, is stable, maintaining low standard deviations.

When considering the highest average scores for the *rem_words_baseline* pipeline, the performance, both in terms of coherence and in terms of perplexity, depicted in figures 5.5 and 5.6, respectively, is similar to the corresponding evaluation criteria for the *baseline* although with a small degradation. This is expected due to removing words that, although common, i.e., coexist with many other words, have no meaning. When looking at the lowest deviation plot for coherence depicted in Figure 5.7, we verify a reduction in the standard deviation of each configuration, letting us conclude that the configurations are stable.

Figure 5.1: Average coherence score by number of topics for the *baseline* pipeline (Highest value)

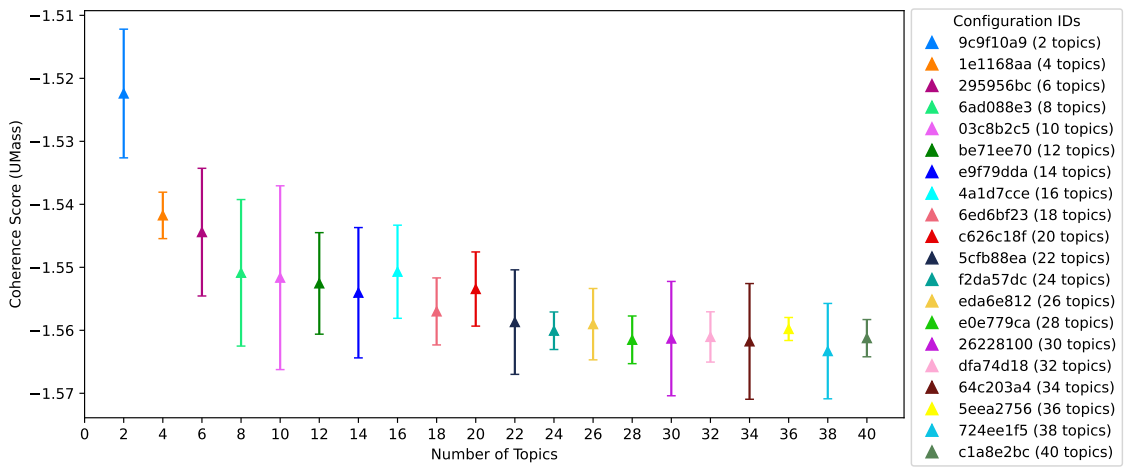


Figure 5.2: Average perplexity score by number of topics for the *baseline* pipeline (Highest value)

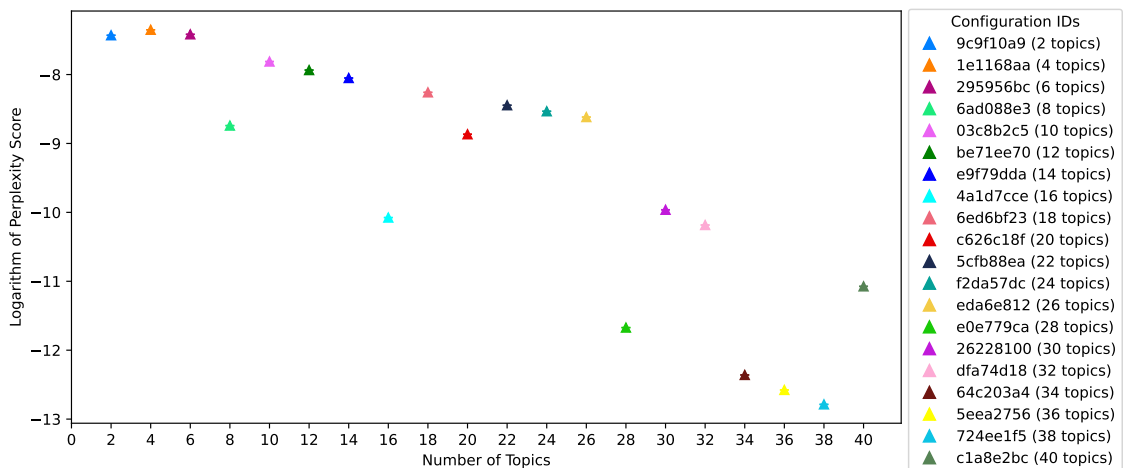


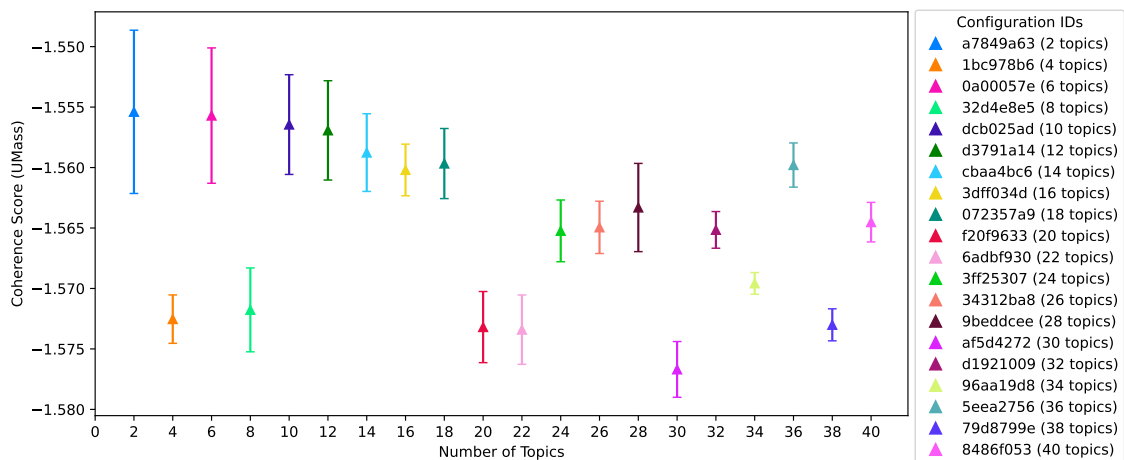
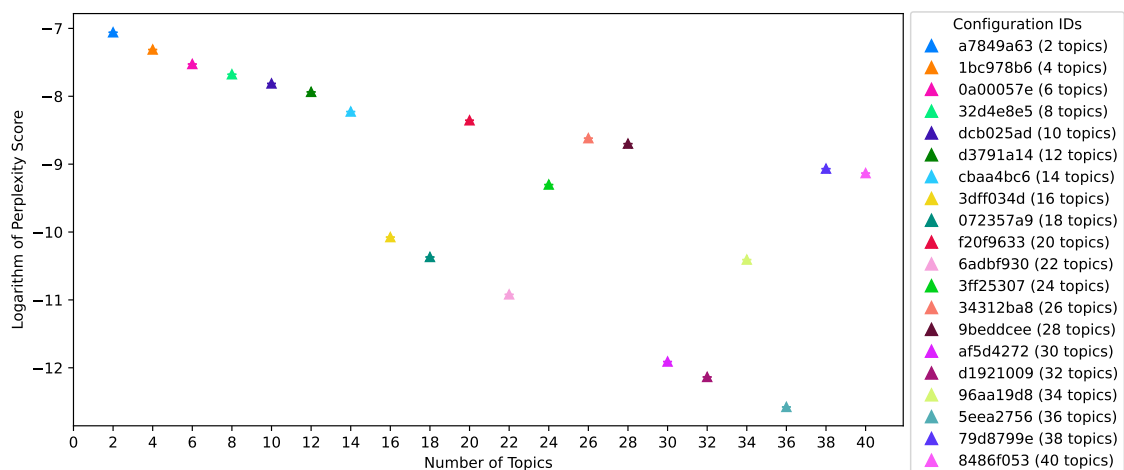
Figure 5.3: Average coherence score by number of topics for the *baseline* pipeline (Lowest deviation)Figure 5.4: Average perplexity score by number of topics for the *baseline* pipeline (Lowest deviation)

Figure 5.5: Average coherence score by number of topics for the *rem_words_baseline* pipeline (Highest value)

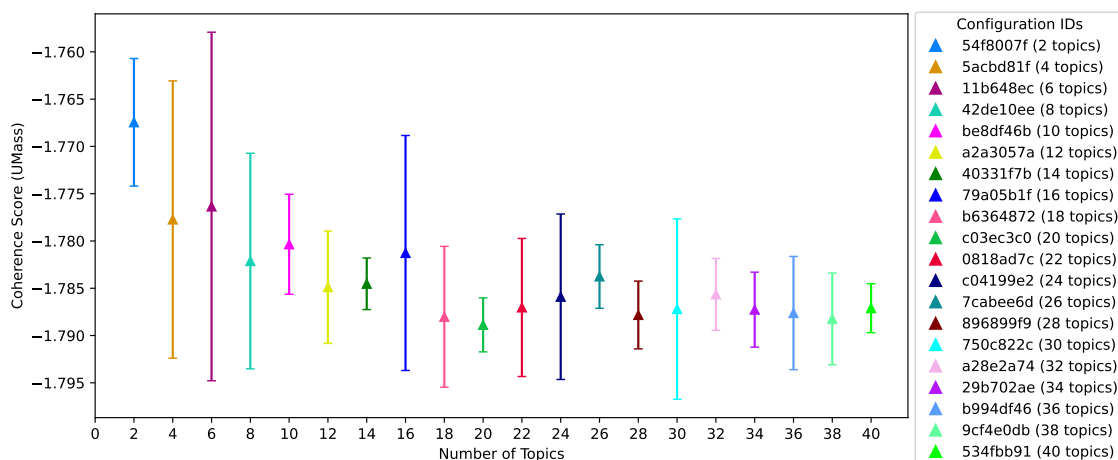


Figure 5.6: Average perplexity score by number of topics for the *rem_words_baseline* pipeline (Highest value)

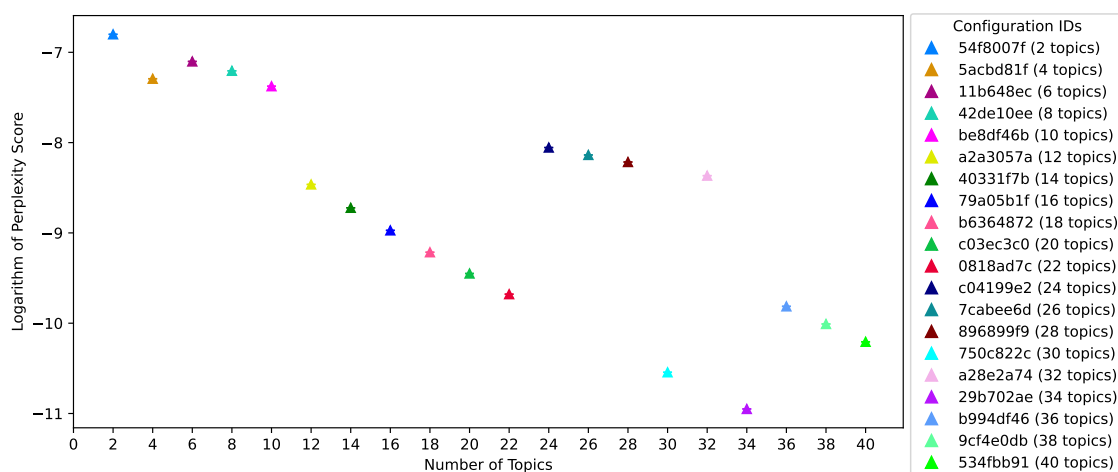


Figure 5.7: Average coherence score by number of topics for the *rem_words_baseline* pipeline (Lowest deviation)

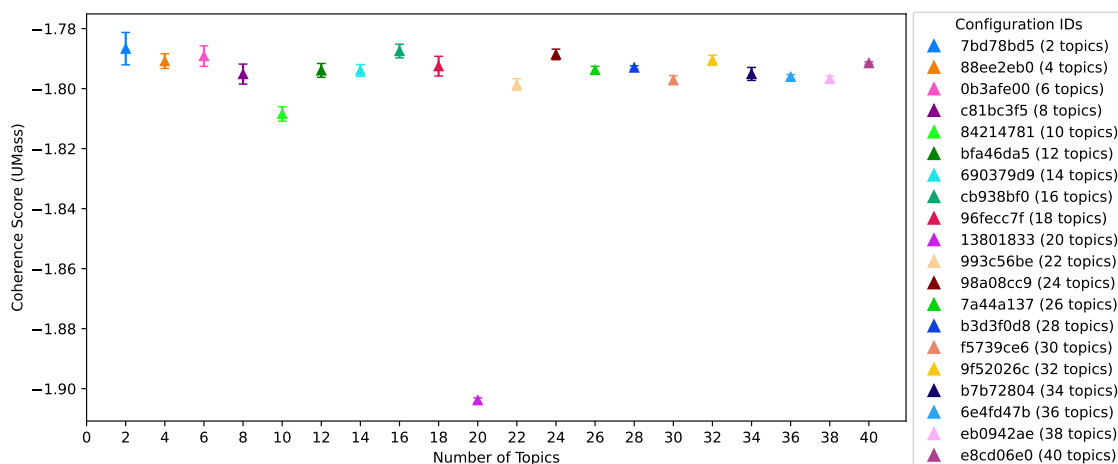
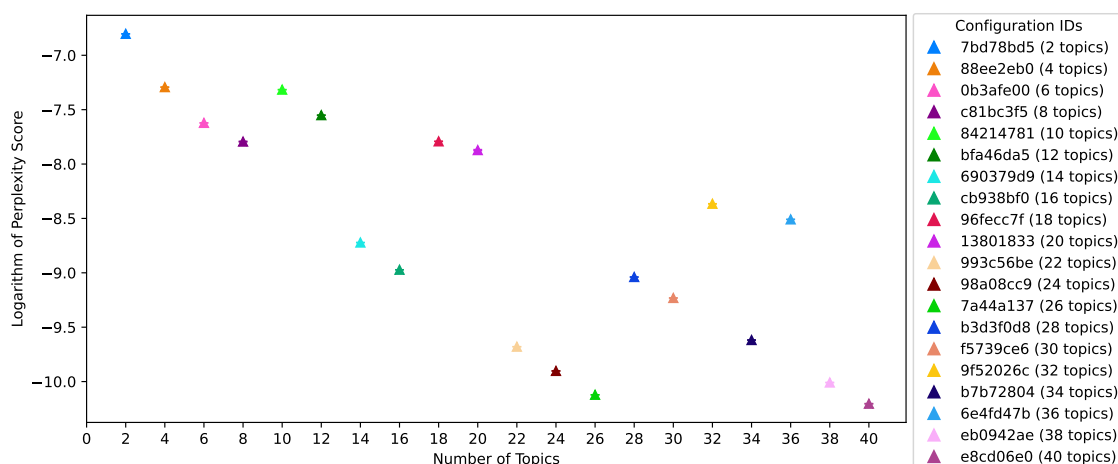


Figure 5.8: Average perplexity score by number of topics for the *rem_words_baseline* pipeline (Lowest deviation)



Looking at the coherence score for the *baseline_bigram* pipeline, depicted in Figure 5.9, we can observe the same overall score variability as observed with the *baseline* pipeline. We can also observe an overall lower coherence score, and although scores between different corpus should not be compared, this might indicate that fewer relationships between pairs of words, in this case, bigrams, were found. These performance results might impact the model's pattern recognition capabilities in identifying multiple-word expressions. The same happens for the coherence of the lowest error model configurations, depicted in Figure 5.11. There is, however, an improvement in the perplexity score in both models with the highest coherence and lowest coherence error.

Regarding the pipeline *rem_words_bigram*, we observe an even higher degradation of coherence score in both the highest coherence average and lowest error; this is depicted in figures 5.13

and 5.15 respectively. Improvements in both criteria perplexity scores were also observed (Figure 5.14 and 5.16).

Figure 5.9: Average coherence score by number of topics for the *baseline_bigram* pipeline (Highest value)

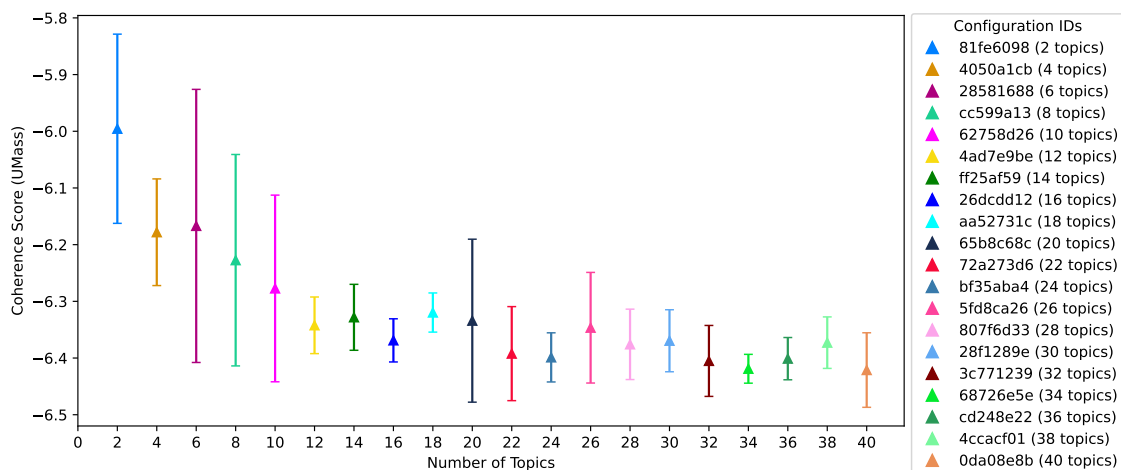


Figure 5.10: Average perplexity score by number of topics for the *baseline_bigram* pipeline (Highest value)

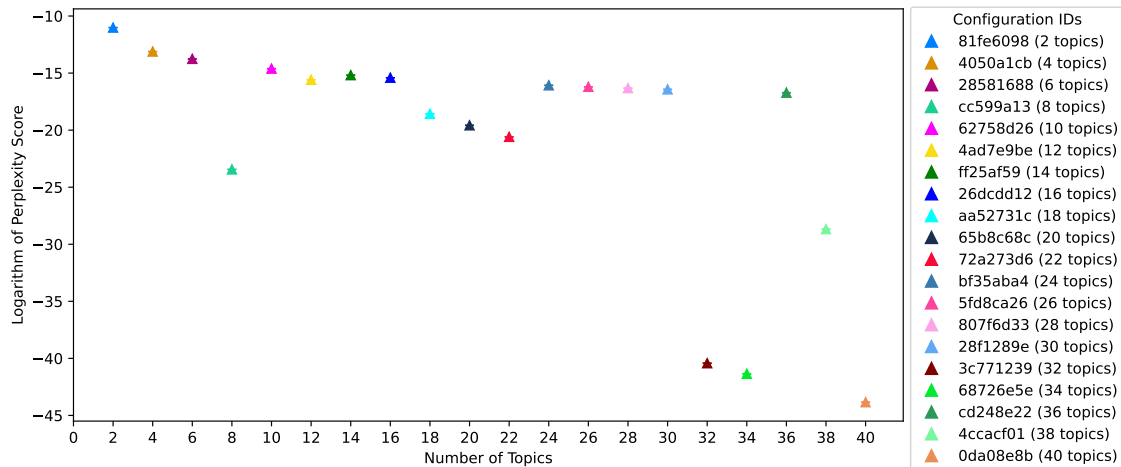


Figure 5.11: Average coherence score by number of topics for the *baseline_bigram* pipeline (Lowest deviation)

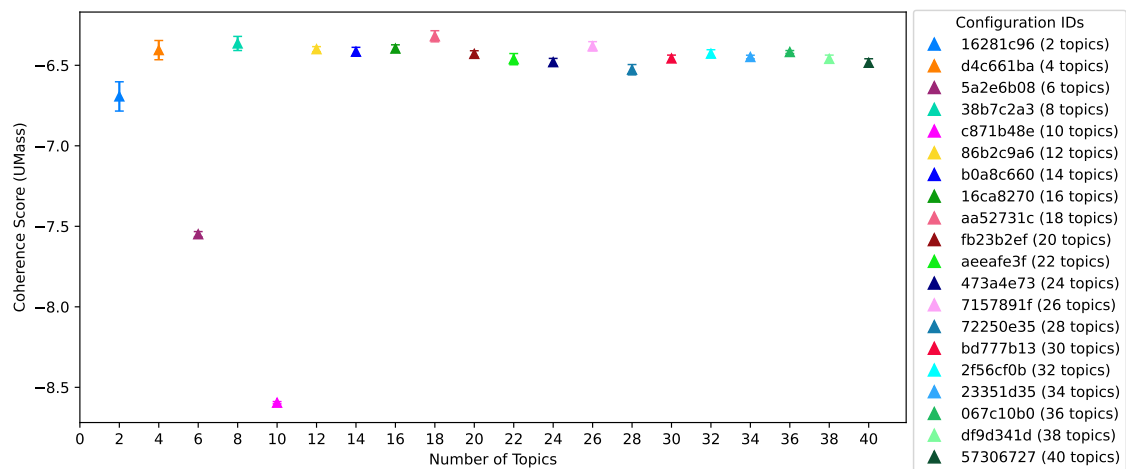


Figure 5.12: Average perplexity score by number of topics for the *baseline_bigram* pipeline (Lowest deviation)

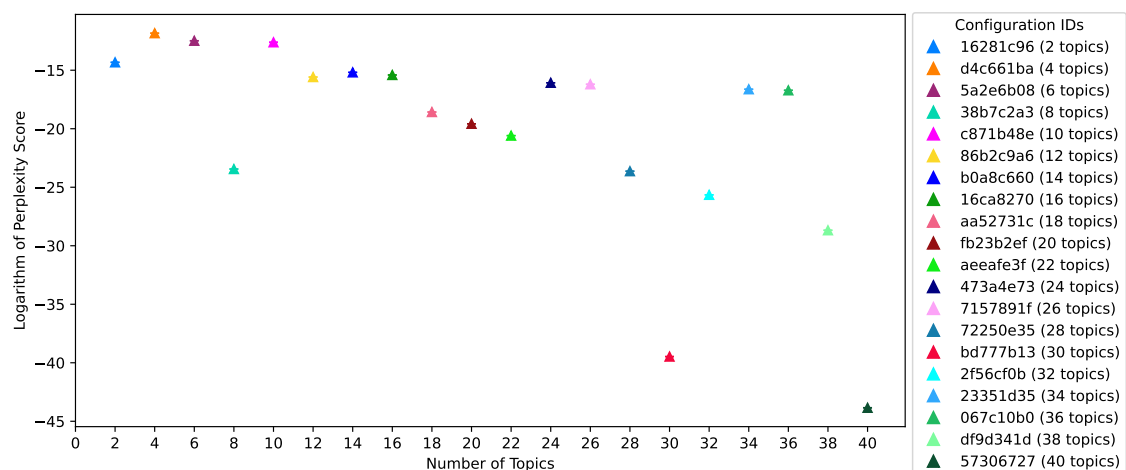


Figure 5.13: Average coherence score by number of topics for the *rem_words_bigram* pipeline (Highest value)

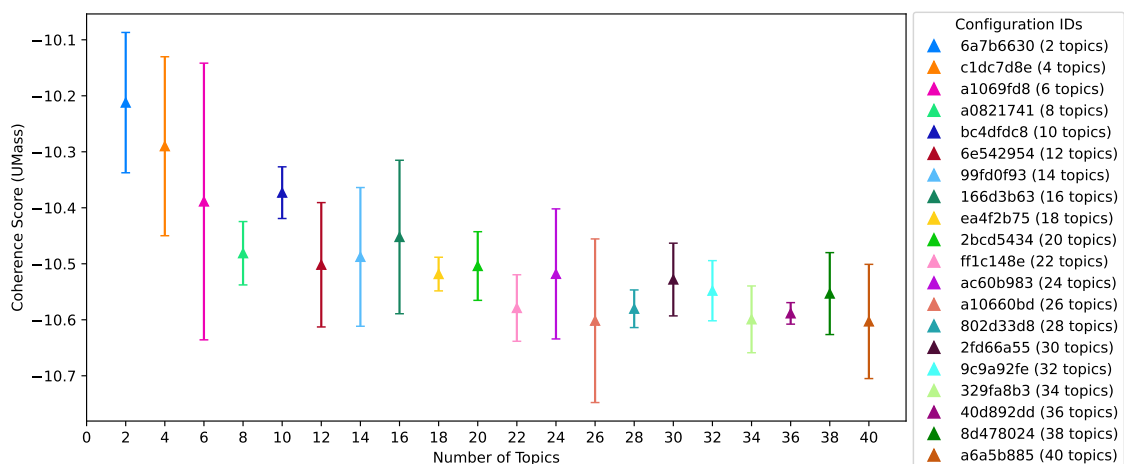


Figure 5.14: Average perplexity score by number of topics for the *rem_words_bigram* pipeline (Highest value)

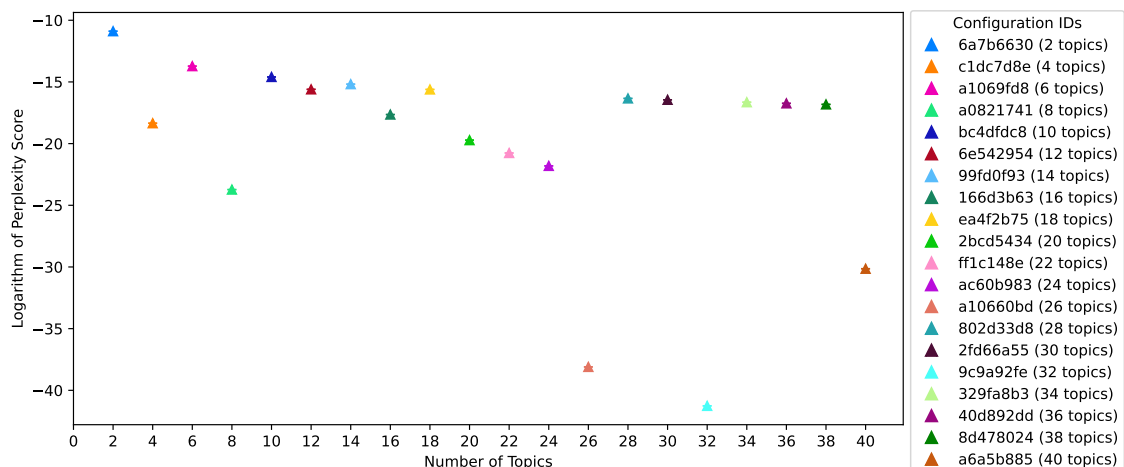


Figure 5.15: Average coherence score by number of topics for the *rem_words_bigram* pipeline (Lowest deviation)

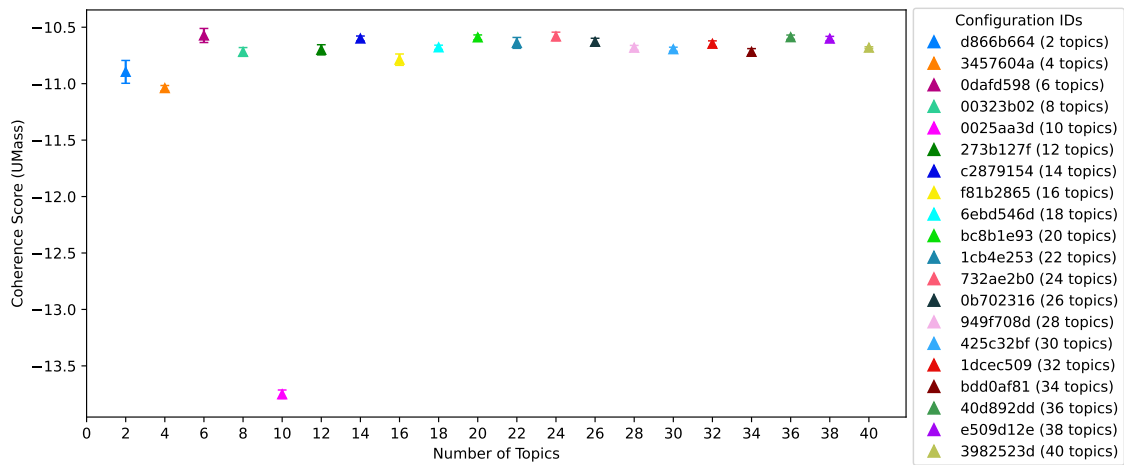
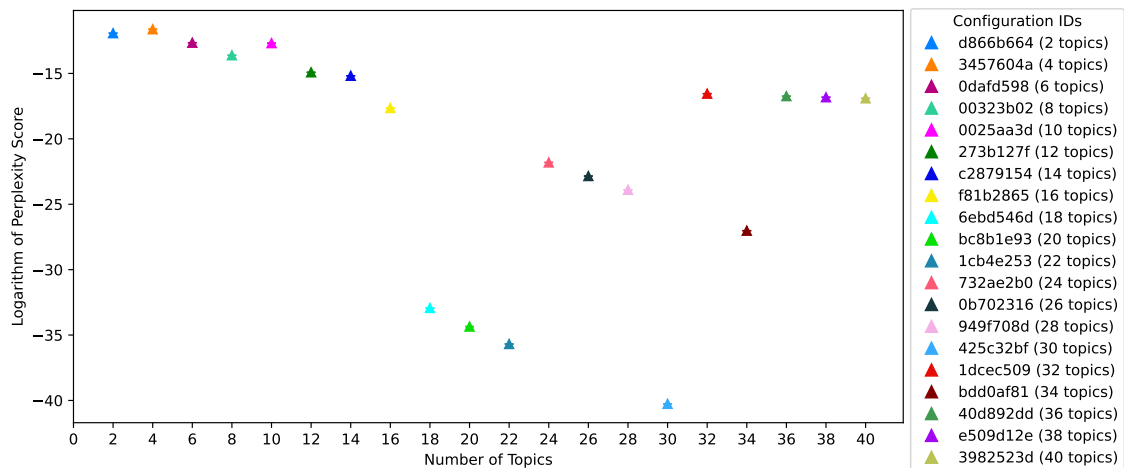


Figure 5.16: Average perplexity score by number of topics for the *rem_words_bigram* pipeline (Lowest deviation)



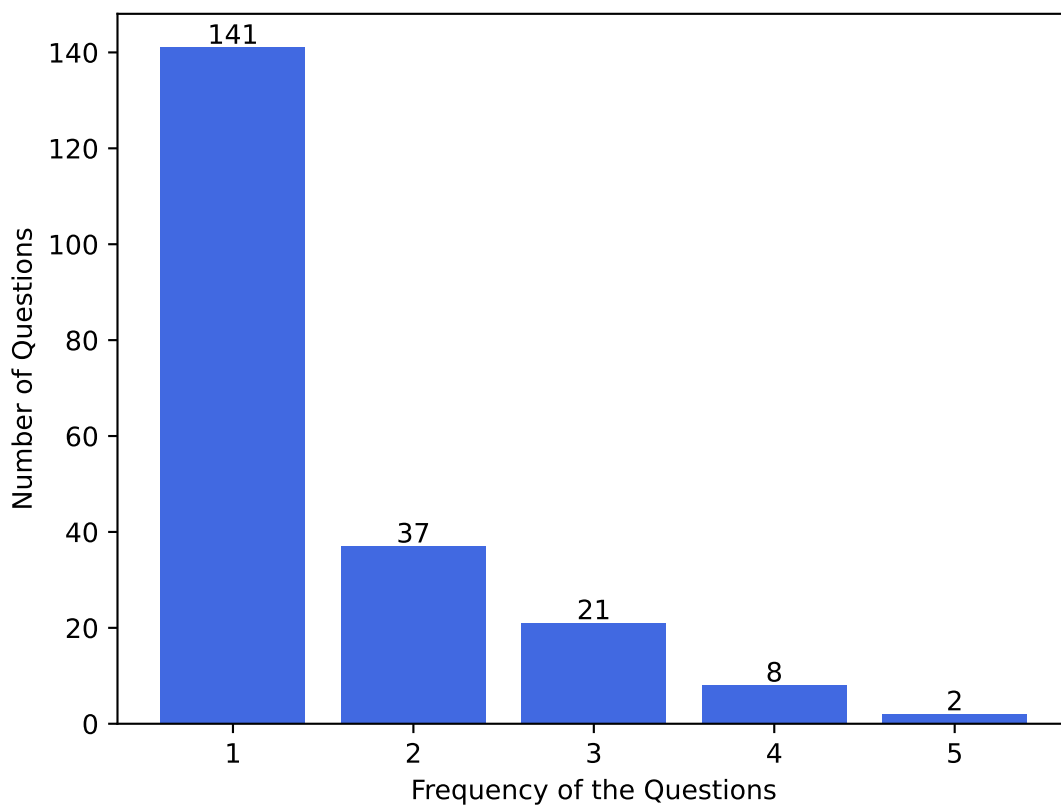
We chose to pick the best configuration from the *rem_words_baseline* pipeline because it had the overall best performance and, unlike *baseline*, it did not contain the common words that gave no hint to discover the answers to RQ2.

Due to the high error in the highest average coherence, we decided to pick a configuration from the lowest error configurations.

Finally, we chose the model with 16 topics, whose id is *edc8afc0*, from configuration *cb938bf0*, despite having a lower perplexity score than the neighboring models with a similar number of topics.

Looking at the composition of the topics, we verify that we obtained 209 unique questions from the 320 that comprise the 16 topics. As shown in Figure 5.17, most questions appear only once, indicating diverse topics.

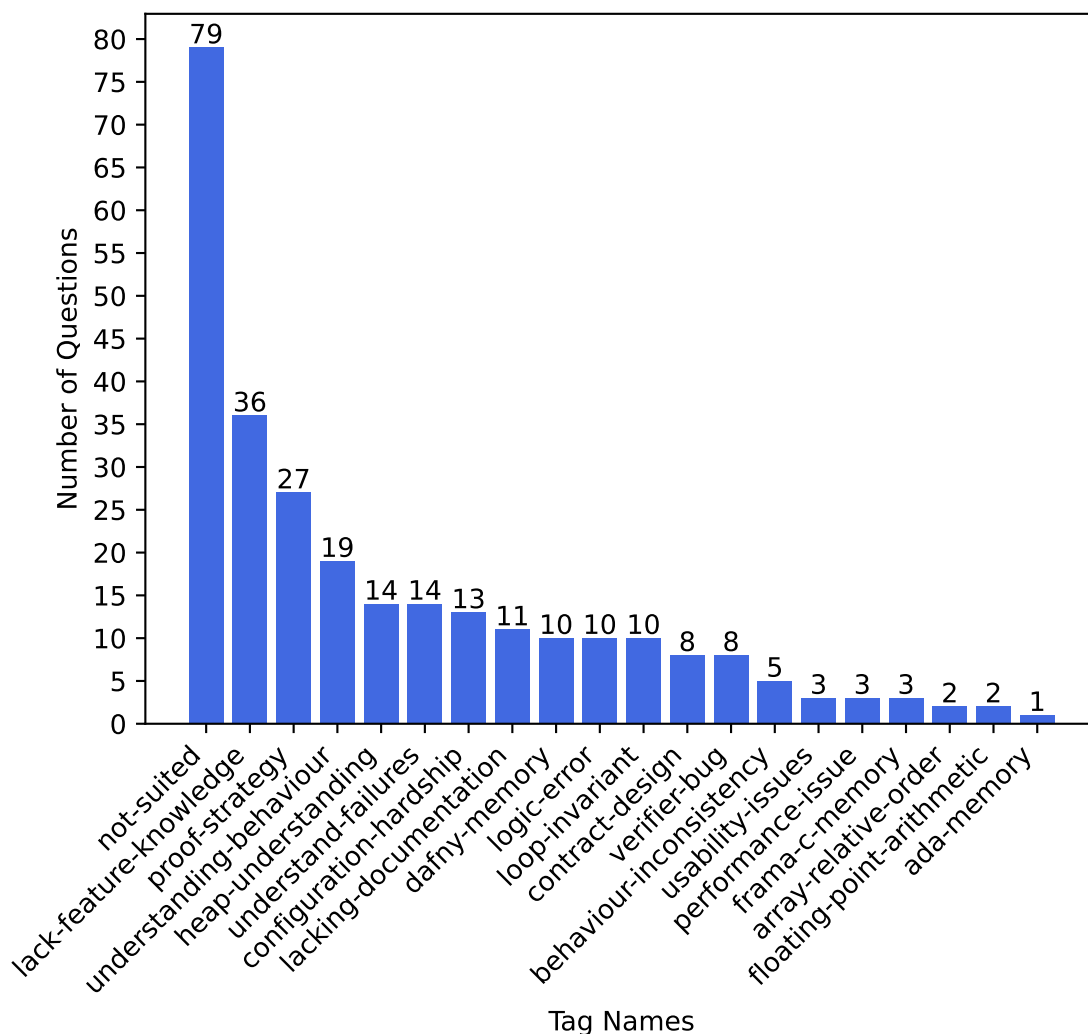
Figure 5.17: Distribution of document frequencies for configuration cb938bf0



After applying open card sorting to the clustered questions, we obtain the tag distribution depicted in Figure 5.18. The most frequent tag is the non-suited tag with 79 questions, meaning that around 38% of questions were considered inadequate to answer RQ2. Of the 16 resulting topics, in 15, the most frequent tag is non-suited, indicating that the topics are not representative of any challenge. In appendix B, we present the distribution of tags for each topic.

To overcome this issue, we decided to filter out these questions and run the training now without the 79 inadequate questions. The result of the second try is analyzed in Section 5.2.

Figure 5.18: Frequency of each tag



5.2 Second Model Training Performance Analysis

In this section, we analyze the results of the second run of the model training after removing the questions identified as inadequate in Section 5.1. This strategy produced four model runs, one for each pipeline; 840 model configurations resulting from the combination of LDA hyperparameters; and 3360 models, four for each configuration.

Similarly to what was observed in the previous section, the baseline pipeline produces models with significant error, resulting in overlapping performance across the configurations in terms of coherence (figures 5.19 and 5.21). The performance is in the same order of magnitude as the previous run both in terms of coherence and in terms of perplexity (Figures 5.20 and 5.22).

Figure 5.19: Average coherence score by number of topics for the *baseline* pipeline (Highest value)

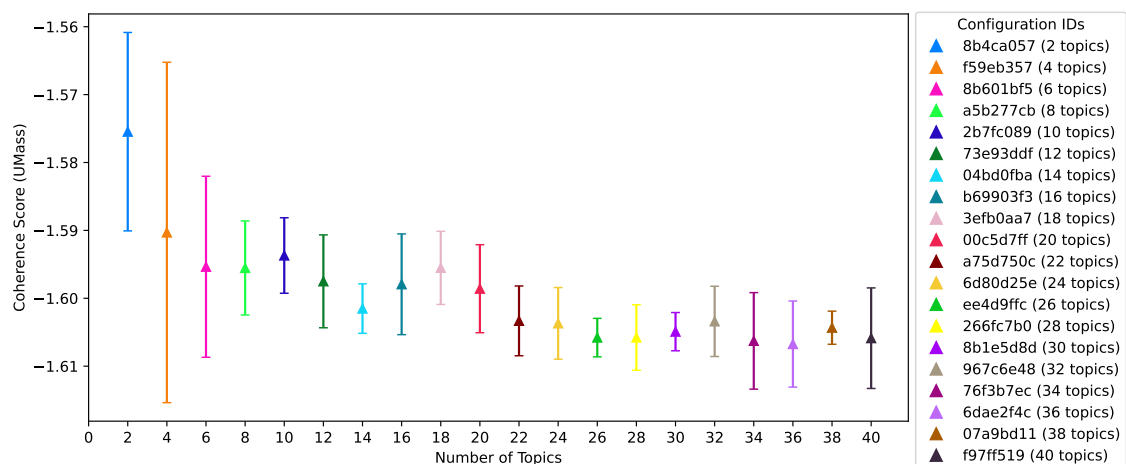


Figure 5.20: Average perplexity score by number of topics for the *baseline* pipeline (Highest value)

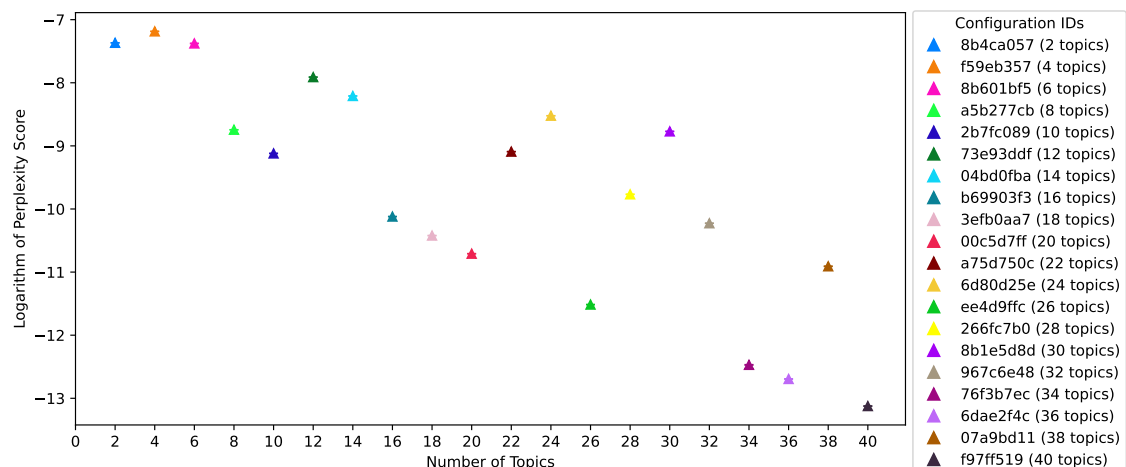
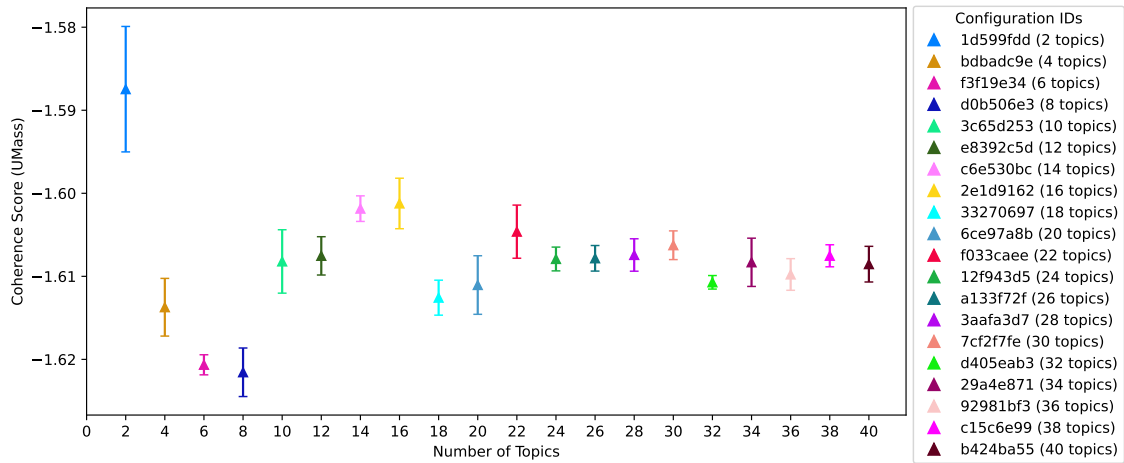
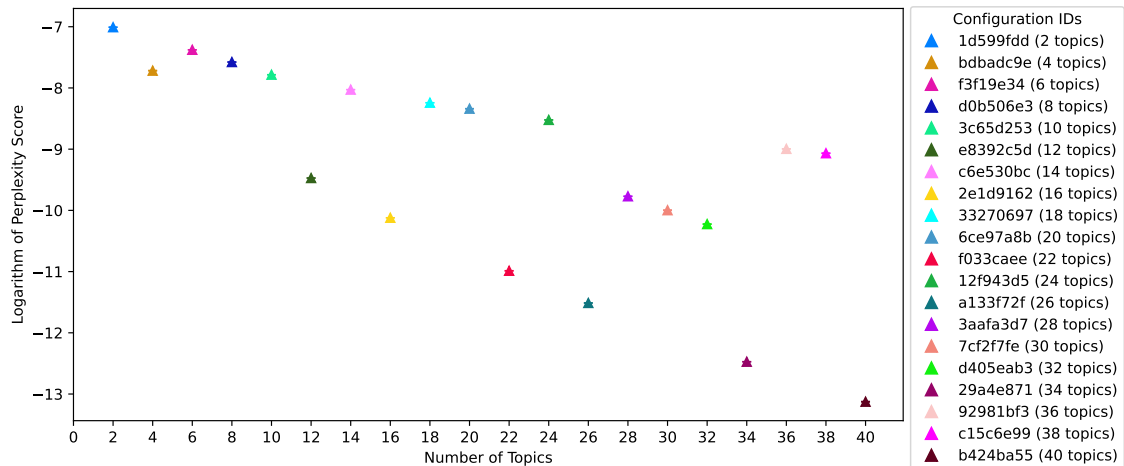


Figure 5.21: Average coherence score by number of topics for the *baseline* pipeline (Lowest deviation)Figure 5.22: Average perplexity score by number of topics for the *baseline* pipeline (Lowest deviation)

Again, like what was verified in the previous section, the *rem_words_baseline* pipeline results in models with a high error regarding the average of the coherence measure (Figure 5.23), maintaining low errors regarding the perplexity score (Figure 5.24). Considering the configurations filtered by the lowest error, both the coherence and perplexity, observed in figures 5.25 and 5.26 respectively, have a low error allowing the conclusion that the models are consistent. Moreover, the coherence and perplexity values align with those observed in the previous section.

Figure 5.23: Average coherence score by number of topics for the *rem_words_baseline* pipeline (Highest value)

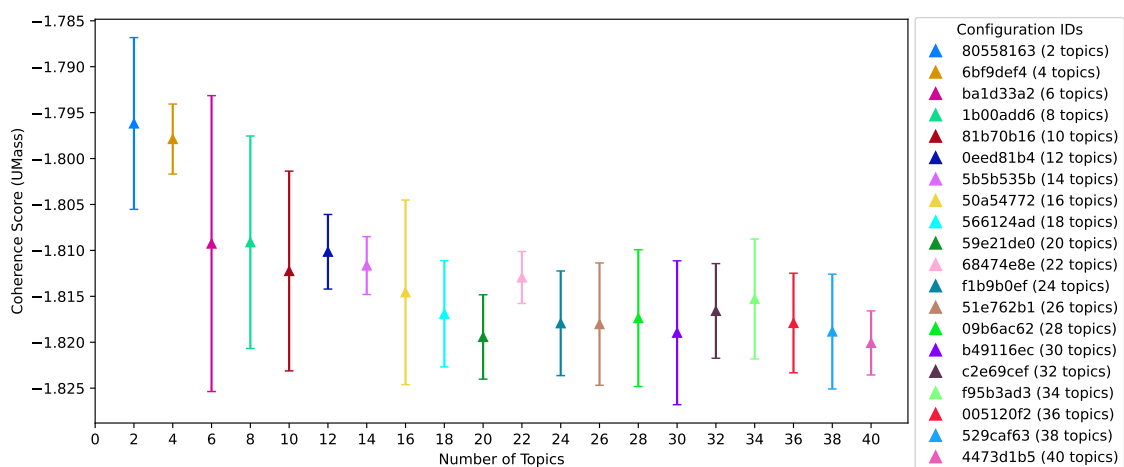


Figure 5.24: Average perplexity score by number of topics for the *rem_words_baseline* pipeline (Highest value)

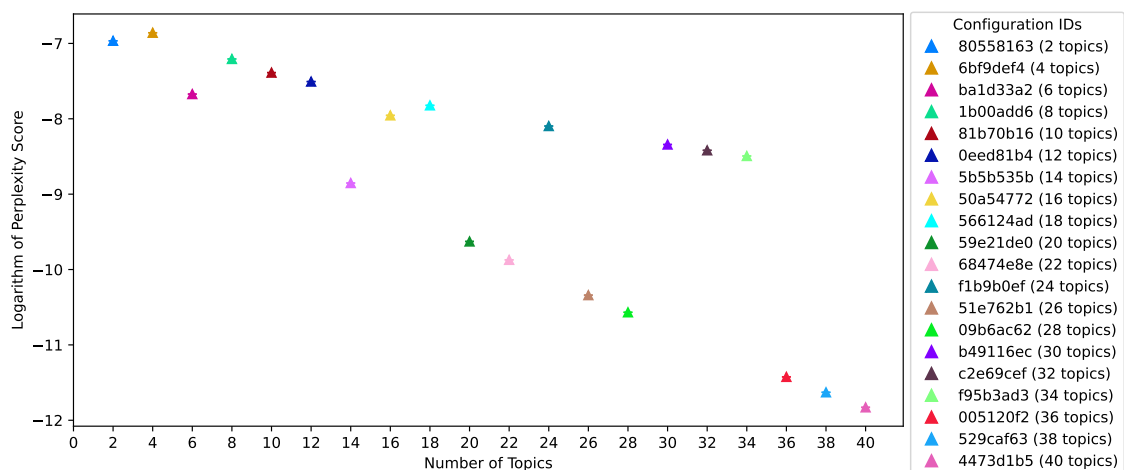


Figure 5.25: Average coherence score by number of topics for the *rem_words_baseline* pipeline (Lowest deviation)

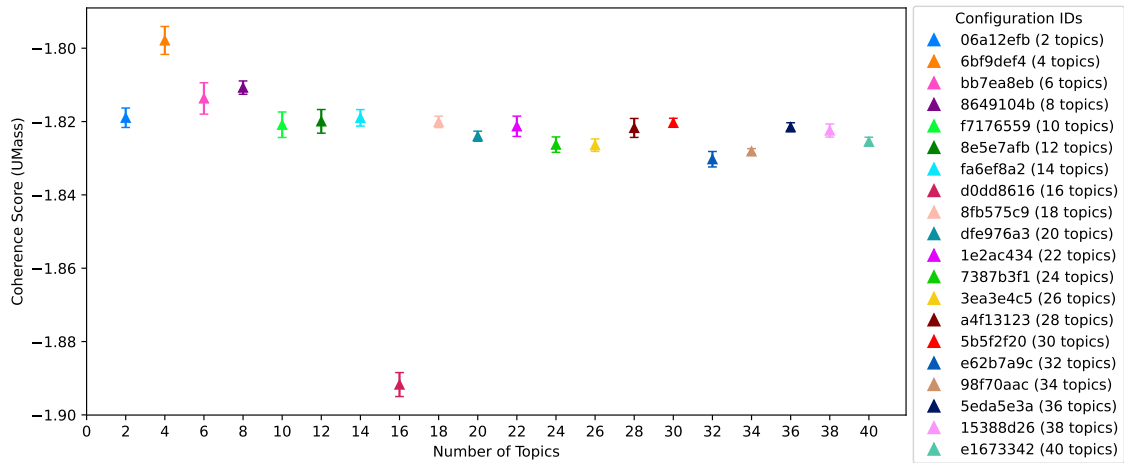
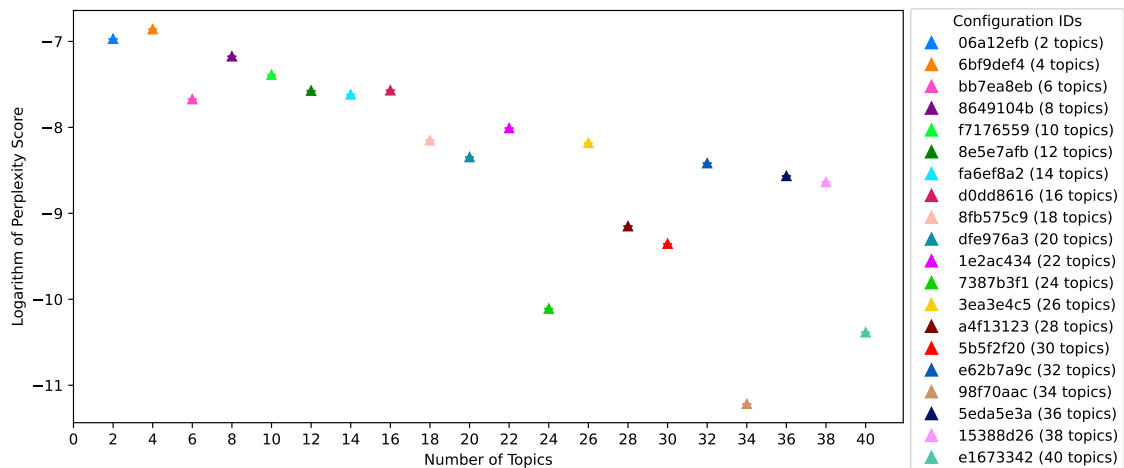


Figure 5.26: Average perplexity score by number of topics for the *rem_words_baseline* pipeline (Lowest deviation)



The performance of both *baseline_bigram* and *rem_words_bigram* is similar to the one observed in the previous section both in terms of highest average coherence, depicted in figures 5.27 and 5.31, respectively, and lowest error for coherence depicted in figures 5.29 and 5.33. In terms of perplexity for *baseline_bigram*, depicted in the figures 5.28 and 5.30, and for *rem_words_bigram*, depicted in the figures 5.32 and 5.34, we observe a similar low score deviation in both pipelines.

Figure 5.27: Average coherence score by number of topics for the *baseline_bigram* pipeline (Highest value)

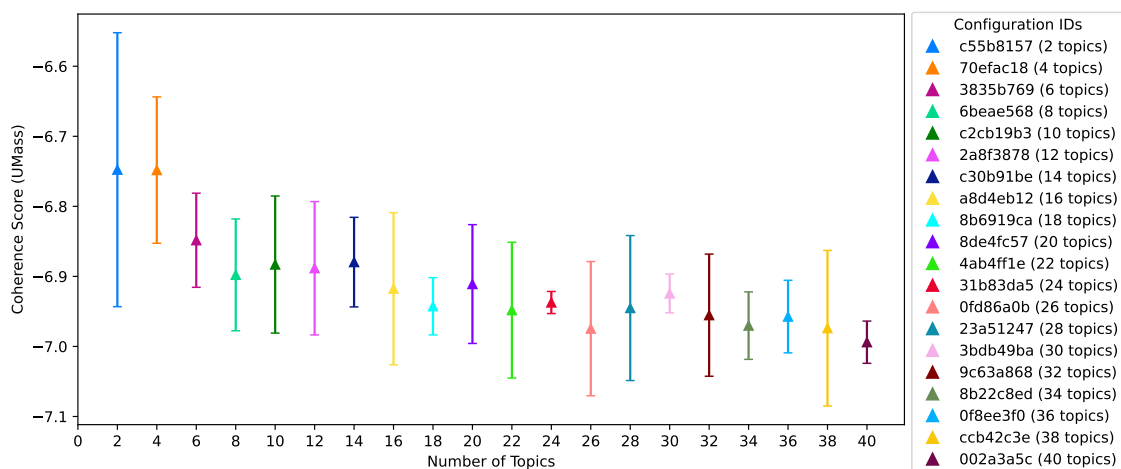


Figure 5.28: Average perplexity score by number of topics for the *baseline_bigram* pipeline (Highest value)

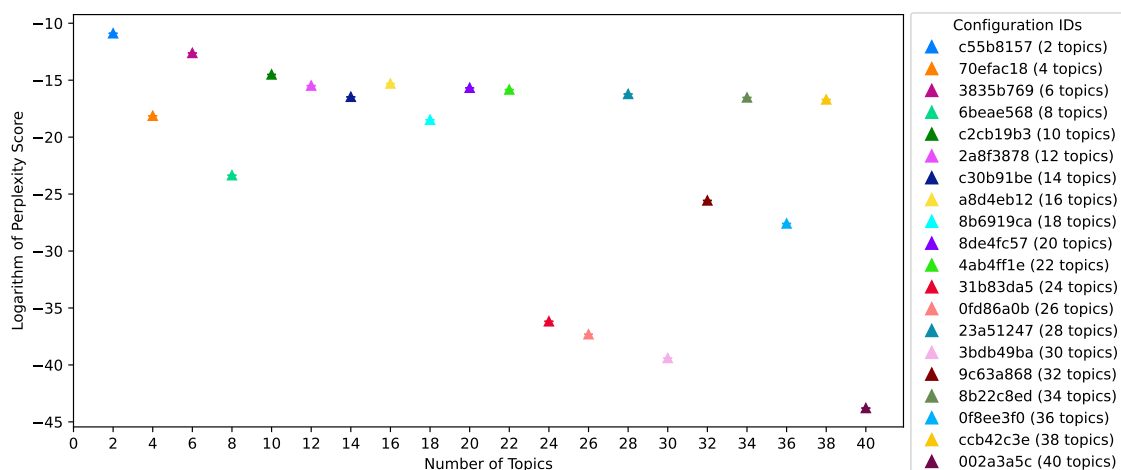


Figure 5.29: Average coherence score by number of topics for the *baseline_bigram* pipeline (Lowest deviation)

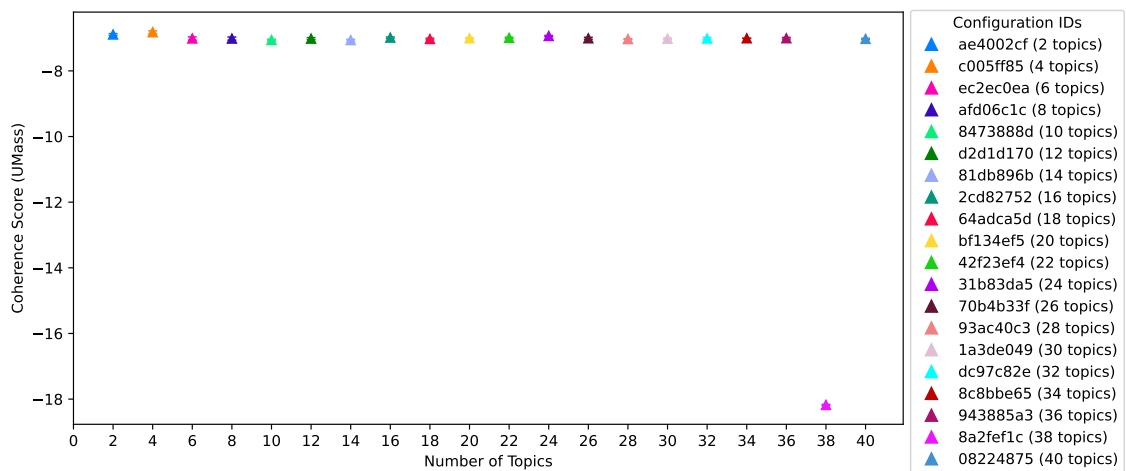


Figure 5.30: Average perplexity score by number of topics for the *baseline_bigram* pipeline (Lowest deviation)

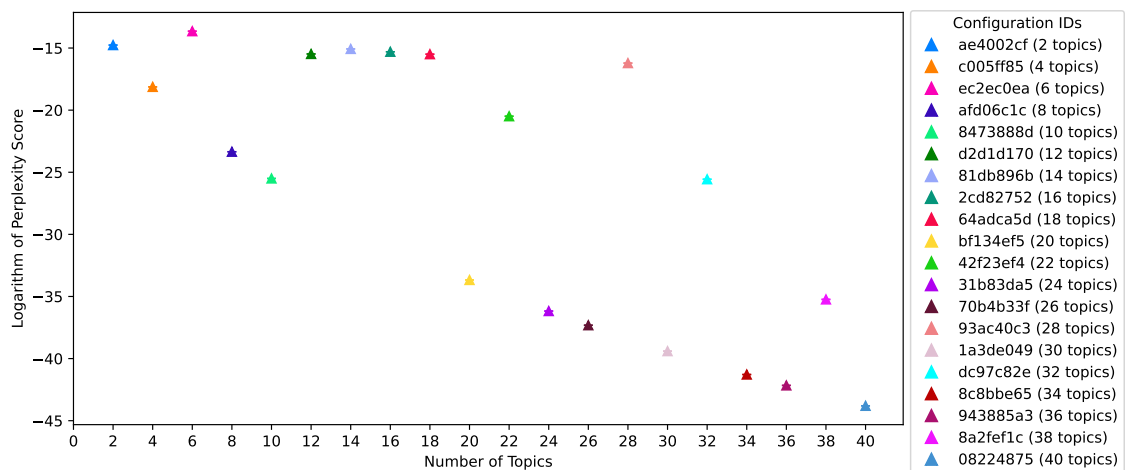


Figure 5.31: Average coherence score by number of topics for the *rem_words_bigram* pipeline (Highest value)

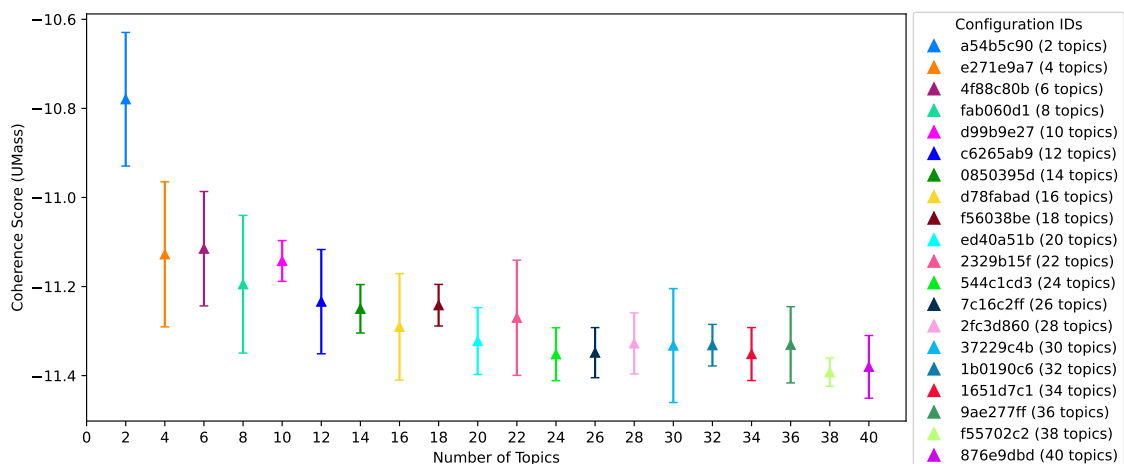


Figure 5.32: Average perplexity score by number of topics for the *rem_words_bigram* pipeline (Highest value)

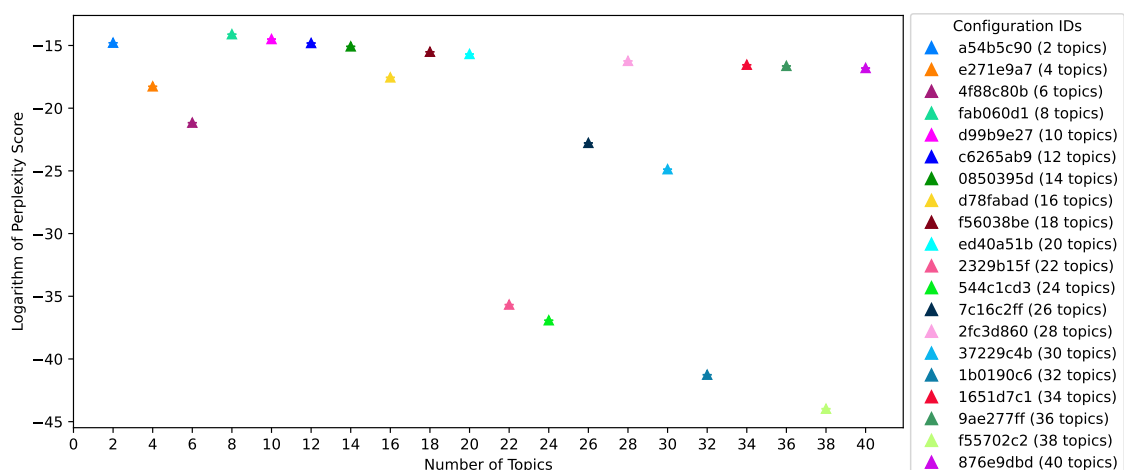


Figure 5.33: Average coherence score by number of topics for the *rem_words_bigram* pipeline (Lowest deviation)

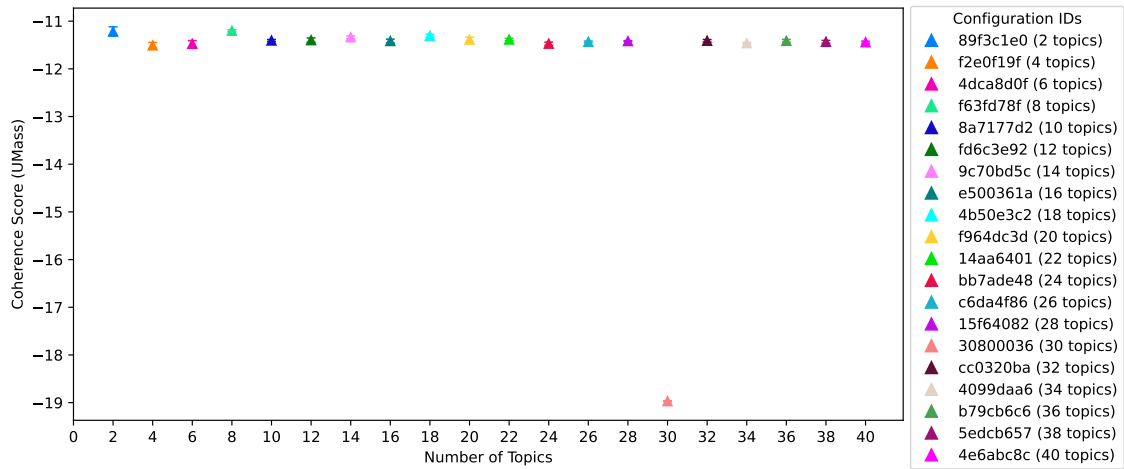
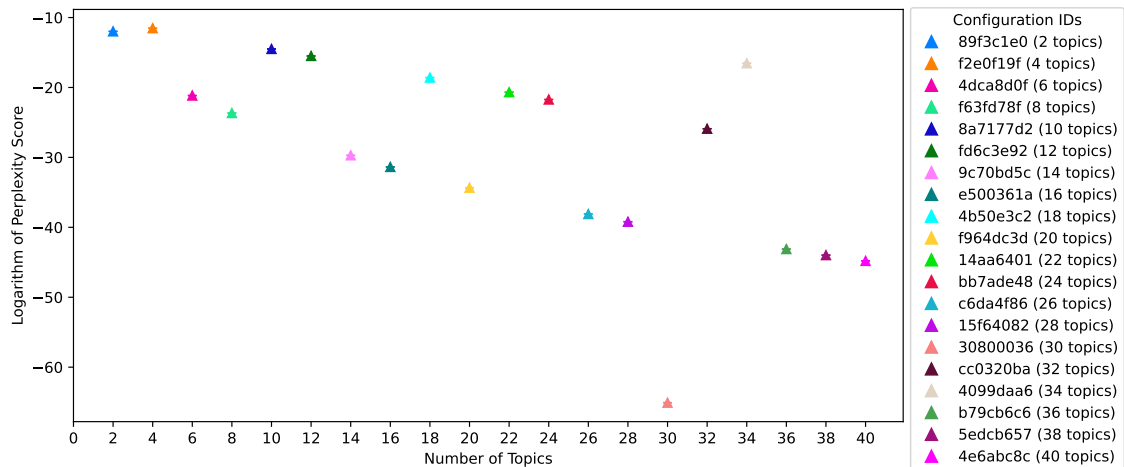


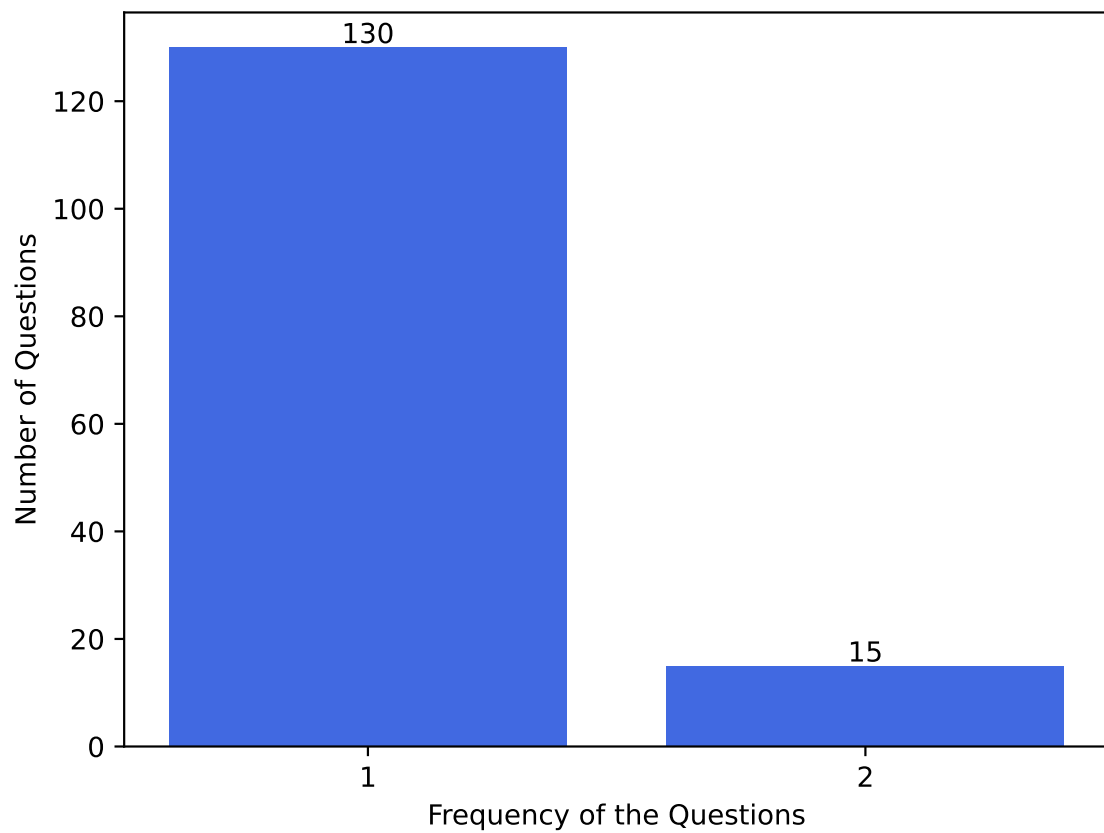
Figure 5.34: Average perplexity score by number of topics for the *rem_words_bigram* pipeline (Lowest deviation)



Considering the similar performance, we chose the *rem_words_baseline* pipeline to pick the best configuration and model for the same reasons as the ones enumerated in the previous section. We chose the configuration with eight topics, with the ID *8649104b*, because it has several topics that will allow us to identify at most eight challenges, which is a value in line with the challenges discovered in Chapter 3. The specific model has the ID *989261b0*.

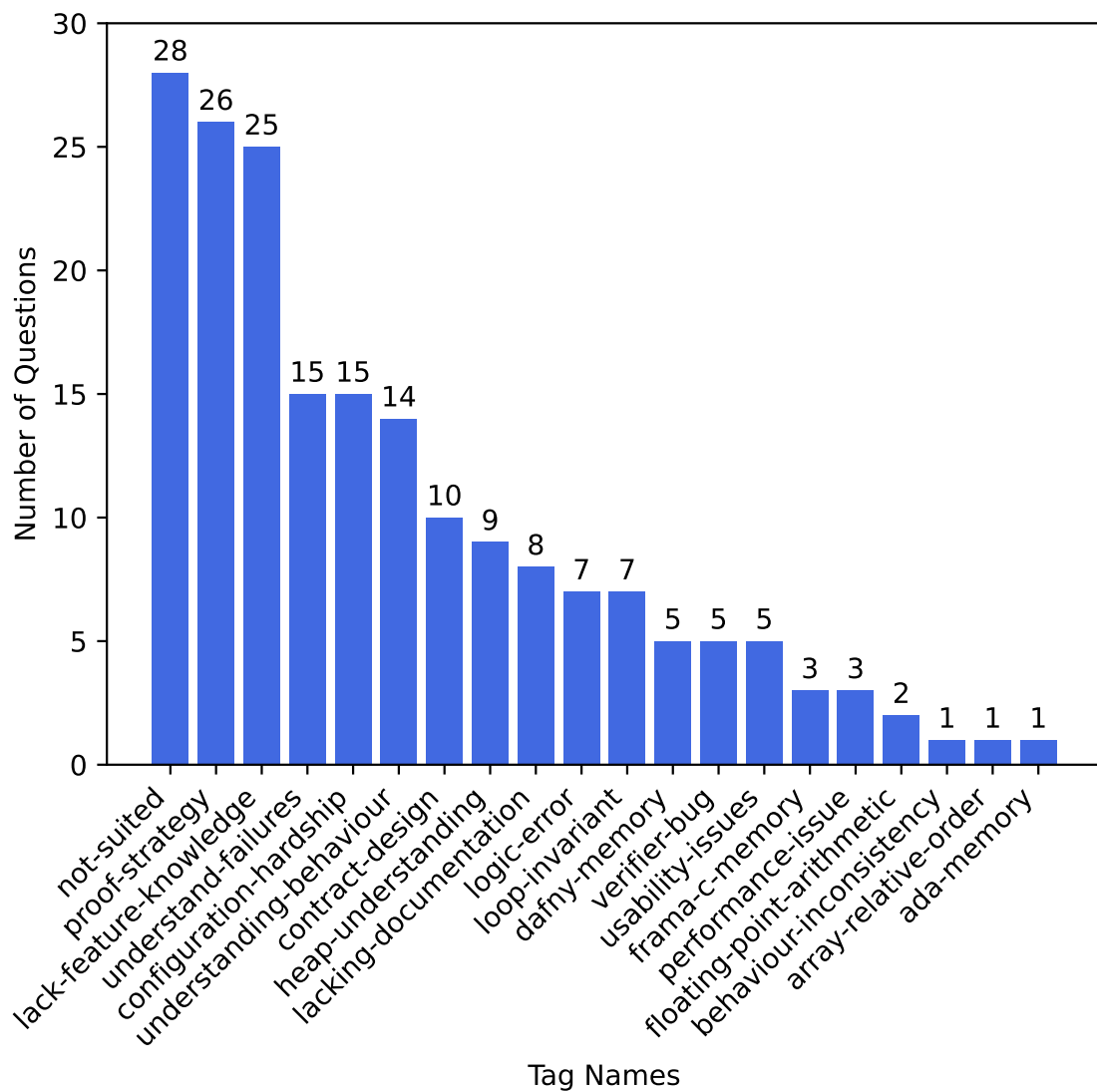
Moving on to the analysis of the topics, they are composed of 160 questions, from which 145 are unique, as can be observed in Figure 5.35. This indicates that this model is diverse and does not repeat questions.

Figure 5.35: Distribution of document frequencies



After performing open card sorting, we end up with the tag diversity shown in Figure 5.36. Despite being the most popular tag, the *not-suited* questions only comprise 19% of the overall questions against the 38% obtained in the previous section. As seen in appendix C, only in two topics the non-suited tag is the most frequent, again a contrast with our previous results.

Figure 5.36: Frequency of each tag



Based on our analysis, we decided that this model was suited for naming and challenge identification.

5.3 Identified tags

This section contains all the tags that were created during open card sorting in the sections 5.1 and 5.2. The tags will be presented in alphabetical order so the ordering does not match the order of discovery.

ada-memory

The questions annotated with this tag are related to the heap management system in Spark-Ada.

There are only two questions annotated with this tag. One question regards the comprehension of what happens if the *Old* attribute is used in a post-condition on a variable that might get deallocated inside the method. It is worth mentioning that this question is relevant because Ada does not contain a garbage collector, so the allocations in the heap must be managed by the programmer.

The remaining question regards the handling of an exception called *Storage_Error* that might be thrown by operations that handle the heap, like allocation. In this question the programmer wanted to handle the exception in Spark-Ada, but the accepted answer claims that such verification is not possible because the exception is generated outside the scope of the program.

array-relative-order

This tag is used to annotate questions where the programmers fail to prove properties about the relative order of elements in an array. It is a specific case of the tag *proof-strategy*.

There are only two questions annotated with this tag. The first is about a programmer that wanted to specify a function that returns all even elements of a numeric array and wanted to guarantee that the returned list contained the elements in the original order but could not write a specification. The second question is similar: the programmer is implementing the quick sort algorithm and fails to prove a lemma that ensures that applying a filtering predicate to two consecutive sub-lists and then concatenating the result, results in a list with the same elements in the same position as applying the filter to the whole list.

behaviour-inconsistency

This tag labels questions where the behavior of the verifier is inconsistent. Some examples include, the same code failing to be verified in the newest version of the verifier while being valid in older versions, or inconsistent behaviour between different verification backends where one verifies the code and the other does not. The last example is the verifier not understanding that a post-condition where eta-reduction was performed is equivalent to the original specification.

configuration-hardship

This tag contains all the questions where configuration challenges occurred. Some examples include incompatible versions of libraries being installed, challenges configuring the tools in a specific operating system, and challenges integrating the tools with IDEs. There are also challenges when integrating the different tools in the verification pipeline, this is prominent in verification-aware languages that allow for customization of backends like Frama-C with the WP plugin.

contract-design

This tag contains questions with challenges related to contract design. In the questions classified with this tag, the programmers cannot identify or express the necessary pre and post-conditions to verify the program they are developing.

dafny-memory

This tag contains questions about challenges in proving the properties of Dafny’s heap variables. Some common challenges include forgetting to express properties about the origin of objects. For instance, one programmer forgot to say that a new node inserted into a tree is *fresh*, and another one forgot to express that the object being modified in a loop did not change, using the *old* statement.

Other challenges include not being able to understand what variables need the *modifies* and *reads* clause, or what values are affected by these modifiers. This is especially common when dealing with properties of objects.

floating-point-arithmetic

This tag is used to annotate questions related to proofs about floating point arithmetic. Only two questions are annotated with this tag. On one question, the programmer was proving properties about salaries, and the verifier could not reason about the proof. The other question concerns a programmer who wrote the contract specification using real numbers but was using floats in the code written in C. The differences in the result of the operation due to rounding were causing Frama-C to deduce that the proprieties defined in the contract could not always be fulfilled.

frama-c-memory

This tag contains questions regarding Frama-C WP Plugin heap management. The WP Plugin defines multiple memory models that the programmer must chose based on the needs of the proof. [Baudin et al. \(2024\)](#) define memory model as *a set of datatypes, operations and properties that are used to abstract the values living inside the heap during the program execution*. In C, this is translated by the guarantees given by functions that manipulate the memory, for example *memcpy*, and the context where pointer casting is allowed. The questions for this tag are related to understanding what memory model beter suits the programmer use-case, and what constraints the current memory model imposes when doing memory manipulations. It also includes more general challenges like what content is actually allowed to change when marked by *loop assigns*.

heap-understanding

This tag is used to annotate questions regarding the management of heap-allocated objects. All questions annotated with *ada-memory*, *dafny-memory* and *frama-c-memory* are also annotated with this tag.

lack-feature-knowledge

This tag is used to annotate questions where the cause of the challenge is the lack of knowledge about the language and its features. The challenges in this category are diverse and depend on the language.

Some challenges include not knowing about language constructs that would make the proof possible, like *twostate* lemmas in Dafny.

There are other challenges related to understanding how the language behaves, like knowing that methods in Dafny are opaque, i.e., the verifier cannot reason about the properties of the code beyond the contract that the user specified.

lacking-documentation

This tag is used to classify questions where the challenge can be attributed to imprecise, outdated, or wrong documentation.

logic-error

This tag is used to classify questions where the programmer's challenges can be attributed to logic errors in the code.

Some examples include a programmer who defined a function to detect binary trees, but the function was wrong. Another example includes a programmer who was writing a function to detect if a given list was sorted in ascending or descending order. The programmer forgot to make the conditions disjoint, creating a conflict between the return value if all elements from the list were equal.

loop-invariant

This tag annotates all questions where the challenges derive from the programmer's inability to identify or express the loop invariants necessary to prove the code valid.

not-suited

Includes all questions not related to our research domain. Some questions include vague questions where the author's intention can not be understood, questions regarding business decisions, questions related to the underlying SAT solvers, etc.

performance-issue

This tag relates to questions where the verifier is slow to verify the program or can not complete the proof due to timeouts.

proof-strategy

This tag is related to questions where the programmer expresses challenges in designing or completing a proof to demonstrate to the verifier that the contract is respected by the created functions.

understand-failures

This tag is related to questions where the user designs a strategy and completes the proof, but the proof is insufficient to verify the program. Some examples are questions where the user cannot understand the counter-examples presented by the verification tool or the error messages.

understanding-behaviours

This tag is related to questions where the user fails to understand the behavior of the verifier. Contrasting with *understand-failures*, this tag can be associated with successful verification if the user fails to understand why the verifier worked.

usability-issues

Includes all challenges related to the usability of tools. The questions annotated with this tag present challenges tied to the tool. Some examples are challenges setting up compiler arguments in Frama-C and hard-to-read counterexamples in Dafny.

verifier-bug

This tag includes challenges related to identified verifier bugs.

5.4 Topic Labeling

This section presents each topic's name and the reasoning supporting the choice from the chosen model configuration with 8 topics, as previously indicated. The exact distribution of tags for each topic can be found in the appendix C.

The first topic we will describe is topic 0. Although the most common is not suited, we see a pattern with 5 questions being tagged with *heap-understanding*. Because of that, we decided to label the tag *Memory Model Understanding*.

Regarding topic 1, we observe an evident pattern related to the tags *lack-feature-knowledge* and *understand-behaviour*. The occurrences of *logic-error* and *loop-invariant* were observed along with the dominant tag, as well as the tag *understand-failures* and *lacking-documentation*. Moreover, two of the *understand-behaviour* occurrences are related to the same question with different StackOverflow IDs. Because of all the reasons stated above, we labeled the topic *Language Understanding*.

Unlike previous topics, for topic 2, we could not find any relation between the clustered questions, so we labeled the topic *Not Suited*.

Concerning topic 3, most of the questions are related to the tag *configuration-hardship*, so we labeled the topic *Environment Setup*.

In both topics 4 and 5, we could not find any relations between the top question labels, so we labeled both topics *Non Relevant*.

As for topics 6 and 7, both contain many questions regarding the label *proof-strategy* and some questions regarding *loop-invariant*, so we labeled the topic *Proof Elaboration*.

In conclusion, we identified 4 great topics of challenges that are enumerated in Section 5.5, being *Memory Model Understanding*, *Language Understanding*, *Environment Setup*, and *Proof Elaboration*.

5.5 Challenge Identification

In this section, we enumerate the challenges derived from the previous challenge groups.

5.5.1 Memory Model Understanding

The challenges from topic 0, *Memory Model Understanding* are language-specific because we observed that languages where the user manually manipulates the memory, like C, have different challenges than garbage-collected languages like Dafny. We could say that the challenges are paradigm-specific, but since we could only identify challenges in two languages, Frama-C annotated with ACSL and Dafny, we decided not to generalize our claims.

We identified three challenges among Dafny programmers, all related to the *modifies* clause. The *modifies* clause specifies the set of memory locations that a method, iterator, or loop body may modify (Leino et al. (2024)).

The first challenge identified in question 305 is related to the user applying a *modifies* clause to a given memory location, holding a reference to an object within the scope of a method but only changing the properties of the given object inside a loop. Dafny assumes that the loop might change the object to which the memory location is pointing, thus making verification based on the previously set object properties impossible. The programmer must create an invariant saying that the memory location always points to the same object.

The second challenge in questions 452 and 489 is related to programmers not understanding what variables are affected by the *modifies* clause.

In question 452, the author is trying to increase the size of an array by creating a new one copying the contents of the old array into the new and replacing the arrays. According to our tests, the author is missing the *modifies this* clause in the method that replaces the smallest array with the biggest.

In question 489, no code is provided, but considering the context given by the accepted answer, we can conclude that the author is trying to modify an array that is a property of a class. Instead of including *modifies this.rectangles* in the methods signature, the author is only including *modifies this*.

In both questions, we can see that the authors fail to acknowledge what is the right target of the *modifies* clause when the target property is a memory location to an object.

In question 451, we observed a similar challenge, but the clause the author is using is the *reads* clause. The *reads* clause marks all the heap memory locations the function is allowed to

read (Leino et al. (2024)). The author fails to write a clause that marks all graph nodes as being read by a given function.

The challenge associated with Frama-C WP Plugin, observed in question 854, is similar to the challenges about the *modifies* statement in Dafny. In this question, the developer fails to understand that the *loop assigns* expression stated that the pointer to the buffer *dst* was changing and not its contents.

5.5.2 Language Understanding

This section presents the challenges identified in topic 1 *Language Understanding*. Similar to the previous section, most of the challenges related to features are language-specific.

First, we will overview the challenges related to Dafny and then the ones related to Frama-C WP Plugin, which uses ACSL as an annotation language. Finally, we will analyze two questions that we consider to be language-agnostic.

In question 188, the author wanted to create a high-order function with certain pre-conditions for its arguments. This concept in Dafny is called partial function, and the author seems to lack knowledge about it.

In question 250, the author tries to understand the *fuel* attribute in Dafny using an example function. The *fuel* attribute is used to specify how many times a function definition should be unfolded when executing the verification (Leino et al. (2024)). The challenge is tied to the internal mechanisms that Dafny uses to verify recursive definitions.

In question 441, the author fails to understand the error message provided by the verifier about a type definition. Upon further inspection, we considered the message self-explanatory for developers who knew about subset types and witnesses. Witnesses are values that prove a certain sub-type is non-empty.

In question 444, the author fails to acknowledge what constructs, i.e., functions, methods, predicates, constructors, etc., are opaque. In Dafny, when a construct is opaque, the verifier does not introspect it to derive pre-conditions and post-conditions. The developer must define the contract; some examples are methods and constructors. On the other hand, predicates and functions are non-opaque, meaning that the verifier derives the contract. In this specific case, the developer forgot the constructor's post-condition.

Regarding the Frama-C questions, in question 938, the author is trying to use a feature defined in the ACSL Reference Manual but not implemented by the Frama-C WP Plugin.

In question 973, the author wants to prove referential transparency that, given the same arguments, the function always returns the same values using ACSL. The accepted answer claims that this is impossible in ACSL and points the author to a plugin called RPP that extends ACSL to provide this functionality.

In question 995, the author is trying to use a C11 specification feature not implemented by the Frama-C pre-processor at the time of writing.

In question 1053, the accepted answer claims that the author fails to acknowledge the need for *loop assigns* in the presented loops. The question has some *loop assigns* expressions and was edited 8 hours after creation, presumably to add these expressions.

Finally, we will analyze 3 questions we consider language agnostic.

In question 353, the author wants to prove a lemma in Dafny. The original code by the author provokes an internal error in Dafny 4.7.0. According to the accepted answer, an assertion was missing to make the verifier accept the proof. Because the need for the assertion is probably tied to the internal verification mechanism, we considered that the proof elicits a lack of language understanding, namely the underlying verification mechanisms.

In question 1111, the author is trying to understand the translation of mapping WhyML code into SMT Logic to understand how the proofs for ACSL annotated programs are provided. This question is not a challenge but introduces a need for understanding the underlying verification mechanisms.

The last question, 1003, contains a simple syntax error where the author does not use the correct syntax to write comments in ACSL.

5.5.3 Environment Setup

This section presents the challenges in topic 3 *Environment Setup*. In contrast to previous challenges, we will treat this section as language agnostic.

The first class of challenges is related to dependency installation. It includes challenges related to missing dependencies, incompatible dependency versions, and dependencies not being found. The questions that represent these challenges are 552, 746, 963, 1144.

The other class of challenges is related to unsupported use cases. This challenge is represented by question 16, in which the user wants to configure the version of function declarations used by the language server.

The last class of challenges is the missing installation instructions, represented by question 1327.

5.5.4 Proof Elaboration

This class of challenges is represented by both topics 6 and 7. Similar to *Environment Setup*, the challenges in this class are related to the programming paradigm, so the language is not important to analyze them.

The first challenge is related to creating a proof strategy. Many question authors could not even start the proof, not knowing what proof strategy to choose. This challenge was observed in questions 123, 221, 236, 243, 259, 346, 994, 1129, and 1359.

The second challenge is related to completing a proof. In contrast with the previous challenge, the authors of questions with these challenges could start to design a proof strategy that then proved impractical in verifying the properties of the code. This challenge was observed in questions 24, 135, and 138.

The third challenge is related to authors who chose the non-ideal strategy to prove the properties of the code; for example, choosing a proof by induction when a proof by contradiction would be easier. This challenge was observed in question 138.

The fourth challenge concerns identifying and expressing the necessary loop invariants to verify loops. This challenge was observed in questions 179, 302, 476, 1037, and 1374.

The fifth challenge is related to the fourth and concerns identifying and expressing a valid decrease clause that proves that loops or recursion functions terminate. This challenge was observed in question 30.

The sixth challenge concerns not being able to gather feedback, i.e., whether the contracts are respected due to the verifier timing out. This challenge was observed in questions 1037 and 1129.

5.6 Conclusion

This chapter explored the results using the metrics defined in Section 2.3.1. We chose the pipeline *rem_words_baseline* after filtering the questions considered unsuited after a first classification.

After choosing the model, we classified 145 questions, resulting in 20 categories. The meaning of each one is explored in Section 5.3.

Finally, in Section 5.5 we identify and detail 24 challenges that will be used as guidance to elaborate the question of the survey of Chapter 6.

Chapter 6

Validation

Recalling the research questions formulated at Chapter 1:

RQ1: What is the context of first contact with Verification-Aware languages?

RQ2: What challenges do Verification-Aware language practitioners face?

RQ3: What improvements can be made to the existing tools to overcome those challenges?

This chapter presents the methodology and results of a survey that was elaborated to validate the responses of RQ2 presented in Chapter 5 and presents a response to RQ1 and RQ3.

We define the methodology in Section 6.1. Next, in the Section 6.2, we describe the survey questions. In Section 6.3, we describe to what population we distributed the survey. Finally, in Section 6.4, we analyze the responses to each question.

6.1 Validation Methodology

As cited in the previous section, to validate the responses to RQ2 and answer the remaining research questions, we elaborated a survey. Surveys pose several advantages in contrast with other methodologies.

According to Müller et al. (2014), survey research might be the most appropriate methodology to gather the characteristics of users and feedback about their experiences. Surveys are also good at collecting data from geographically dispersed locations, which proved helpful in gathering feedback about the experience of using Verification-aware languages from professionals in the area.

6.2 Survey Design

The survey is composed of 10 sections. The name of the sections and the respective abbreviation, which will be used as a prefix for question abbreviated names, can be consulted in Table 6.1.

Table 6.1: Sections of the survey and respective abbreviated names

Name	Abbreviated Name
Consent	C
Developer Context	DC
Proof Elaboration	PE
Environment Setup	ES
Feature Knowledge	FK
Dafny Knowledge	DK
Frama-C Knowledge	WPK
Additional Challenges	AC
Overcoming Challenges	OC
Demographics	DM

6.2.1 Consent

The survey starts with Section *Consent*, which contains an introduction to the study explaining its purpose, procedures, participant requirements, risks, and how we use the information collected. This section includes the first two questions, C01 and C02, whose details can be consulted in tables 6.2 and 6.3, respectively.

Table 6.2: Details about question C01

Description	I have read and understand the information above.
Type	Single Choice
Required	Yes
Options	• Yes • No

Table 6.3: Details about question C02

Description	I want to participate in this research and continue with the survey.
Type	Single Choice
Required	Yes
Options	• Yes • No

After answering the questions about consent, if either question is responded to with *No*, we thank the respondent for their time and end the survey. If both questions are answered with *Yes*, we continue to the *Developer Context* Section.

6.2.2 Developer Context

The *Developer Context* Section is designed to answer question RQ1 and provide additional context about what languages the respondent is knowledgeable in. It starts by defining design-by-contract and verification-aware language, followed by an example of a method that computes the

Fibonacci's sequence in Dafny.

After the necessary contextualization and explanations, we start with question DC01, which asks about the respondents' years of experience in software development. Its details can be consulted in Table 6.4.

Table 6.4: Details about question DC01

Description	How many years of experience do you have in software development?
Type	Single Choice
Required	Yes
Options	• <1 year • 1-3 years • 4-6 years • 7-10 years • >10 years • I don't have any experience in software development

Like DC01, the next question, DC02, asks about the years of experience with verification-aware languages. The details of the question can be consulted at the Table 6.5.

Table 6.5: Details about question DC02

Description	How many years of experience do you have with verification-aware languages?
Type	Single Choice
Required	Yes
Options	• <1 year • 1-3 years • 4-6 years • 7-10 years • >10 years • I don't have any experience in software development

Following the questions about development experience, we ask about the verification-aware languages the respondent knows. The list is based on our research presented in Chapter 3. Apart from using this data to understand the distribution of languages of the collected answers, we used it to display language-specific sections. This question was named DC03 and its details can be consulted at the Table 6.6.

Table 6.6: Details about question DC03

Description	Please select the languages/technologies that you have programming experience with.	
Type	Multiple Choice	
Required	Yes	
Options	• Bandera	• Dafny
	• Escher	• Frama-C (WP Plugin)
	• Java PathFinder	• Mbeddr
	• Spec#	• Spark Ada
	• VeriFast	• VCC
	• Whiley	• Why3
	• Eiffel	• KeY
	• JML	• Other

The last two questions aim to discover the answer to RQ1 by asking how the respondents learned about verification-aware languages, DC04, and where they learned how to program with them, DC05. The details can be consulted in tables 6.7 and 6.8, respectively.

Table 6.7: Details about question DC04

Description	Where did you learn about verification-aware languages? Please select all that apply.
Type	Multiple Choice
Required	Yes
Options	• Online Discussions
	• Mandatory University Course
	• Optional University Course
	• Work Context
	• Other

Table 6.8: Details about question DC05

Description	Where did you learn how to program with verification-aware languages? Please select all that apply.
Type	Multiple Choice
Required	Yes
Options	• Online Discussions
	• Mandatory University Course
	• Optional University Course
	• Work Context
	• Other

6.2.3 Proof Elaboration

After tackling the research question RQ1, we start validating the answers to RQ2 found in Chapter 5. The first challenge category to be validated is *Proof Elaboration*. For this section, we elaborated

a single input table type of question. In the columns, we have a 5-level Likert scale. The rows are statements that translate each challenge identified in Section 5.5.4, and the order in which they appear is random. Since we only have one question for this section, it will be named just PE, and the details about it can be consulted in Table 6.9.

Table 6.9: Details about question PE

Description	The following statements regard proof elaboration. Please fill the matrix considering all your knowledge, regardless of the language.
Type	Input Table
Required	Yes
Columns	• Strongly Disagree • Somewhat Disagree • Neither Agree nor Disagree • Somewhat Agree • Strongly Agree • I'm not sure
Rows	• I struggle to design proof strategies that allow the validator to consider the code valid. • I have difficulties completing the strategy I designed to prove the properties of my code. • I choose not ideal proof strategies to prove the properties of my code. • I'm unable to identify the necessary loop invariants to prove the post-conditions. • I'm unable to express the necessary loop invariants to prove the post-conditions. • I'm unable to identify the necessary clause to prove loop/recursion termination (usually represented by the keyword decreases/terminates). • I'm unable to express the necessary clause to prove loop/recursion termination (usually represented by the keyword decreases/terminates).

6.2.4 Environment Setup

Like in the previous section, the challenges in this survey section resulted in one question with the challenges identified in Chapter 5, namely in Section 5.5.3. Similar to the previous section question, the order of the rows is randomized. The question is called ES, and its details are present in Table 6.10.

Table 6.10: Details about question ES

Description	Please think about your experience with verification-aware languages, namely the configuration of the development tools. How much do you agree with the following?
Type	Input Table
Required	Yes
Columns	• Strongly Disagree • Somewhat Disagree • Neither Agree nor Disagree • Somewhat Agree • Strongly Agree • I'm not sure
Rows	• I had incompatible dependencies installed, which caused a conflict with the verifier. • I couldn't find the correct version of the verification tool compatible with my preferred IDE. • The documentation of the verification tool did not consider my specific use case. For instance, change the language server's version without changing the language compiler's version.

6.2.5 Feature Knowledge

This survey section contains the subset of language agnostic challenges identified in Section 5. We decided not to include the challenge about writing comments from question 1327 because we considered it too general. This survey section contains only one question called FK, which is similar to the previous one. Its details are presented in the Table 6.11.

Table 6.11: Details about question FK

Description	Please recall your knowledge of verification-aware languages and their features and internal mechanisms.
Type	Input Table
Required	Yes
Columns	• Strongly Disagree • Somewhat Disagree • Neither Agree nor Disagree • Somewhat Agree • Strongly Agree • I'm not sure
Rows	• I understand how my code is translated to the underlying SMT code. • I struggle to translate the proofs I made in other support (e.g. paper) to the target verification-aware language.

6.2.6 Dafny Knowledge

This section contains the Dafny-specific challenges identified in Section 5.5.2 and is only displayed if, in question DC03, the respondent selected Dafny.

We start with a contextualization for the section, followed by a question about development experience, specifically with Dafny. This question, we call DK01, has its details described in the Table 6.12.

Table 6.12: Details about question DK01

Description	Please rate your expertise using Dafny on a scale of 0 to 5 with increasing level of skill. Consider your education, experience, publications, and familiarity with techniques and tools.
Type	Single Choice
Required	Yes
Options	• 0 - I have no expertise using Dafny • 1 - Novice (can complete basic proofs with guidance) • 2 • 3 - Intermediate (can complete basic proofs and intermediate proofs with guidance) • 4 • 5 - Expert (can create novel approaches to prove advanced algorithms)

After the question about Dafny's experience, we ask questions about the specific challenges identified in Chapter 5. From now on, every question comes in pairs. The first question in the pair asks about the respondent familiarity with a particular concept and the second asks about specific challenges related to that concept.

The first pair is about Dafny's memory model. The first question, DK02, whose description can be found in Table 6.13, asks about the familiarity with Dafny's memory model. Because

memory model is an expression not used by the Dafny documentation, we elicit the term with common keywords associated with it.

Table 6.13: Details about question DK02

Description	Are you familiar with Danfy’s memory model? A memory model describes the behavior of accesses to shared variables or heap objects. Its use is usually tied to the keywords “modifies” and “reads”.
Type	Single Choice
Required	Yes
Options	• Yes • No • I’m not sure

Like the previous questions about specific challenges, DK03, whose description is shown in Table 6.14, asks about the challenges identified in Chapter 5 related to Dafny’s memory model.

Table 6.14: Details about question DK03

Description	Please answer these questions based on your knowledge regarding Dafny’s memory model.
Type	Input Table
Required	Yes
Columns	• Strongly Disagree • Somewhat Disagree • Neither Agree nor Disagree • Somewhat Agree • Strongly agree • I’m not sure
Rows	• I forget to express that the value of a variable does not change (when adequate). • I don’t know what values the “modifies” clause affects. • I don’t know what values the “reads” clause affects.

The second pair is about opacity: question DK04, described in Table 6.15, asks about familiarity with the concept in question (opacity) and question DK05, described in Table 6.16, contains a question about the challenge.

Table 6.15: Details about question DK04

Description	Are you familiar with the concept of opacity?
Type	Single Choice
Required	Yes
Options	• Yes • No • I’m not sure

Table 6.16: Details about question DK05

Description	How confident are you that you understand what statements are opaque and non-opaque?
Type	Single Choice
Required	Yes
Options	• Not Confident at All • Slightly Confident • Somewhat Confident • Fairly Confident • Completely Confident • I'm not sure

The pair of questions DK06 and DK07, described in tables 6.17 and 6.18, respectively, are about the *fuel* attribute, related to function unfolding.

Table 6.17: Details about question DK06

Description	Are you familiar with the concept of fuel?
Type	Single Choice
Required	Yes
Options	• Yes • No • I'm not sure

Table 6.18: Details about question DK07

Description	How confident are you that you understand how the fuel attribute works?
Type	Single Choice
Required	Yes
Options	• Not Confident at All • Slightly Confident • Somewhat Confident • Fairly Confident • Completely Confident • I'm not sure

The pair of questions DK08 and DK09, described in tables 6.19 and 6.20, respectively, ask about the challenges of defining preconditions for the arguments of functions given as parameters. Function arguments with such requirements are known as partial functions.

Table 6.19: Details about question DK08

Description	Are you familiar with the concept of partial functions?
Type	Single Choice
Required	Yes
Options	• Yes • No • I'm not sure

Table 6.20: Details about question DK09

Description	How confident are you that you can use partial functions to reason about the fulfillment of high-order functions function argument requirements?
Type	Single Choice
Required	Yes
Options	• Not Confident at All • Slightly Confident • Somewhat Confident • Fairly Confident • Completely Confident • I'm not sure

The last pair of questions DK10 and DK11, described in tables 6.21 and 6.22, respectively, ask about the concept of witnesses, values provided to type definitions to prove the non-emptiness of such types.

Table 6.21: Details about question DK10

Description	Are you familiar with the concept of witnesses?
Type	Single Choice
Required	Yes
Options	• Yes • No • I'm not sure

Table 6.22: Details about question DK11

Description	How confident are you that you can prove the non-emptiness of type definitions by providing witnesses?
Type	Single Choice
Required	Yes
Options	• Not Confident at All • Slightly Confident • Somewhat Confident • Fairly Confident • Completely Confident • I'm not sure

6.2.7 Frama-C Knowledge

This section includes the challenges from Section 5.5.2 regarding Frama-C WP Plugin and, like the previous section, is only displayed if, in question DC03, the respondent selected Frama-C (WP Plugin).

Like in the *Dafny Knowledge* section, we start by asking about the language proficiency for Frama-C (question WPK01, details are in Table 6.23).

Table 6.23: Details about question WPK01

Description	Please rate your expertise using Frama-C with WP Plugin on a scale of 1 to 5 with increasing level of skill. Consider your education, experience, publications, and familiarity with techniques and tools.
Type	Single Choice
Required	Yes
Options	• 0 - I have no expertise using Dafny • 1 - Novice (can complete basic proofs with guidance) • 2 • 3 - Intermediate (can complete basic proofs and intermediate proofs with guidance) • 4 • 5 - Expert (can create novel approaches to prove advanced algorithms)

Following the question about the developer experience is a pair of questions similar to the ones in *Dafny Knowledge*, and is about WP Plugin memory models. Question WPK02, whose details are in Table 6.24, asks if the respondent is knowledgeable in WP memory models. Question WPK03, whose details are in Table 6.25, addresses the challenges identified in the section that was mentioned in the introduction of Chapter 5.

Table 6.24: Details about question WPK02

Description	Are you familiar with Frama-C WP Plugin memory models? A memory model describes the behavior of accesses to shared variables or heap objects.
Type	Single Choice
Required	Yes
Options	• Yes • No • I'm not sure

Table 6.25: Details about question WPK03

Description	Please fill out the matrix with your knowledge regarding Frama-C WP Plugin memory model.
Type	Input Matrix
Required	Yes
Columns	• Not Confident at All • Slightly Confident • Somewhat Confident • Fairly Confident • Completely Confident • I'm not sure
Rows	• How confident are you that you understand the difference between Frama-C's memory models? • How confident are you that you can choose the correct memory model to prove the properties of your code? • How confident are you that you are able to choose what variables you need to declare in "loop assigns" statements?

The last question of the section addresses the remaining challenges about general language features. Therefore, there is no question whether the respondent knows about a specific feature. This question was named WPK04 and its details are in Table 6.26.

Table 6.26: Details about question WPK04

Description	Please fill in the following matrix regarding your knowledge of Frama-C WP Plugin features.
Type	Input Matrix
Required	Yes
Columns	• Strongly Disagree • Somewhat Disagree • Neither Agree nor Disagree • Somewhat Agree • Strongly Agree • I'm not sure
Rows	• I find features of C specifications not implemented by Frama-C and of which I'm not aware. • I find features missing in Frama-C that other plugins have implemented.

6.2.8 Additional Challenges

This section contains a single open-ended question that must be filled with additional challenges that the respondent considers not addressed in the previous sections. This question was named AC and its details are in Table 6.27.

Table 6.27: Details about question AC

Description	What challenges, if any, have you encountered with verification-aware languages we haven't mentioned?
Type	Open-Ended
Required	Yes

6.2.9 Overcoming Challenges

This section aims to answer RQ3 and contains some solutions that we developed based on what was read during question tagging.

It starts with another open-ended question asking for suggestions to improve the adoption of verification-aware languages. This question was called OC01. Its details are in the Table 6.28.

Table 6.28: Details about question OC01

Description	What suggestions do you have, if any, to improve the adoption of verification-aware languages?
Type	Open-Ended
Required	Yes

The next question is an input table containing suggestions we developed based on observing problems during question analysis.

The first statement is about creating user-readable counter-examples that complement the error messages. From our question analysis, two questions mention Dafny’s counter-examples, question 199 and question 396. In both questions, counter-examples do not seem to help, and in the latter question, their usage is even discouraged by one of the answers. In general, we did not observe that counter-examples were useful for developers.

The second statement is about creating official tutorials about the verification of common developer exercises, like implementing a function to generate the n^{th} term of the Fibonacci sequence or calculating the greatest common divisor between two numbers. Considering the two most common tools in our dataset, Frama-C and Dafny, none contain guided tutorials on the official tool page.

The third statement is about creating a search feature in the documentation. Again, our dataset’s two most common tools do not contain documentation searching features. Regarding Frama-C, it does not contain documentation published on a website and is only available in PDF format. Dafny has online documentation, but there is no search feature, and the table of contents is on the top of the page and disappears while navigating. Other popular reliability-ensuring tools like Jest¹, JUnit², and Pytest³ possess a table of contents on the side of the page where documentation is located or a full-text search feature.

The fourth statement concerns the themes the documentation covers and the possibility of expanding documentation.

The fifth and last statement was included because verification-aware language contact often appears to be made through optional courses and not presented in mandatory courses like testing usually is.

The question containing all these statements is question OC02, whose details are in Table 6.29.

¹<https://jestjs.io/docs/getting-started> [last access 16th of July 2024]

²<https://junit.org/junit5/docs/current/user-guide> [last access 16th of July 2024]

³<https://docs.pytest.org/en/8.2.x/contents.html> [last access 16th of July 2024]

Table 6.29: Details about question OC02

Description	Please fill in the following matrix regarding your knowledge of Frama-C WP Plugin features.
Type	Input Matrix
Required	Yes
Columns	• Not important at All • Somewhat Important • Very Important • I'm not sure
Rows	• Create user-readable counterexamples that could elicit errors in the proof. • Improve official tutorials to include projects that programmers with previous knowledge know. • Improve documentation search mechanisms with a search function. • Expand the themes that are included in the documentation. • Include verification-aware language in undergraduate programs.

6.2.10 Demographics

In this last section, we ask questions about the respondents' demographics, including age, level of education, gender, and ethnic identification.

The first question asks about the age of the participants, dividing all ages into seven categories. It was named DM01. Its details are in the Table 6.30.

Table 6.30: Details about question DM01

Description	How old are you?
Type	Single Choice
Required	Yes
Options	• Under 18 • 18-24 years old • 25-34 years old • 35-44 years old • 45-54 years old • 55-64 years old • 65+ years old

The second question is about the gender of the respondent. It was named DM02. Its details are in the Table 6.31

Table 6.31: Details about question DM02

Description	How do you describe yourself?
Type	Single Choice
Required	Yes
Options	• Male • Female • Non-binary / third gender • Prefer not to say • Other

Following the respondent's characterization is the characterization of their level of education. We included the most common levels of education. The detailed list can be consulted in Table 6.32 that describes the question DM03.

Table 6.32: Details about question DM03

Description	What is the highest level of education you have completed?
Type	Single Choice
Required	Yes
Options	• Some high school or less • High school diploma or GED • Some college, but no degree • Associates or technical degree • Bachelor's degree • Graduate or professional degree (MA, MS, MBA, PhD, JD, MD, DDS etc.) • Prefer not to say

In question DM04, we ask about the ethnic identity of the respondent. We have a comprehensive list of races in the United States since the survey was also distributed to respondents there. Still, we also allow identifications relevant to other parts of the world. The question in detail can be consulted in Table 6.33.

Table 6.33: Details about question DM04

Description	Choose one or more races that you consider yourself to be
Type	Multiple Choice
Required	Yes
Options	• White or Caucasian • Black or African American • American Indian/Native American or Alaska Native • Asian • Native Hawaiian or Other Pacific Islander • Prefer not to say • Other

Question DM05 asks about previous work experience in the computing field. Its details are presented in Table 6.34.

Table 6.34: Details about question DM05

Description	Are you employed in a computing field (e.g., IT, software engineer, programmer)?
Type	Single Choice
Required	Yes
Options	• Yes • No • Black or African American • No, but I have been in the past • Other

In question DM06, we ask if the respondent has formal education in the computing field. The question details are presented in Table 6.35.

Table 6.35: Details about question DM06

Description	Do you have formal education in a computing field (e.g., degree in computer science or computer engineering)?
Type	Single Choice
Required	Yes
Options	• Yes • No • Other

The survey ends with an open-ended question asking for additional feedback from the respondent. This question was named DM07, and its details are in Table 6.36.

Table 6.36: Details about question DM07

Description	Do you have any other feedback for us? Feel free to share your thoughts about anything, from our survey about verification-aware languages and challenges to their use
Type	Open-Ended
Required	Yes

6.3 Survey Distribution

The survey was distributed to students of the 2023/2024 edition of Formal Methods for Critical Systems, an optional first-year Master’s level course from Faculdade de Engenharia da Universidade do Porto (M.EIC037), Portugal. In total the survey was distributed to 15 students.

Using Proxycurl⁴, but limited by the free version of the API, we gathered 28 profiles of LinkedIn users from the US with skills in both software verification and formal verification. From the 28 users, we could contact three profiles limited by the free trial of LinkedIn Premium.

We also contacted 16 industry professionals from our network of contacts and 28 Dafny contributors on GitHub, the ones that made their contact public. In addition, we contacted our professional network through the X social media platform⁵. We used snowball sampling by asking our contacts to also distribute the survey to relevant members of their professional and student networks.

Finally, we also distributed the survey through Frama-C’s mailing list.

6.4 Result Analysis

In this section, we analyze the results of our survey, which was answered by 31 participants from different backgrounds. We split the participants by experience: if they have more than three

⁴<https://nubela.co/proxycurl/> [last access 29th of July 2024]

⁵<https://x.com/>

years working with Verification-Aware Languages we consider them experienced; all others are considered beginners.

For participant characterization, we present tables showing the data acquired through the survey. In Section 6.4.1, we present the demographic analysis of the participants and, in Section 6.4.2, we present their background.

After the initial characterization, we analyzed each challenge in the group. First, we apply a *Mann-Whitney U* test to check if the data from the two groups, beginners and experienced, differ significantly and if the data needs to be handled separately. This test has the following assumptions (Nachar (2008)):

1. The two groups must be randomly drawn from the target population;
2. Each measurement must correspond to a different participant;
3. The data is ordinal or continuous.

If these assumptions are met, the test has the following null and alternate hypotheses, respectively:

H_0 : The two groups are homogeneous and have the same distribution

H_1 : The first group data distributions differs from the second

The specific implementation used comes from the *stats* library⁶, a library that provides multiple statistical tests implemented in R.

The test was chosen because our data complies with all the assumptions. Moreover, the test is non-parametric, which allows us to apply it to data that does not follow a Normal distribution, which is the case for some questions.

After applying the aforementioned test, we calculate the effect size of the test. The effect size is a quantitative measure that reflects the magnitude of a phenomenon. In this case, the similarity or difference between the distribution of the two compared groups. It helps us understand if the sample size was big enough to apply the Mann-Whitney U test.

We used an R function called *wilcox_effsize* from the library *rstatix*⁷ that computes the effect size for Mann-Whitney U tests. According to the documentation, effect sizes between 0.10 to 0.30 (excluding) are considered to be of small effect, values from 0.30 to 0.50 (excluding) are of moderate effect, and values above 0.50 are considered to be of large effect.

Finally, we analyze the frequency of each response to understand the distribution of values, i.e., if the participants encountered the challenge or not. A similar approach will be used to answer RQ3, considering if the participant found a specific improvement relevant to reduce the friction in adopting Verification-Aware languages.

⁶<https://www.rdocumentation.org/packages/stats/versions/3.6.2> [last access 12th of August 2024]

⁷<https://rpkg.sdatanovia.com/rstatix/> [last access 12th of August 2024]

6.4.1 Demographic Analysis

Considering the participants' demographics, we observe that most are males employed in the computing field with formal education. We also observe that most experienced participants have some graduate or professional degree, while most beginners only have a bachelor's degree. Table 6.37 contains the rest of the responses to demographic questions.

Table 6.37: Demographics of the survey participants

		Beginners	Experienced
Gender (DM02)	Male	14	11
	Female	4	1
	Third Gender/Other	-	-
	Prefer not to say	-	1
Age (DM01)	<18 years	0	-
	18-24 years	15	1
	25-34 years	3	2
	35-44 years	-	7
	45-54 years	-	1
	55-64 years	-	2
	>65 years	-	-
Level of Education (DM03)	Some college, but no degree	2	-
	Associates or technical degree	-	1
	Bachelor's degree	14	1
	Graduate or professional degree	2	11
	Other/Prefer not to say	-	-
Race (DM04)	Asian	1	1
	Black or African American	3	-
	White or Caucasian	12	8
	Prefer not to say	1	4
	African (Other)	1	-
Computing Field Employment (DM05)	Yes	7	13
	No	9	-
	No, but I have been in the past	2	-
Formal Education in the Field (DM06)	Yes	18	11
	No	-	2
	Other	-	-
TOTAL		18	13

6.4.2 Developer Background

This section explores the background of the participants and aims to characterize them. In this section, we also answer *RQ1: What was the context of the first contact with verification-aware languages?*. Table 6.38 summarizes the answers to the questions in this section.

We consider experienced programmers to have three years or more experience with Verification-Aware languages and have a balanced distribution of 18 beginners and 13 experienced programmers.

It is worth noting that many languages were grouped into “Other.” These languages were only observed once and include VCC, VDM-SL/VDM++, Eiffel, Java PathFinder, Creusot, Gillian, and Viper for experienced developers and VeRus for beginners.

Regarding the answer to RQ1, we observed a difference in the context of the first contact and learning of Verification-Aware languages. Beginner programmers had their first contact and learned Verification-Aware languages in an optional university course. On the other hand, most experienced developers had contact with Verification-Aware languages in a work context and learned them through a mix of self-taught and work contexts. 73% of the respondents who learned Verification-Aware languages at work also considered the learning process self-taught.

From our observation, we conclude that most participants learn about Verification-Aware languages as a choice either of education or career path. However, we can also observe that knowledge about Verification-Aware languages happens earlier and earlier in developers’ careers.

Table 6.38: Background and context of survey participants

		Beginners	Experienced
Software Development Experience (DC01)	<1 year	-	-
	1-3 years	4	-
	4-6 years	14	2
	7-10 years	-	1
	>10 years	-	10
VA Language Development Experience (DC02)	<1 year	18	-
	1-3 years	-	-
	4-6 years	-	7
	7-10 years	-	1
	>10 years	-	5
Known Languages (DC03)	Dafny	18	6
	Frama-C (WP Plugin)	-	7
	JML	1	3
	VeriFast	-	2
	Spark Ada	-	2
	Why3	-	8
	Other	1	12
VA Language First Contact Context (DC04)	Optional University Course	16	6
	Work Context	3	12
	Online Discussions	1	2
	Mandatory University Course	-	2
	Other	-	3
VA Language Learning Context (DC05)	Self-Taught	3	11
	Mandatory University Course	-	3
	Optional University Course	15	6
	Work Context	2	10
	Other	-	1
TOTAL		18	13

6.4.3 Proof Elaboration Challenges

In this section, we validate the challenges included in the Section 6.2.3, *Proof Elaboration (PE)*, related to creating contracts and elaborating proofs needed to prove their properties. We removed

Not Sure responses from the statistical test, assuming that they were due to the participant not understanding the statement, given that a *neutral* response was also provided.

We start by analyzing the responses to the challenge:

PE_1 : I struggle to design proof strategies that allow the validator to consider the code valid.

As stated previously, we start by applying the Mann-Whitney U test, which resulted in a p-value of 0.07473 that does not allow us to reject the null hypothesis. We obtained an effect size of 0.3238, which resulted in a moderate magnitude.

Since we can not reject the null hypothesis, we joined the data from both groups and presented it in Table 6.39. Most developers agreed with the statement, validating the challenge we found in Chapter 5.

Table 6.39: Frequency of each response to challenge PE_1

Agreed	Strongly Agree	6	20	64.5%
	Somewhat Agree	14		
Neutral			5	16.1%
Disagreed	Somewhat Disagree	6	6	19.4%
	Strongly Disagree	0		
Not Sure			0	0.0%
Total			31	100%

The specific response frequency for beginner and experienced developers can be found in the tables 6.40 and 6.41, respectively. Notably, the relative number of experienced developers who agreed with the statement is greater than among beginner developers.

Table 6.40: Frequency of each response to challenge PE_1 from beginner developers

Agreed	Strongly Agree	2	9	50.0%
	Somewhat Agree	7		
Neutral			5	27.8%
Disagreed	Somewhat Disagree	4	4	22.2%
	Strongly Disagree	0		
Not Sure			0	0.0%
Total			18	100%

Table 6.41: Frequency of each response to challenge PE_1 from experienced developers

Agreed	Strongly Agree	4	11	84.6%
	Somewhat Agree	7		
Neutral			0	0.0%
Disagreed	Somewhat Disagree	2	2	15.4%
	Strongly Disagree	0		
Not Sure			0	0.0%
Total			13	100%

In conclusion, we observed that most developers had some kind of challenge in choosing the proof strategy to verify the code, not knowing where to start.

The next challenge to be analyzed is related to completing a started proof. In contrast with the previous challenge, the developers started to develop a strategy, but it is insufficient to prove the validity of the contracts. It was formulated as the following sentence:

PE_2 : I have difficulties completing the strategy I designed to prove the properties of my code.

We applied the hypothesis test, which resulted in a p-value of 0.7764 with an effect size of 0.05583, of negligible magnitude, which again does not allow us to reject the null hypothesis and indicates that more participants are needed to understand if there is a difference between the responses of beginner and experienced developers.

In this challenge, the most common response is to agree with the statement. However, the participants who agreed were not the majority. Still, we considered the challenge valid because the number of agreements overwhelms the number of disagreements, as shown in Table 6.42.

Table 6.42: Frequency of each response to challenge PE_2

Agreed	Strongly Agree	4	15	48.4%
	Somewhat Agree	11		
Neutral			8	25.8%
Disagreed	Somewhat Disagree	7	7	22.6%
	Strongly Disagree	0		
Not Sure			1	3.2%
Total			31	100%

Tables 6.43 and 6.44 show the frequency of each response among beginner and experienced developers, respectively. The separate data analysis has not changed our conclusion, which confirms that the number of participants agreeing with the statement is greater than the number of disagreements in both groups.

Table 6.43: Frequency of each response to challenge PE_2 from beginner developers

Agreed	Strongly Agree	2	8	44.4%
	Somewhat Agree	6		
Neutral			5	27.8%
Disagreed	Somewhat Disagree	4	4	22.2%
	Strongly Disagree	0		
Not Sure			1	5.6%
Total			18	100%

Table 6.44: Frequency of each response to challenge PE_2 from experienced developers

Agreed	Strongly Agree	2	7	53.8%
	Somewhat Agree	5		
Neutral			3	23.1%
Disagreed	Somewhat Disagree	3	3	23.1%
	Strongly Disagree	0		
Not Sure			0	0.0%
Total			13	100%

In conclusion, we observed that less than the majority of developers had difficulties completing the proof strategy to verify the code. However, the number of developers who faced the challenge is not negligible, surpassing the number of those who did not in both groups, validating the challenge. The effect size suggests that a greater sample is needed to understand if there are differences in the responses of both groups.

The third challenge is related to developers who choose a viable proof strategy that is not the easiest way to prove the properties of the code. It was phrased as follows:

PE_3 : I choose not ideal proof strategies to prove the properties of my code

When subject to the chosen hypothesis test, we verify that we can not reject the hypothesis that the distribution of responses between beginner and experienced developers is the same, with a p-value of 0.5822. This test had a small magnitude of 0.1045, which suggests that more observations are necessary to increase the test's magnitude.

Again, like in the previous challenge, the number of participants that agreed with the statement is greater than the number that disagreed but is not the majority of the participants if neutral responses are considered. Table 6.45 contains the frequency of each response.

Table 6.45: Frequency of each response to challenge PE_3

Agreed	Strongly Agree	3	15	48.4%
	Somewhat Agree	12		
Neutral			8	25.8%
Disagreed	Somewhat Disagree	7	7	22.6%
	Strongly Disagree	0		
Not Sure			1	3.2%
Total			31	100%

Similar to what was observed in the previous challenge, the number of participants agreeing with the statement is greater in the group of experienced developers than in the group of beginners. The frequency of each response for beginner and experienced groups is presented in tables 6.46 and 6.47, respectively.

Table 6.46: Frequency of each response to challenge PE_3 from beginner developers

Agreed	Strongly Agree	2	7	38.9%
	Somewhat Agree	5		
Neutral			6	33.3%
Disagreed	Somewhat Disagree	4	4	22.2%
	Strongly Disagree	0		
Not Sure			1	5.6%
Total			18	100%

Table 6.47: Frequency of each response to challenge PE_3 from experienced developers

Agreed	Strongly Agree	1	8	61.5%
	Somewhat Agree	7		
Neutral			2	15.4%
Disagreed	Somewhat Disagree	3	3	23.1%
	Strongly Disagree	0		
Not Sure			0	0.0%
Total			13	100%

In conclusion, the most common response is agreeing with the statement in both developer groups. However, among beginners, they are not the majority. Still, the challenge is validated since the most common response is to agree with the statement. The effect size suggests that a greater sample is needed to understand if there are differences in the responses of both groups.

The next challenges are related and correspond to identifying and expressing loop invariants. The statements that resulted from the challenge are respectively:

PE_4 : I'm unable to identify the necessary loop invariants to prove the post-conditions

PE_5 : I'm unable to express the necessary loop invariants to prove the post-conditions

We are not able to reject the hypothesis that the distribution of responses among the two groups is the same for both challenges, with p-values of 0.2552 and 0.1935, for PE_4 and PE_5 , respectively. However, the effect size suggests that more observations are required to apply the test, with both tests having a small effect of 0.2081 and 0.2373, respectively.

A preliminary analysis of the frequency responses of both beginner and experienced developers, present in tables 6.48 and 6.49, seems to indicate that neither has difficulties identifying and expressing loop invariants.

Table 6.48: Frequency of each response to challenge PE₄

Agreed	Strongly Agree	5	9	29.0%
	Somewhat Agree	4		
Neutral			7	22.6%
Disagreed	Somewhat Disagree	11	15	48.4%
	Strongly Disagree	4		
Not Sure			0	0.0%
Total			31	100%

Table 6.49: Frequency of each response to challenge PE₅

Agreed	Strongly Agree	3	9	29.0%
	Somewhat Agree	6		
Neutral			6	19.4%
Disagreed	Somewhat Disagree	13	16	51.6%
	Strongly Disagree	3		
Not Sure			0	0.0%
Total			31	100%

However, when taking a closer look at the responses of beginners, we observe that the results are inconclusive and require further data to understand if beginners face either challenges (Tables 6.50 and 6.51).

Table 6.50: Frequency of each response to challenge PE₄ from beginner developers

Agreed	Strongly Agree	3	7	38.9%
	Somewhat Agree	4		
Neutral			4	22.2%
Disagreed	Somewhat Disagree	5	7	38.9%
	Strongly Disagree	2		
Not Sure			0	0.0%
Total			18	100%

Table 6.51: Frequency of each response to challenge PE₅ from beginner developers

Agreed	Strongly Agree	2	7	38.9%
	Somewhat Agree	5		
Neutral			4	22.2%
Disagreed	Somewhat Disagree	5	7	38.9%
	Strongly Disagree	2		
Not Sure			0	0.0%
Total			18	100%

In contrast with the results of beginners, most experienced developers do not face challenges when either identifying or expressing loop invariants, as shown in tables 6.52 and 6.53.

Table 6.52: Frequency of each response to challenge PE₄ from experienced developers

Agreed	Strongly Agree	2	2	15.4%
	Somewhat Agree	0		
Neutral			3	23.1%
Disagreed	Somewhat Disagree	6	8	61.5%
	Strongly Disagree	2		
Not Sure			0	0.0%
Total			13	100%

Table 6.53: Frequency of each response to challenge PE₅ from experienced developers

Agreed	Strongly Agree	1	2	15.4%
	Somewhat Agree	1		
Neutral			2	15.4%
Disagreed	Somewhat Disagree	8	9	69.2%
	Strongly Disagree	1		
Not Sure			0	0.0%
Total			13	100%

In conclusion, further data needs to be gathered to understand if there is a difference in the responses of beginners and experienced developers. The data we gathered is inconclusive when considering beginners but seems to indicate that more experienced developers do not face challenges when identifying and expressing loop invariants, suggesting that mitigation measures should be applied when teaching verification.

The last two challenges are also tied and are similar to the previous question. They regard the identification and expression of *decreases* clause, necessary to prove loop and recursion terminations. The challenges were translated into the following statements:

PE₆: I'm unable to identify the necessary clause to prove loop/recursion termination (usually represented by the keyword *decreases*/terminates)

PE₇: I'm unable to express the necessary clause to prove loop/recursion termination (usually represented by the keyword *decreases*/terminates)

Regarding the first statement, the chosen hypothesis testing resulted in a p-value of 0.06562 with a moderate magnitude of 0.3401, meaning that we are not able to reject that the distribution of responses is the same.

When considering the joined data of beginners and experienced developers (Table 6.54) we can conclude that the participants do not consider identifying decrease clauses as a challenge.

Table 6.54: Frequency of each response to challenge PE₆

Agreed	Strongly Agree	3	9	29.0%
	Somewhat Agree	6		
Neutral			5	16.1%
Disagreed	Somewhat Disagree	11	16	51.6%
	Strongly Disagree	5		
Not Sure			1	3.2%
Total			31	100%

However, when taking a closer look at the responses of beginner developers, we observe that the results are inconclusive, with just a difference of one response between agreements and disagreements (Table 6.55).

Table 6.55: Frequency of each response to challenge PE₆ from beginner developers

Agreed	Strongly Agree	2	7	38.9%
	Somewhat Agree	5		
Neutral			2	11.1%
Disagreed	Somewhat Disagree	8	8	44.4%
	Strongly Disagree	0		
Not Sure			1	5.6%
Total			18	100%

Experienced developers seem to not consider identifying *decreases* clauses as a challenge with the majority of participants disagreeing with the statement (Table 6.56).

Table 6.56: Frequency of each response to challenge PE₆ from experienced developers

Agreed	Strongly Agree	1	2	15.4%
	Somewhat Agree	1		
Neutral			3	23.1%
Disagreed	Somewhat Disagree	3	8	61.5%
	Strongly Disagree	5		
Not Sure			0	0.0%
Total			13	100%

In conclusion, there seems to be a difference in the distribution of responses between beginners and experienced developers, with most experienced developers disagreeing with the statement. In contrast, only 44% of the beginners disagreed with it, indicating that beginners struggle more to identify *decreases* clause. However, further investigation is needed to support this claim since, according to the results of the Mann-Whitney U test, we can not reject the null hypothesis.

When considering the expression of *decreases* clauses, the test result yields a p-value of 0.02666 with an effect size of 0.4018 of moderate magnitude.

Considering the total number of responses, most developers disagree with the statement, indicating that the developers have no difficulties in this regard when identified (Table 6.57).

Table 6.57: Frequency of each response to challenge PE₇

Agreed	Strongly Agree	1	7	22.6%
	Somewhat Agree	6		
Neutral			4	12.9%
Disagreed	Somewhat Disagree	12	20	64.5%
	Strongly Disagree	8		
Not Sure			0	0.0%
Total			31	100%

As expected, there is a difference in the distribution of responses between the two groups. While only 50% of the beginners disagree with the statement, with 38% facing some kind of challenge in this regard (Table 6.58), none of the experienced developers faced any challenges (6.59).

Table 6.58: Frequency of each response to challenge PE₇ from beginner developers

Agreed	Strongly Agree	1	7	38.9%
	Somewhat Agree	6		
Neutral			2	11.1%
Disagreed	Somewhat Disagree	6	9	50.0%
	Strongly Disagree	3		
Not Sure			0	0.0%
Total			18	100%

Table 6.59: Frequency of each response to challenge PE₇ from experienced developers

Agreed	Strongly Agree	0	0	0.0%
	Somewhat Agree	0		
Neutral			2	15.4%
Disagreed	Somewhat Disagree	6	11	84.6%
	Strongly Disagree	5		
Not Sure			0	0.0%
Total			13	100%

In conclusion, since we discarded the hypothesis that the distribution of responses is equal and none of the experienced developers faced this challenge, this seems to indicate that this challenge is overcome with experience, indicating that beginners struggle more with the expression of *decreases* clauses.

6.4.4 Environment Setup Challenges

In this section, we validate the challenges included in the Section 6.2.4, *Environment Setup (ES)*, related to the setup of verification tools. Similarly to what was done in the previous section, we removed *Not Sure* responses from the statistical test, assuming that they were due to the participant not understanding the statement, given that a *neutral* response was also provided.

The first challenge we consider regards possible dependency conflicts when installing verification tools. It resulted in the following statement:

ES_1 : I had incompatible dependencies installed, which caused a conflict with the verifier

Applying the Mann-Whitney U test to the responses of the participants results in a p-value of 0.2571. However, the effect size of 0.2074, a small magnitude, suggests that more data is required to trust the results of the hypothesis testing.

When considering the joined data of beginner and experienced developers, we verify that most developers disagree with the statement (Table 6.60).

Table 6.60: Frequency of each response to challenge ES_1

Agreed	Strongly Agree	3	7	22.6%
	Somewhat Agree	4		
Neutral			3	9.7%
Disagreed	Somewhat Disagree	6	21	67.7%
	Strongly Disagree	15		
Not Sure			0	0.0%
Total			31	100%

The same phenomenon is observed in both the responses of beginner (Table 6.61) and experienced developers (Table 6.62).

However, the percentage of experienced developers who agreed with the statement is greater than the percentage of beginner developers. This is to be expected, since the longer and more diverse the tools developers use, the more likely you are to find incompatibilities between dependencies. Moreover, all the experienced developers that agreed with the statement had experience in multiple Verification-Aware languages.

Table 6.61: Frequency of each response to challenge ES_1 from beginner developers

Agreed	Strongly Agree	1	2	11.1%
	Somewhat Agree	1		
Neutral			3	16.7%
Disagreed	Somewhat Disagree	3	13	72.2%
	Strongly Disagree	10		
Not Sure			0	0.0%
Total			18	100%

Table 6.62: Frequency of each response to challenge ES_1 from experienced developers

Agreed	Strongly Agree	2	5	38.5%
	Somewhat Agree	3		
Neutral			0	0.0%
Disagreed	Somewhat Disagree	3	8	61.5%
	Strongly Disagree	5		
Not Sure			0	0.0%
Total			13	100%

In conclusion, our sample suggests that most developers did not encounter incompatible dependencies when installing verification tools. However, further investigation is needed to verify how the number of Verification-Aware languages a participant has experience with and how many are installed simultaneously in the participant's machine affects the likelihood of encountering this challenge.

The next challenge in this group is related to the compatibility between IDEs and verification tools. It was expressed as the following statement:

ES_2 : I couldn't find the correct version of the verification tool compatible with my preferred IDE

The application of the hypothesis testing results in a p-value of 0.4330 and a small effect size of 0.1447. This suggests that the distribution of responses between both groups is the same but more data is required to trust the results of the test.

Overall, most participants reject the statement indicating that the prevalence of this challenge is negligible (Table 6.63).

Table 6.63: Frequency of each response to challenge ES_2

Agreed	Strongly Agree	2	4	12.9%
	Somewhat Agree	2		
Neutral			3	9.7%
Disagreed	Somewhat Disagree	8	24	77.4%
	Strongly Disagree	16		
Not Sure			0	0.0%
Total			31	100%

The distribution of responses between both groups seems to be the same indicating that experience does not affect the prevalence of this challenge. The complete distribution of responses for beginners and experienced developers can be consulted at the tables 6.64 and 6.65, respectively.

Table 6.64: Frequency of each response to challenge ES₂ from beginner developers

Agreed	Strongly Agree	0	2	11.1%
	Somewhat Agree	2		
Neutral			3	16.7%
Disagreed	Somewhat Disagree	2	13	72.2%
	Strongly Disagree	11		
Not Sure			0	0.0%
Total			18	100%

Table 6.65: Frequency of each response to challenge ES₂ from experienced developers

Agreed	Strongly Agree	2	2	15.4%
	Somewhat Agree	0		
Neutral			0	0.0%
Disagreed	Somewhat Disagree	6	11	84.6%
	Strongly Disagree	5		
Not Sure			0	0.0%
Total			13	100%

In conclusion, we observed that most inquired developers did not encounter this challenge, and, contrary to the previous one the response distribution seems homogeneous between both groups.

The last challenge is related to the use cases of verifiers and the configurations that can be changed. This challenge was translated to the following statement:

ES₃: The documentation of the verification tool did not consider my specific use case. For instance, change the language server's version without changing the language compiler's version.

The application of the test resulted in a p-value of 0.1272 with a moderate effect size of 0.3045. This seems like a good indication that the distribution of responses is the same between participant groups.

When considering the overall responses of the participants, most disagreed with the statement. (Table 6.66). However, we observed a difference in the distribution of responses.

Table 6.66: Frequency of each response to challenge ES₃

Agreed	Strongly Agree	4	5	16.1%
	Somewhat Agree	1		
Neutral			5	16.1%
Disagreed	Somewhat Disagree	5	16	51.6%
	Strongly Disagree	11		
Not Sure			5	16.1%
Total			31	100%

While none of the beginner developers agreed with the statement (Table 6.67), half of the experienced participants that had an opinion formed about the statement agreed with it (Table 6.68).

Table 6.67: Frequency of each response to challenge ES₃ from beginner developers

Agreed	Strongly Agree	0	0	0.0%
	Somewhat Agree	0		
Neutral			4	22.2%
Disagreed	Somewhat Disagree	4	11	61.1%
	Strongly Disagree	7		
Not Sure			3	16.7%
Total			18	100%

Table 6.68: Frequency of each response to challenge ES₃ from experienced developers

Agreed	Strongly Agree	4	5	38.5%
	Somewhat Agree	1		
Neutral			1	7.7%
Disagreed	Somewhat Disagree	1	5	38.5%
	Strongly Disagree	4		
Not Sure			2	15.4%
Total			13	100%

In conclusion, while the statistical test does not allow us to reject the hypothesis that the distribution of responses between the group of developers is equal, we observed a difference in the responses. It shows that half of the experienced developers with an opinion formed about the statement, with possibly more complex workloads, find the tools lacking to support their use cases. Further data is necessary to understand how this challenge's prevalence evolves with the experience.

6.4.5 Feature Knowledge Challenges

In this section, we validate the challenges included in the Section 6.2.5, *Feature Knowledge (FK)*, related the knowledge about the underlying mechanisms of verification tools. Similarly to what was done in the previous section, we removed *Not Sure* responses from the statistical test, assuming that they were due to the participant not understanding the statement, given that a *neutral* response was also provided.

The first challenge regards the underlying translation of high-level code to the underlying SMT code. Understanding the translation that happens to allow verification is important to comprehend deductions the verifiers make. This resulted in the following statement:

FK_1 : I understand how my code is translated to the underlying SMT code

The application of the Mann-Whitney U test to the responses of the survey results in a p-value of 0.2953 and a small effect size of 0.1951. These results do not allow us to reject the hypothesis that the distribution of responses is the same among beginner and experienced developers. However, the effect size suggests that more responses are necessary to apply the test.

Most developers disagree with the statement, suggesting that most do not understand how the high-level verification code is translated to the underlying SMT code (Table 6.69).

Table 6.69: Frequency of each response to challenge FK₁

Agreed	Strongly Agree	3	13	41.9%
	Somewhat Agree	10		
Neutral			2	6.5%
Disagreed	Somewhat Disagree	10	15	48.4%
	Strongly Disagree	5		
Not Sure			1	3.2%
Total			31	100%

The distribution of responses among the categories of developers seems to differ, with the majority of beginner developers claiming not to understand how this translation is done (Table 6.70).

Table 6.70: Frequency of each response to challenge FK₁ from beginner developers

Agreed	Strongly Agree	1	6	33.3%
	Somewhat Agree	5		
Neutral			2	11.1%
Disagreed	Somewhat Disagree	5	9	50.0%
	Strongly Disagree	4		
Not Sure			1	5.6%
Total			18	100%

However, when looking at experienced developers displayed in Table 6.71, we verify that most developers agree with the statement, claiming they understand how the translation is done.

Table 6.71: Frequency of each response to challenge FK₁ from experienced developers

Agreed	Strongly Agree	2	7	53.8%
	Somewhat Agree	5		
Neutral			0	0.0%
Disagreed	Somewhat Disagree	5	6	46.2%
	Strongly Disagree	1		
Not Sure			0	0.0%
Total			13	100%

In conclusion, the understatement of the underlying verification mechanisms seems to be more prevalent among experienced developers, validating the challenge among beginner developers.

Further investigation is necessary to understand if this lack of knowledge impacted beginners' usability of verification tools.

The second challenge regards the translation of proofs from other supports, such as paper, into the verification language. It was translated as the following statement:

FK₂: I struggle to translate the proofs I made in other support (e.g. paper) to the target verification-aware language

The responses to this question resulted in a p-value of 0.2286 and a small effect size of 0.2280, indicating that the distribution of responses is the same, although more observations are necessary to increase the effect size.

The results in Table 6.72 indicate that most developers have no difficulties translating proofs to the desired verification language.

Table 6.72: Frequency of each response to challenge *FK₂*

Agreed	Strongly Agree	2	9	29.0%
	Somewhat Agree	7		
Neutral			4	12.9%
Disagreed	Somewhat Disagree	14	16	51.6%
	Strongly Disagree	2		
Not Sure			2	6.5%
Total			31	100%

The distribution of responses, displayed in Table 6.73, seems to differ among the group of developers in our sample, with most beginner developers disagreeing with the challenge. However, the results of the experienced developers, displayed in Table 6.74, contain relatively more developers agreeing with the statement.

Table 6.73: Frequency of each response to challenge *FK₂* from beginner developers

Agreed	Strongly Agree	1	4	22.2%
	Somewhat Agree	3		
Neutral			3	16.7%
Disagreed	Somewhat Disagree	9	11	61.1%
	Strongly Disagree	2		
Not Sure			0	0.0%
Total			18	100%

Table 6.74: Frequency of each response to challenge FK₂ from experienced developers

Agreed	Strongly Agree	1	5	38.5%
	Somewhat Agree	4		
Neutral			1	7.7%
Disagreed	Somewhat Disagree	5	5	38.5%
	Strongly Disagree	0		
Not Sure			2	15.4%
Total			13	100%

In conclusion, we observed a difference in the distribution of responses in our sample, with the challenge being more prevalent among experienced developers. Further investigation is needed to understand if the difference exists and why.

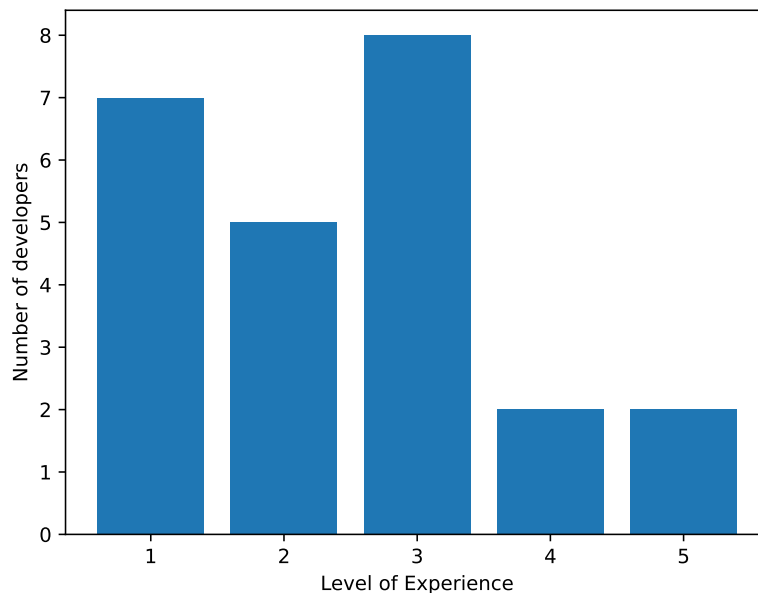
6.4.6 Dafny Knowledge

In this section, we analyze the challenges specific to Dafny. Since these are language-specific questions, we wanted to understand how knowledgeable the participants are about Dafny. We refer to Table 6.12 for the levels' specification and meaning of each experience level.

As shown in Figure 6.1, most participants are at the intermediate level, requiring guidance to complete intermediate proofs. Considering that experienced developers can complete intermediate proofs without guidance, i.e., have a level of experience above 4, we have insufficient data to split participants by Dafny experience since we would have two unbalanced groups, one with 20 participants and the other with just 4.

In conclusion, we will keep using the overall years of experience since it provides a better distribution of participants.

Figure 6.1: Distribution of responses to question DK01



The challenges in this group are split by category. At the start of each category, the participant is asked if they know a specific aspect of Dafny. If they answer with “No,” the question is kept hidden; other responses, i.e., “I’m not sure” and “Yes,” show the question to the participant.

The first group of challenges regards Dafny’s memory model. From the 24 participants who claimed to know Dafny, 20 knew about memory models, and 4 did not, totaling 20 responses to the challenges presented in question DK03, described in Table 6.14.

The first challenge regards the expression of the immutability of variables. For example, in loops that mutate arrays, an invariant is often needed to guarantee that the part of the array that is not yet touched was not mutated. This challenge was translated as follows:

DK_{3OP1} : I forget to express that the value of a variable does not change (when adequate)

We obtained a p-value of 0.8275 with a negligible effect size of 0.0585, which means that although we can not reject the null hypothesis that the distribution of responses between the groups is the same, the effect size of the test suggests that more data is needed to apply the test.

When observing the participants’ responses, we see that most disagreed with the statement. However, there is a non-negligible percentage of participants that agreed with it, validating the challenge (Table 6.75)

Table 6.75: Frequency of each response to challenge DK_{3OP1}

Agreed	Strongly Agree	0	8	40.0%
	Somewhat Agree	8		
Neutral			1	5.0%
Disagreed	Somewhat Disagree	5	11	55.0%
	Strongly Disagree	6		
Not Sure			0	0.0%
Total			20	100%

When considering the data aggregated by level of experience, the difference between the number of responses of beginner (Table 6.76) and experienced developers (Table 6.77) do not allow for a conclusion in the evolution of the challenge with the level of experience.

Table 6.76: Frequency of each response to challenge DK_{3OP1} from beginner developers

Agreed	Strongly Agree	0	5	35.7%
	Somewhat Agree	5		
Neutral			1	7.1%
Disagreed	Somewhat Disagree	5	8	57.1%
	Strongly Disagree	3		
Not Sure			0	0.0%
Total			14	100%

Table 6.77: Frequency of each response to challenge DK_{3OP1} from experienced developers

Agreed	Strongly Agree	0	3	50.0%
	Somewhat Agree	3		
Neutral			0	0.0%
Disagreed	Somewhat Disagree	0	3	50.0%
	Strongly Disagree	3		
Not Sure			0	0.0%
Total			6	100%

In conclusion, while further data is needed to understand the distribution of this challenge among beginner and experienced developers, 40% of the developers agreed with the statement showing that it relevant for a non-negligible number of developers in our sample.

The second challenge in this group translates the challenge of understanding the target of the *modifies* clause in Dafny. It was translated as the following statement:

DK_{3OP2} : I don't know what values the “modifies” clause affects

Applying the hypothesis testing resulted in a p-value of 0.02688 and a large effect size of 0.5054327. These values allow us to confidently conclude that the response distribution among developers is different.

When looking at the participants' responses, we verify that all participants who had an opinion formed about the statement rejected it. (Table 6.78)

Table 6.78: Frequency of each response to challenge DK_{3OP2}

Agreed	Strongly Agree	0	0	0.0%
	Somewhat Agree	0		
Neutral			3	15.0%
Disagreed	Somewhat Disagree	5	17	85.0%
	Strongly Disagree	12		
Not Sure			0	0.0%
Total			20	100%

The difference detected by the hypothesis testing is due to the conviction with which the participants reject the statement. The data shows that all experienced developers strongly disagreed with the statement (Table 6.80). In contrast, in beginner developers, the responses are evenly split between somewhat and strongly disagreeing (Table 6.79).

Table 6.79: Frequency of each response to challenge DK_{3OP2} from beginner developers

Agreed	Strongly Agree	0	0	0.0%
	Somewhat Agree	0		
Neutral			3	21.4%
Disagreed	Somewhat Disagree	5	11	78.6%
	Strongly Disagree	6		
Not Sure			0	0.0%
Total			14	100%

Table 6.80: Frequency of each response to challenge DK_{3OP2} from experienced developers

Agreed	Strongly Agree	0	0	0.0%
	Somewhat Agree	0		
Neutral			0	0.0%
Disagreed	Somewhat Disagree	0	6	100.0%
	Strongly Disagree	6		
Not Sure			0	0.0%
Total			6	100%

In conclusion, we verified that no developer encountered the challenge of not knowing exactly what the target of a “modifies” clause is, giving us evidence that the challenge is not relevant.

The last challenge in this group is similar to the previous one but regarding the “reads” clause. It resulted in the following statement:

DK_{3OP3} : I don't know what values the “reads” clause affects.

The application of Mann-Whitney U test resulted in a p-value of 0.01408 and with a large effect size of 0.5594. The test results allow us to confidently conclude that the distribution of responses between both groups is different.

The results show that all but one participant disagreed with the statement. (Table 6.81)

Table 6.81: Frequency of each response to challenge DK_{3OP3}

Agreed	Strongly Agree	0	1	5.0%
	Somewhat Agree	1		
Neutral			1	5.0%
Disagreed	Somewhat Disagree	7	18	90.0%
	Strongly Disagree	11		
Not Sure			0	0.0%
Total			20	100%

The difference detected by the hypothesis testing is, again, due to the conviction with which the participants reject the statement. The data shows that all experienced developers strongly disagreed with the statement (Table 6.83). In contrast, in beginner developers, the responses are evenly split between somewhat and strongly disagreeing (Table 6.82).

Table 6.82: Frequency of each response to challenge DK_{3OP3} from beginner developers

Agreed	Strongly Agree	0	1	7.1%
	Somewhat Agree	1		
Neutral			1	7.1%
Disagreed	Somewhat Disagree	7	12	85.7%
	Strongly Disagree	5		
Not Sure			0	0.0%
Total			14	100%

Table 6.83: Frequency of each response to challenge DK_{3OP3} from experienced developers

Agreed	Strongly Agree	0	0	0.0%
	Somewhat Agree	0		
Neutral			0	0.0%
Disagreed	Somewhat Disagree	0	6	100.0%
	Strongly Disagree	6		
Not Sure			0	0.0%
Total			6	100%

In conclusion, we verified that no developer encountered the challenge of not knowing exactly what the target of a “reads” clause is, giving us evidence that the challenge is not relevant.

The next section deals with the concept of opacity in Dafny. In Section 5.5.2, we explain in depth what opacity is in Dafny.

When looking at the responses to question DK04, detailed in Table 6.15, we discover that only 12 developers claimed to know what opacity is, with 9 developers explicitly saying that they do not know the concept and 3 unsure developers.

The challenge of distinguishing opaque statements from non-opaque resulted in the following statement:

DK₅ : How confident are you that you understand what statements are opaque and non-opaque?

The results of hypothesis testing resulted in a p-value of 0.4149 and a small effect size of 0.2267, suggesting that the distribution of responses is the same but that more data is needed to increase the effect size.

When looking at the results presented in Table 6.84 we can see that most participants are confident that they can distinguish opaque and non-opaque statements, with the majority answering from “somewhat confident” and above.

Table 6.84: Frequency of each response to challenge DK₅

Low Confidence	Not Confident	2	4	26.7%
	Slightly Confident	2		
Somewhat Confident		4	4	26.7%
High Confidence	Fairly Confident	3	7	46.7%
	Completely Confident	4		
Not Sure			0	0.0%
Total			15	100%

The exact distribution of responses is observed when looking at the data of the specific groups, both beginners, with results at Table 6.85, and experienced developers, with results at Table 6.86.

Table 6.85: Frequency of each response to challenge DK₅ from beginner developers

Low Confidence	Not Confident	2	3	30.0%
	Slightly Confident	1		
Somewhat Confident		3	3	30.0%
High Confidence	Fairly Confident	2	4	40.0%
	Completely Confident	2		
Not Sure			0	0.0%
Total			10	100%

Table 6.86: Frequency of each response to challenge DK₅ from experienced developers

Low Confidence	Not Confident	0	1	20.0%
	Slightly Confident	1		
Somewhat Confident		1	1	20.0%
High Confidence	Fairly Confident	1	3	60.0%
	Completely Confident	2		
Not Sure		0	0	0.0%
Total		5		100%

In conclusion, most developers familiar with opacity were confident they knew how to distinguish between opaque and non-opaque statements. However, 37.5% of the participants were knowledgeable in Dafny and did not even recognize the concept. The grasp of opacity is crucial to understanding what the verifier can deduct from our code and what needs to be explicitly stated in the contracts. Our study suggests that more effort should be put into divulging this aspect of Dafny.

The next question asked about the knowledge of the fuel attribute. This attribute allows the developer to specify the number of times a function is unfolded during verification. Of the 24 developers, only 8 (33%) knew about this attribute. From the challenge, we created the following sentence:

DK₇ : How confident are you that you understand how the fuel attribute works?

Since we have only a small sample of developers, we will analyze each response's frequency, not applying the usual hypothesis testing. Also, we will analyze the joined data of beginner and experienced developers since we do not have enough data to infer the differences between those groups.

Table 6.87: Frequency of each response to challenge DK₇

Low Confidence	Not Confident	0	2	25.0%
	Slightly Confident	2		
Somewhat Confident		2	2	25.0%
High Confidence	Fairly Confident	1	4	50.0%
	Completely Confident	3		
Not Sure		0	0	0.0%
Total		8		100%

Summarizing the data of Table 6.87 developers that answered the question answered with level 3, "Somewhat Confident," or above, showing that the users who knew about fuel were confident that they were knowledgeable about its working mechanism. Still, more data is needed to understand the impact of most participants not knowing about the concept, which may be unknown due to niche applications.

The next challenge concerns the knowledge of partial functions; these functions can be used as high-order functions with the preconditions necessary for the arguments already specified. To measure the level of confidence the participants had when using partial functions as arguments, the following statement was formulated:

DK₉: How confident are you that you can use partial functions to reason about the fulfillment of high-order functions function argument requirements?

Of the 24 participants, 12 (50%) knew about the concept, and one was unsure. In total, 13 participants answered the question. Since we have a more balanced distribution of beginners with 7 answers and experienced developers with 6 answers, we applied the Mann-Whitney U test.

We obtained a p-value of 0.09151 with a moderate magnitude of 0.4884, so we can assume the distribution of responses is the same with moderate confidence.

In Table 6.88, we present the results relative to the totality of responses. As can be observed, the majority of participants are at least somewhat confident that they know how to use partial functions in this context.

Table 6.88: Frequency of each response to challenge *DK₉*

Low Confidence	Not Confident	3	4	30.8%
	Slightly Confident	1		
Somewhat Confident		4	4	30.8%
High Confidence	Fairly Confident	2	5	38.5%
	Completely Confident	3		
Not Sure		0	0	0.0%
Total		13	13	100%

When considering the results for beginners (Table 6.89) and experienced developers (Table 6.90), we observe a similar distribution with the majority of participants somewhat confident or above. However, most experienced developers chose a response that was considered high confidence, while only one beginner answered in the same category. Also, relatively more beginners answered the question with “Not Confident,” with a relative distribution of 29% against 17% experienced developers. Still, this difference has not been validated by our hypothesis testing.

Table 6.89: Frequency of each response to challenge *DK₉* from beginner developers

Low Confidence	Not Confident	2	3	42.9%
	Slightly Confident	1		
Somewhat Confident		3	3	42.9%
High Confidence	Fairly Confident	1	1	14.3%
	Completely Confident	0		
Not Sure		0	0	0.0%
Total		7	7	100%

Table 6.90: Frequency of each response to challenge DK₉ from experienced developers

Low Confidence	Not Confident	1	1	16.7%
	Slightly Confident	0		
Somewhat Confident		1	1	16.7%
High Confidence	Fairly Confident	1	4	66.7%
	Completely Confident	3		
Not Sure		0	0	0.0%
Total		6		100%

In conclusion, most developers who knew about partial functions were confident about their skills when using them to reason about the pre-conditions of high-order functions. However, a high number of developers, 11 (46%), did not know about partial functions, suggesting that their knowledge should spread more.

The last Dafny-specific question is about using witnesses to prove the non-emptiness of type definitions. This resulted in the following statement:

DK₁₁: How confident are you that you can prove the non-emptiness of type definitions by providing witnesses?

From the 24 inquired participants, only 7 knew about the concept, and two were unsure, resulting in 9 (38 %) responses. Given the low number of responses, we will not apply hypothesis testing analyzing just the frequency of each response.

Table 6.91: Frequency of each response to challenge DK₁₁

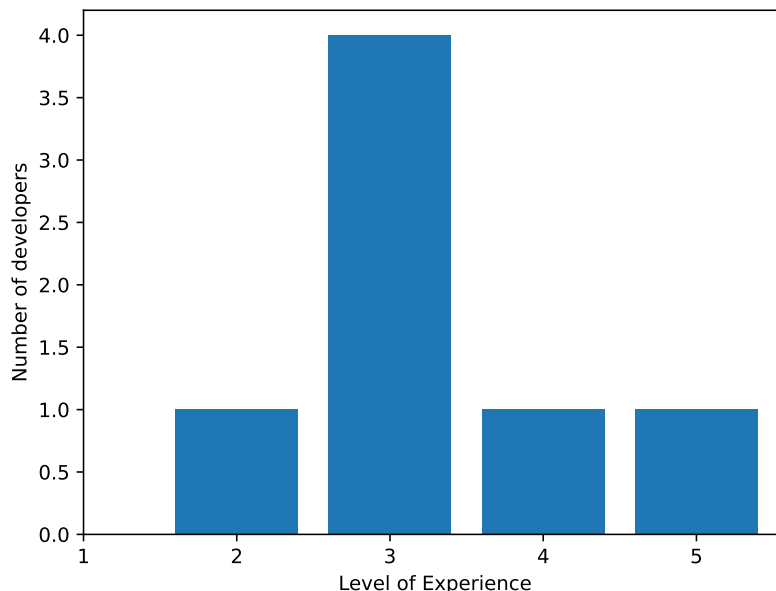
Low Confidence	Not Confident	1	3	33.3%
	Slightly Confident	2		
Somewhat Confident		1	1	11.1%
High Confidence	Fairly Confident	3	5	55.6%
	Completely Confident	2		
Not Sure		0	0	0.0%
Total		9		100%

Most developers who knew about the concept were at least somewhat confident that they could prove the non-emptiness of type definitions with witnesses. However, there was a concerning number of participants, 15 (63 %) in total, who did not even know about the concept, suggesting that its knowledge should be more spread. Still, more data is needed to understand the impact of most participants not knowing about the concept, which may be unknown due to niche applications.

6.4.7 Frama-C WP Plugin Challenges

In this section, we analyze the challenges specific to Frama-C, namely the WP Plugin. For simplicity reasons, we will use the terms WP Plugin and Frama-C interchangeably, despite one referring to a plugin and the other being the tool. Since these are language-specific questions, we wanted to understand how knowledgeable the participants are about Frama-C. We refer to Table 6.23 for the levels' specifications and meaning of each experience level.

Figure 6.2: Distribution of responses to question WPK01



In Figure 6.2, we observe the responses to the questions WPK01 detailed at Table 6.23. As can be seen, similar to what was observed with Dafny, most developers cannot complete intermediate proofs without guidance. Still, all Frama-C developers are considered experienced, considering their years of experience with verification. Because of that, we were not able to apply our hypothesis testing.

The first group of challenges is related to WP Plugin memory models. Memory models allow the developer to specify the assumptions of C's memory allocation functions. Of the seven participants with knowledge of Frama-C, four knew about memory models, two were unsure, and one did not, resulting in six responses for this group.

The first question asked the participants if they knew the difference between the different memory models of Frama-C, this resulted in the following statement:

WPK_{3OP1}: How confident are you that you understand the difference between Frama-C's memory models?

When looking at the responses to the question, shown in Table 6.92, we verify that most developers are confident that they understand the differences between WP Plugin memory models, suggesting that the challenge is not prevalent among experienced verification developers. However, more data is necessary to validate the challenge, given that only 6 participants answered.

Table 6.92: Frequency of each response to challenge WPK_{3OP1}

Low Confidence	Not Confident	1	1	16.7%
	Slightly Confident	0		
Somewhat Confident		3	3	50.0%
High Confidence	Fairly Confident	1	2	33.3%
	Completely Confident	1		
Not Sure		0	0	0.0%
Total		6		100%

The next challenge is related to the choice of the right memory model that allows the developer to verify the properties of the code. This challenge was translated as the following statement:

WPK_{3OP2}: How confident are you that you can choose the correct memory model to prove the properties of your code?

The responses to this challenge allow us to conclude that most developers are confident about choosing memory models, with just one participant choosing “Not Confident.” However, given the low number of answers, more data is necessary to validate the challenge.

Table 6.93: Frequency of each response to challenge WPK_{3OP2}

Low Confidence	Not Confident	1	3	50.0%
	Slightly Confident	2		
Somewhat Confident		1	1	16.7%
High Confidence	Fairly Confident	2	2	33.3%
	Completely Confident	0		
Not Sure		0	0	0.0%
Total		6		100%

The last challenge regarding memory models is about the use of the “loop assigns” statement. The challenge of identifying the statements that need this keyword was formulated as the following statement:

WPK_{3OP3}: How confident are you that you are able to choose what variables you need to declare in “loop assigns” statements?

The answers to this question show that most developers are able to understand what variables needs the “loop assigns” statement. However, more data is necessary to validate the challenge, given that only 6 participants answered.

Table 6.94: Frequency of each response to challenge WPK_{3OP3}

Low Confidence	Not Confident	0	1	16.7%
	Slightly Confident	1		
Somewhat Confident		2	2	33.3%
High Confidence	Fairly Confident	1	3	50.0%
	Completely Confident	2		
Not Sure		0	0	0.0%
Total		6		100%

The last group of challenges asks questions about the general understanding of Fram-C and the WP Plugin. The first question concerns the features of C Fram-C supports; it is relevant because Fram-C supports only a subset of C's specification. It was phrased as follows:

WPK_{4OP1}: I find features of C specifications not implemented by Fram-C and of which I'm not aware.

The responses to the question are displayed in table 6.95, and suggest that most developers that understood the question found features of C not implemented by Fram-C. However, many "Not Sure" responses suggest that the statement should have been phrased differently. Moreover, the low number of responses prevents us from confidently concluding that the challenge was found.

Table 6.95: Frequency of each response to challenge WPK_{4OP1}

Agreed	Strongly Agree	1	3	42.9%	
	Somewhat Agree	2			
Neutral			0		0.0%
Disagreed	Somewhat Disagree	0	1	14.3%	
	Strongly Disagree	1			
Not Sure			3		42.9%
Total			7		100%

Finally, the last challenge is related to the number of plugins Fram-C has and the challenge of finding the right plugin for a specific use case. The challenge was phrased as follows:

WPK_{4OP2}: I find features missing in Fram-C that other plugins have implemented.

The responses show that most developers have found features missing in WP-Plugin that other plugins have implemented. However, the low number of responses prevents us from confidently concluding that the challenge was found.

Table 6.96: Frequency of each response to challenge WPK_{4OP2}

Agreed	Strongly Agree	0	3	42.9%	
	Somewhat Agree	3			
Neutral			1		14.3%
Disagreed	Somewhat Disagree	1	2	28.6%	
	Strongly Disagree	1			
Not Sure			1		14.3%
Total			7		100%

6.4.8 Additional Challenges

In this section, we will analyze the challenges expressed in the open-ended question about verification. We will separate each challenge into categories and mention which participants expressed them.

Understanding Proof Errors

The participants placed in this category expressed challenges related to the understanding of proof errors, namely in proofs where the verifier timed out or the error displayed was simply unknown.

Of the 31 participants, 7 expressed this challenge, 6 were experienced, and 1 was a beginner. Although this only represents approximately 23% of the participants, it was the most mentioned challenge. The results of the next section, Overcoming Challenges, later revealed that most participants need a counter-example system to further elicit errors in the proofs.

Understanding Purpose

Two beginner participants from the 2023/2024 edition of Formal Methods for Critical Systems, a challenge related to understanding how verification can be applied in large-scale projects. Since both beginners had the same introduction to verification, it might reveal an issue with the course that may not apply to other beginners. Further data is needed to verify if this is the case.

Verification Time

This challenge encompasses complaints about the verification taking a long time or timing out, not allowing the developers to gather feedback about the proof's correctness. It was identified by two beginners and one experienced developer. It reinforces the challenge identified in Section 5.5.4, however, more data is necessary to understand its prevalence among these groups.

Scalability Issues

This challenge encompasses scalability issues when verifying larger-scale or more complex projects like state machines. It was identified by two experienced developers.

Verifier Behaviour

This section encompasses challenges related to understating “obscure“ verifier behavior. It was particularly observed as the challenge of understanding what the verifier can deduce about the code and how the ability to deduce properties changes across different verifier versions. One beginner and one experienced developer observed it.

Other Challenges

This subsection contains challenges only identified by one participant.

The first challenge was identified by one beginner participant and is about knowing when to update the memory representation of class objects. The formulation of the challenge is ambiguous, but we speculate that it might be related to constructs like *Repr* that store the current data in a data structure, useful, for instance, in trees where the data is distributed across multiple nodes.

In the second challenge, the beginner participant claims implementing contracts is hard. However, they do not specify what aspect of it is hard.

In the third challenge, the experienced participant mentions gathering the know-how of the verification language and proof environment that, due to lacking documentation, relies on oral transmission. The challenge seems that it could happen in other development contexts and not only in verification. More data is necessary to understand how this challenge affects verification environments specifically.

The fourth challenge was identified by an experienced developer and is related to the difficulty associated with proofs about bitwise operations and heterogeneous pointer casts between different types. The participant only knew Frama-C, so more data is necessary to understand how prevalent this challenge is in other languages and Frama-C.

In the fifth challenge, a beginner developer complained about the support of verification languages by IDEs. However, since the participant selected multiple languages, it is hard to pinpoint which one is problematic.

The sixth challenge was also identified by a beginner developer who said that they felt that there was no strategy to develop invariants and that deriving them was “guesswork.”

The seventh and final challenge is verifying existing algorithms by a beginner developer.

6.4.9 Overcoming Challenges

In this section, we validate the relevance of the improvements included in Section 6.2.9, *Overcoming Challenges (OC)*, where the reasoning for each improvement is also presented. In closed questions, we removed the “I’m not sure” option from the hypothesis testing to keep the Likert scale balanced and because this option was provided as a way for participants who did not want to answer to an improvement to still be able to complete the survey.

Open Questions

In this section, we will analyze the responses to the open question of this section. This question was presented before the other closed questions to prevent bias and allow the participants to think more freely without being conditioned by the options they already saw.

The first improvement was proposed by two experienced developers who wanted ways to expose the internal proof mechanism of the SMT to improve debugging.

The second is related to the automation of tasks. One beginner developer mentioned that the proof must rely less on the developer but provided no suggestions for improvements. One developer proposed automation for “easy tasks,” and another proposed heuristics to provide hints on how to rewrite code to be more convenient for verification. Two developers proposed automatic suggestions for verification strategies. One experienced developer mentioned automation generally without a target.

Improvements in documentation and learning material were also suggested. Five developers, two beginners and three experienced, did not specify the target of the improvements. One beginner developer mentioned that more documentation about coming up with invariants would be useful.

Four developers mentioned that reusable building blocks should be provided. Two beginner developers mentioned libraries, and the experienced developer only mentioned “predefined building blocks.”

Two developers, one beginner and one experienced, mentioned that Verification-Aware languages should be publicized more, with the experienced developer mentioning that it should be done at university.

When facing errors, two beginners and one experienced developer mentioned that more explanatory messages would help them understand why the verification fails. Another experienced developer mentioned user-readable counterexamples. Also, two experienced developers mentioned better integration with existing profiling and debugging tools would be helpful.

Two experienced developers suggested that more investment should be made in verification tools, with one suggesting commercial support.

One experienced developer mentioned that the efficiency of generated code should be improved and that the behavior of the verification tools should be predictable in terms of what can be verified.

One beginner developer mentioned that Dafny should be in package managers like Aptitude.

Finally, one beginner developer mentioned that he did not understand the purpose of verification languages as standalone tools and that their functionality should be integrated into existing languages.

Summarizing, we identified 10 categories of improvement for Verification-Aware languages. However, further research is needed to understand how these improvements should be prioritized and how they would impact beginners and experienced developers.

Closed Questions

In this section, we will analyze the closed questions of this section with pre-determined response options.

The first improvement regards the inclusion of user-readable counter-examples that could allow users to “debug” the proofs. It was translated as the following statement:

OC_1 : Create user-readable counterexamples that could elicit errors in the proof

The hypothesis testing resulted in a p-value of 0.6927 and a negligible effect size of 0.07872, suggesting that the distribution of responses is the same among beginner and experienced developers but that more data is needed to trust the results of the test.

Looking at the results in the Table 6.97, we see that most participants agreed that having user-readable counter-examples would be very important.

Table 6.97: Frequency of each response to challenge OC₁

Not Important	0	0.0%
Somewhat Important	4	12.9%
Very Important	26	83.9%
Not Sure	1	3.2%
Total	31	100%

The responses from both groups corroborate the results of the hypothesis testing given that the distribution of responses among beginners (Table 6.98) and experienced (Table 6.99) developers is similar.

Table 6.98: Frequency of each response to challenge OC₁ from beginner developers

Not Important	0	0.0%
Somewhat Important	2	11.1%
Very Important	16	88.9%
Not Sure	0	0.0%
Total	18	100%

Table 6.99: Frequency of each response to challenge OC₁ from experienced developers

Not Important	0	0.0%
Somewhat Important	2	15.4%
Very Important	10	76.9%
Not Sure	1	7.7%
Total	13	100%

In conclusion, most participants agree that it would be useful to have some way of using counter-examples to elicit errors of the proofs.

The next improvement we analyze is the inclusion of tutorials with common projects, such as tic-tac-toe, to understand how verification should be included in previously known code. This improvement resulted in the following statement:

OC₂: Improve official tutorials to include projects that programmers with previous knowledge know

Applying the hypothesis testing resulted in a p-value of 0.1052 with a moderate effect size of 0.3005, suggesting that both groups' responses are the same.

When looking at the joined responses of both groups, we see that most participants consider the improvement relevant.

Table 6.100: Frequency of each response to challenge OC₂

Not Important	1	3.2%
Somewhat Important	9	29.0%
Very Important	20	64.5%
Not Sure	1	3.2%
Total	31	100%

However, when looking at the responses for beginner (Table 6.101) and experienced (Table 6.102) developers we observe a small difference in the intensity of the response, with relatively more beginners, 77.8%, considering the improvement very important, against only 46.2% of the experienced developers.

Table 6.101: Frequency of each response to challenge OC₂ from beginner developers

Not Important	0	0.0%
Somewhat Important	4	22.2%
Very Important	14	77.8%
Not Sure	0	0.0%
Total	18	100%

Table 6.102: Frequency of each response to challenge OC₂ from experienced developers

Not Important	1	7.7%
Somewhat Important	5	38.5%
Very Important	6	46.2%
Not Sure	1	7.7%
Total	13	100%

In conclusion, the improvement is considered important by most developers. Still, despite not being validated by the hypothesis testing, the differences in the responses of both groups reveal that the improvements would be more important for beginners.

The next improvement concerns including a search function to search concepts within the documentation. This improvement resulted in the following statement:

OC₃ : Improve documentation search mechanisms with a search function

Applying the hypothesis test to the responses to this question resulted in a p-value of 0.02822 with a moderate effect size of 0.4122. These values allow us to conclude that there is a difference between the distribution of responses among beginner and experienced developers.

When looking at the responses of both developers, shown in table 6.103, we observe that most consider adding a search tool an important improvement.

Table 6.103: Frequency of each response to challenge OC₃ from experienced developers

Not Important	1	3.2%
Somewhat Important	14	45.2%
Very Important	14	45.2%
Not Sure	2	6.5%
Total	31	100%

Still, when looking at the responses of both groups separately, we observe that beginner developers, whose responses are displayed in Table 6.104, attributed more importance to the improvement, with 66.7% stating that the improvement would be very important in contrast with experienced developers, with only 15.4% attributing the same level of importance (Table 6.105).

Table 6.104: Frequency of each response to challenge OC₃ from beginner developers

Not Important	1	5.6%
Somewhat Important	5	27.8%
Very Important	12	66.7%
Not Sure	0	0.0%
Total	18	100%

Table 6.105: Frequency of each response to challenge OC₃ from experienced developers

Not Important	0	0.0%
Somewhat Important	9	69.2%
Very Important	2	15.4%
Not Sure	2	15.4%
Total	13	100%

In conclusion, although the improvement seems important, our data suggests it would be more impactful for beginner developers than experienced developers.

The next improvement that we hypothesized was the expansion of documentation themes. This improvement resulted in the following statement:

OC₄ : Expand the themes that are included in the documentation

The hypothesis testing resulted in a p-value of 0.4688 with a small effect size of 0.1512. This hints that there is no difference between the responses of both groups. However, more data would be necessary to increase the effect size.

The results of the question show that most developers consider this improvement important (Table 6.106). Still, there is a large number of participants who are not sure about the impact of this improvement. This can also mean that the statement was not well formulated in some way. However, we do not have enough data to support this claim.

Table 6.106: Frequency of each response to challenge OC₄

Not Important	1	3.2%
Somewhat Important	11	35.5%
Very Important	13	41.9%
Not Sure	6	19.4%
Total	31	100%

When looking at the responses of beginner (Table 6.107) and experienced (Table 6.108) developers, we do not observe a significant difference in the distribution of responses, this is corroborated by the result of the hypothesis testing.

Table 6.107: Frequency of each response to challenge OC₄ from beginner developers

Not Important	1	5.6%
Somewhat Important	7	38.9%
Very Important	7	38.9%
Not Sure	3	16.7%
Total	18	100%

Table 6.108: Frequency of each response to challenge OC₄ from experienced developers

Not Important	0	0.0%
Somewhat Important	4	30.8%
Very Important	6	46.2%
Not Sure	3	23.1%
Total	13	100%

In conclusion, these improvements seem as important to beginners as they are for experienced developers. However, the large number of “Somewhat Important” and “Not Sure” responses makes us believe this improvement would be less impactful than the ones analyzed before.

Finally, the last improvement concerns the inclusion of Verification-Aware languages in undergraduate programs. This improvement has the potential to be very impactful given that most inquired developers had their first contact with Verification-Aware languages because of a choice or by chance. This improvement was translated as the following statement:

OC₅ : Include verification-aware language in undergraduate programs

The hypothesis testing yielded a p-value of 0.7657 with a negligible effect size of 0.0614. This indicates that the response distribution among both developer groups is the same. However, more data is necessary to apply the test.

When looking at the opinions about the improvement, shown in Table 6.109, we observe that most developers consider the improvement very important. Corroborating our hypothesis that this improvement would be very impactful in adopting Verification-Aware languages.

Table 6.109: Frequency of each response to challenge OC₅

Not Important	1	3.2%
Somewhat Important	11	35.5%
Very Important	16	51.6%
Not Sure	3	9.7%
Total	31	100%

The distribution of responses among beginner 6.110 and experienced 6.111 developers is the same, with the majority considering the improvement as very important in both groups.

Table 6.110: Frequency of each response to challenge OC₅ from beginner developers

Not Important	0	0.0%
Somewhat Important	8	44.4%
Very Important	9	50.0%
Not Sure	1	5.6%
Total	18	100%

Table 6.111: Frequency of each response to challenge OC₅ from experienced developers

Not Important	1	7.7%
Somewhat Important	3	23.1%
Very Important	7	53.8%
Not Sure	2	15.4%
Total	13	100%

In conclusion, including Verification-Aware languages in undergraduate programs would anticipate contact with these tools, possibly improving their adoption. The data from our survey, from beginners and experienced alike, corroborate our hypothesis.

6.5 Conclusion

In conclusion, in this chapter, we described the methodology to answer RQ1 and RQ3 and validate the answers to RQ2.

With the previously mention objectives in mind we built a survey.

The first section of the survey asked about the developers' verification background and was used to answer RQ1.

The next six sections of the survey were used complement the answer to RQ2 and included statements to validate the 24 challenges identified at Chapter 5. From the 24 challenges only five were not faced by the developers. The seventh section also included an open-ended question where seven additional challenges were identified.

The eighth section answered RQ3 and provided insights into what improvements were relevant and how they would affect the two groups, beginners and experienced developers. Additionally, this section provided an open-ended question that expanded on the improvements we envisioned.

Finally the last section provided insights into the demographics of each participant.

Chapter 7

Conclusion

7.1 Summary

The teaching and adoption of VA languages still pose multiple challenges. The delay in developers becoming aware of them also hinders their adoption. Still, ensuring reliability is more important than ever in a world of increasing digitization.

This work presents a broad analysis of developers' challenges when using VA languages, containing feedback from both beginners and experienced developers with many years of experience. We trained a machine learning model with 1420 questions, of which 145 were analyzed after clustering. The analysis of the clustering results allowed us to discover 24 challenges that were then graded by 31 survey participants, some by agreeing with affirmations, others by saying with what confidence they could perform specific actions. We also developed five suggestions for improvement that the participants in the survey evaluated and discovered another 12 areas of improvement in our open question.

Overall, this work can pave the way for future investigations of specific challenges and improvements. It also provides valuable information to direct the development of VA languages.

Following this section,

7.2 Answers to Research Questions

This section will summarize the response to the research questions formulated in Chapter 1.

RQ1: What is the context of first contact with Verification-Aware languages?

What we discovered with the responses to questions DC04 and DC05, described in tables 6.7 and 6.8 respectively, is that the majority of developers learn about VA languages as a choice of education or career paths. This fact naturally limits the number of developers with knowledge in verification compared to mandatory themes.

RQ2: What are the challenges that Verification-Aware language practitioners face?

After analyzing the responses to the questions made to the participants of the survey we found that all challenges, but the challenges ES_1 , ES_2 , ES_3 , DK_{3OP2} , DK_{3OP3} , were faced by a large number of participants. Resulting in 19 (79%) validated challenges. Moreover, the responses to questions DK_4 , DK_6 , DK_8 , and DK_{10} revealed a lack of knowledge of Dafny’s core concepts essential for successful verifications. Regarding the number of responses, in Section 6.4.7, we were unable to compare beginners and experienced developers due to a lack of responses.

Finally, we identified 13 new challenges the developers faced and paved the way for future investigation.

RQ3: What improvements can be made to the existing tools to overcome those challenges?

The responses to Section 6.4.9, Overcoming Challenges, validate all the improvements proposed in question OC02, described in Table 6.29. Moreover, the survey participants revealed 12 areas that we did not consider providing valuable insights for future work that can be developed to evaluate those improvements.

7.3 Limitations and Threats to Validity

In this section, we discuss the limitations and threats to the validity of our study. Firstly we will analyze the topic modeling and then the survey elaboration and analysis.

7.3.1 Topic Modeling

Since we followed the methodology of Han et al. (2020), our study suffers from the same limitations and threats to validity as theirs. We will adapt the threats identified by the authors to the context of our work.

Firstly, the usage of question bodies introduces more data and context to the algorithm. However, there is also noise associated with off-topic discussions, attempts to solve the error, and errors thrown by the verifier. These details might obfuscate the vocabulary associated with the challenge of the developer. To alleviate this threat, when determining the LDA topics, apart from looking at the most relevant topic words, we also analyzed the question’s bodies when classifying them.

Secondly, although we have incorporated StackOverflow in our study, it may not be enough because we did not get questions related to some tools, and for the tools we got, it might be insufficient.

Thirdly, the mapping between languages and StackOverflow questions might not have been perfect and left out some relevant questions.

7.3.2 Survey

This section will analyze the limitations and threats to the validity of the elaborated validation survey.

Firstly, the number of participants often resulted in small effect sizes suggesting that more participants were needed to assess the differences between beginner and experienced developers

Secondly, the translation of challenges to affirmations might have left off essential context for participants to really understand each challenge.

Thirdly, the survey format did not allow for further elaboration on each response and how the participants perceived the affirmations

7.4 Future Work

Despite the work fulfilling its goal of answering the research questions defined in Chapter 1, we identify some areas in which the work could be further developed.

Firstly, we identify some research directions related to the scope of the work:

- **Inclusion of other sources:** This work only considered StackOverflow to gather the challenges of developers. However, some languages were left off from our analysis since the chosen forum did not contain questions regarding some tools.
- **More participants:** Our survey only received 31 responses making the comparison between beginners and experienced developers dubious due to low effect sizes.

Secondly, we identify research directions related to the discovered challenges and improvements:

- **Validate New Discoveries:** The participants expressed several new challenges and improvements, these improvements were not validated because they were only discovered through participant's responses.
- **Rank The Challenges:** This work only focused on identifying challenges and directions of improvement. However, since tool development resources are limited, further investigation is necessary to rank the importance of each challenge and improvement for each tool.
- **Implement Improvements:** Future work should implement solutions that address the identified challenges. For some of the challenges, the application of recent artificial intelligence techniques seem a promising avenue [Silva et al. \(2024\)](#); [Trigo \(2024\)](#).

References

- Ahmad Abdellatif, Diego Costa, Khaled Badran, Rabe Abdalkareem, and Emad Shihab. Challenges in Chatbot Development: A Study of Stack Overflow Posts. In *Proceedings of the 17th International Conference on Mining Software Repositories*, MSR '20, pages 174–185, New York, NY, USA, September 2020. Association for Computing Machinery. doi: 10.1145/3379597.3387472.
- Syed Ahmed and Mehdi Bagherzadeh. What do concurrency developers ask about? a large-scale study using stack overflow. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '18, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450358231. doi: 10.1145/3239235.3239524.
- Md Abdullah Al Alamin, Sanjay Malakar, Gias Uddin, Sadia Afroz, Tameem Bin Haider, and Anindya Iqbal. An empirical study of developer discussions on low-code software development challenges. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, pages 46–57, 2021. doi: 10.1109/MSR52588.2021.00018.
- Jeff Atwood. Stack Overflow Creative Commons Data Dump, 2009. URL <https://stackoverflow.blog/2009/06/04/stack-overflow-creative-commons-data-dump>. [Online]; accessed 03 June 2024.
- Shariq Aziz Butt. Study of agile methodology with the cloud. *Pacific Science Review B: Humanities and Social Sciences*, 2(1):22–28, 2016. ISSN 2405-8831. doi: <https://doi.org/10.1016/j.psr.2016.09.007>.
- B. Meyer. Applying 'Design by Contract'. *Computer*, 25(10):40–51, 1992. ISSN 1558-0814. doi: 10.1109/2.161279.
- Mehdi Bagherzadeh and Raffi Khatchadourian. Going big: a large-scale study on what big data developers ask. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2019, pages 432–442, New York, NY, USA, August 2019. Association for Computing Machinery. doi: 10.1145/3338906.3338939.
- Sebastian Baltes, Lorik Dumani, Christoph Treude, and Stephan Diehl. Sotorrent: reconstructing and analyzing the evolution of stack overflow posts. In *Proceedings of the 15th International Conference on Mining Software Repositories*, MSR '18, page 319–330, New York, NY, USA, 2018. Association for Computing Machinery. doi: 10.1145/3196398.3196430.
- Anton Barua, Stephen W. Thomas, and Ahmed E. Hassan. What are developers talking about? An analysis of topics and trends in Stack Overflow. *Empirical Software Engineering*, 19(3): 619–654, June 2014. doi: 10.1007/s10664-012-9231-y.

- Patrick Baudin, François Bobot, Loïc Correnson, Zaynah Dargaye, and Allan Blanchard. *WP Plug-in Manual For Frama-C 29.0 (Copper)*. CEA-List, June 2024.
- David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.
- Guilan Dai, Xiaoying Bai, Yongbo Wang, and Fengjun Dai. Contract-based testing for web services. In *31st Annual International Computer Software and Applications Conference (COMPSAC 2007)*, volume 1, pages 517–526, 2007. doi: 10.1109/COMPSAC.2007.100.
- Adel Djoudi, Martin Hána, and Nikolai Kosmatov. Formal Verification of a JavaCard Virtual Machine with Frama-C. In *Formal Methods*, pages 427–444. Springer International Publishing, 2021. ISBN 978-3-030-90870-6.
- Frank Dordowsky. An experimental Study using ACSL and Frama-C to formulate and verify Low-Level Requirements from a DO-178C compliant Avionics Project. *Electronic Proceedings in Theoretical Computer Science*, 187:28–41, 2015. ISSN 2075-2180. doi: 10.4204/eptcs.187.3.
- Mohamed Raed El aoun, Heng Li, Foutse Khomh, and Moses Openja. Understanding quantum software engineering challenges an empirical study on stack exchange forums and github issues. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 343–354, 2021. doi: 10.1109/ICSME52107.2021.00037.
- Robert W. Floyd. *Assigning Meanings to Programs*, pages 65–81. Springer Netherlands, Dordrecht, 1967. ISBN 978-94-011-1793-7. doi: 10.1007/978-94-011-1793-7_4.
- David Fullerton. Stack Exchange Creative Commons data now hosted by the Internet Archive, 2014. URL <https://stackoverflow.blog/2014/01/23/stack-exchange-cc-data-now-hosted-by-the-internet-archive/>. [Online]; accessed 03 June 2024.
- Dilian Gurov, Christian Lidström, Mattias Nyberg, and Jonas Westman. Deductive Functional Verification of Safety-Critical Embedded C-Code: An Experience Report. In Laure Petrucci, Cristina Seculeanu, and Ana Cavalcanti, editors, *Critical Systems: Formal Methods and Automated Verification*, pages 3–18. Springer International Publishing, 2017. ISBN 978-3-319-67113-0.
- Matthias Güdemann. Online Teaching of Verification of C Programs in Applied Computer Science. In João F. Ferreira, Alexandra Mendes, and Claudio Menghi, editors, *Formal Methods Teaching*, pages 18–34. Springer International Publishing, 2021. ISBN 978-3-030-91550-6.
- Junxiao Han, Emad Shihab, Zhiyuan Wan, Shuiguang Deng, and Xin Xia. What do Programmers Discuss about Deep Learning Frameworks. *Empirical Software Engineering*, 25(4):2694–2747, July 2020. doi: 10.1007/s10664-020-09819-6.
- C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, oct 1969. ISSN 0001-0782. doi: 10.1145/363235.363259.
- Reiner Hähnle and Marieke Huisman. Deductive software verification: From pen-and-paper proofs to industrial tools. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10000:345 – 373, 2019. doi: 10.1007/978-3-319-91908-9_18.

- S. Ilieva, P. Ivanov, and E. Stefanova. Analyses of an agile methodology implementation. In *Proceedings. 30th Euromicro Conference, 2004.*, pages 326–333, 2004. doi: 10.1109/EURMIC.2004.1333387.
- Alexander Knüppel, Thomas Thüm, Carsten Pardylla, and Ina Schaefer. Experience Report on Formally Verifying Parts of OpenJDK’s API with KeY. *Electronic Proceedings in Theoretical Computer Science*, 284:53–70, 2018. ISSN 2075-2180. doi: 10.4204/eptcs.284.5. URL <http://dx.doi.org/10.4204/EPTCS.284.5>. Publisher: Open Publishing Association.
- K. Rustan M. Leino, David R. Cok, and Dafny contributors. Dafny Reference Manual, 2024. URL <https://github.com/dafny-lang/dafny/releases/download/v4.7.0/DafnyRef.pdf>. [Online]; accessed 06th July 2024.
- Johannes Link. Chapter 4 - Test Ideas and Heuristics. In *Unit Testing in Java*, The Morgan Kaufmann Series in Software Engineering and Programming, pages 65–90. Morgan Kaufmann, San Francisco, 2003. ISBN 978-1-55860-868-9. doi: <https://doi.org/10.1016/B978-155860868-9/50006-7>.
- Andrew Kachites McCallum. Mallet: A machine learning for language toolkit. <http://mallet.cs.umass.edu>, 2002.
- Bertrand Meyer. *Design by Contract*. Interactive Software Engineering Inc., 1986.
- David Mimno, Hanna M. Wallach, Edmund Talley, Miriam Leenders, and Andrew McCallum. Optimizing semantic coherence in topic models. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP ’11, page 262–272, USA, 2011. Association for Computational Linguistics. ISBN 9781937284114.
- Hendrik Müller, Aaron Sedley, and Elizabeth Ferrall-Nunge. *Survey Research in HCI*, pages 229–266. Springer New York, New York, NY, 2014. ISBN 978-1-4939-0378-8. doi: 10.1007/978-1-4939-0378-8_10. URL https://doi.org/10.1007/978-1-4939-0378-8_10.
- Glenford J. Myers. *Software Reliability: Principles and Practices*. Wiley-Interscience, 1976.
- Nadim Nachar. The mann-whitney u: A test for assessing whether two independent samples come from the same distribution. *Tutorials in Quantitative Methods for Psychology*, 4, 03 2008. doi: 10.20982/tqmp.04.1.p013.
- Mattias Nyberg, Dilian Gurov, Christian Lidström, Andreas Rasmusson, and Jonas Westman. Formal verification in automotive industry: Enablers and obstacles. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11247 LNCS:139 – 158, 2018. doi: 10.1007/978-3-030-03427-6_14.
- Moses Openja, Bram Adams, and Foutse Khomh. Analysis of modern release engineering topics : – a large-scale study using stackoverflow –. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 104–114, 2020. doi: 10.1109/ICSME46990.2020.00020.
- David L. Parnas, A. John van Schouwen, and Shu Po Kwan. Evaluation of safety-critical software. *Commun. ACM*, 33(6):636–648, 1990. ISSN 0001-0782. doi: 10.1145/78973.78974.
- Anthony Peruma, Steven Simmons, Eman Abdullah AlOmar, Christian D. Newman, Mohamed Wiem Mkaouer, and Ali Ouni. How do i refactor this? An empirical study on refactoring

- trends and topics in Stack Overflow. *Empirical Software Engineering*, 27(1), January 2022. doi: 10.1007/s10664-021-10045-x.
- Michael Röder, Andreas Both, and Alexander Hinneburg. Exploring the Space of Topic Coherence Measures. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, WSDM '15, pages 399–408, New York, NY, USA, February 2015. Association for Computing Machinery. doi: 10.1145/2684822.2685324.
- Marc Schoolderman, Sjaak Smetsers, and Marko Van Eekelen. Is Deductive Program Verification Mature Enough to be Taught to Software Engineers? In *Proceedings of the 8th Computer Science Education Research Conference*, pages 50–57. ACM, 2019. ISBN 978-1-4503-7717-1. doi: 10.1145/3375258.3375265. URL <https://dl.acm.org/doi/10.1145/3375258.3375265>.
- Yuvaraj Selvaraj, Wolfgang Ahrendt, and Martin Fabian. Verification of Decision Making Software in an Autonomous Vehicle: An Industrial Case Study. In *Formal Methods for Industrial Critical Systems: 24th International Conference, FMICS 2019, Amsterdam, The Netherlands, August 30–31, 2019, Proceedings*, pages 143–159. Springer-Verlag, 2019. ISBN 978-3-030-27007-0. doi: 10.1007/978-3-030-27008-7_9. URL https://doi.org/10.1007/978-3-030-27008-7_9.
- Carson Sievert and Kenneth Shirley. LDAvis: A method for visualizing and interpreting topics. In Jason Chuang, Spence Green, Marti Hearst, Jeffrey Heer, and Philipp Koehn, editors, *Proceedings of the Workshop on Interactive Language Learning, Visualization, and Interfaces*, pages 63–70, Baltimore, Maryland, USA, June 2014. Association for Computational Linguistics. doi: 10.3115/v1/W14-3110.
- Julia Silge and David Robinson. Text Mining with R: A Tidy Approach, 2024. URL <https://www.tidyttextmining.com/>. [Online]; accessed 30 May 2024.
- Álvaro F. Silva, Alexandra Mendes, and João F. Ferreira. Leveraging Large Language Models to boost Dafny's developers productivity. In *Proceedings of the 2024 IEEE/ACM 12th International Conference on Formal Methods in Software Engineering (FormaliSE)*, FormaliSE '24, page 138–142, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400705892. doi: 10.1145/3644033.3644374. URL <https://doi.org/10.1145/3644033.3644374>.
- Donna Spencer. *Card Sorting: Designing Usable Categories*. Rosenfeld Media, 2009.
- Chade-Meng Tan, Yuan-Fang Wang, and Chan-Do Lee. The use of bigrams to enhance text categorization. *Information Processing & Management*, 38(4):529–546, 2002. doi: 10.1016/S0306-4573(01)00045-0.
- Vassil Todorov, Frederic Boulanger, and Safouan Taha. Formal verification of automotive embedded software. In *FormaliSE '18: Proceedings of the 6th Conference on Formal Methods in Software Engineering*, page 84 – 87, 2018. doi: 10.1145/3193992.3194003.
- Emanuel Correia de Sousa Dias Trigo. A VS Code plug-in for automated software verification powered by large language models. 2024.
- John F. Tripp and Deborah J. Armstrong. Exploring the relationship between organizational adoption motives and the tailoring of agile methods. In *2014 47th Hawaii International Conference on System Sciences*, pages 4799–4806, 2014. doi: 10.1109/HICSS.2014.589.

- Zhiyuan Wan, David Lo, Xin Xia, and Liang Cai. Bug characteristics in blockchain systems: A large-scale empirical study. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 413–424, 2017. doi: 10.1109/MSR.2017.59.
- Neil White, Stuart Matthews, and Roderick Chapman. Formal verification: Will the seedling ever flower? *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 375(2104), 2017. doi: 10.1098/rsta.2015.0402.
- Xin-Li Yang, David Lo, Xin Xia, Zhi-Yuan Wan, and Jian-Ling Sun. What Security Questions Do Developers Ask? A Large-Scale Study of Stack Overflow Posts. *Journal of Computer Science and Technology*, 31(5):910–924, September 2016. doi: 10.1007/s11390-016-1672-0.

Appendix A

Details about Stack Overflow Question Analysis

Table A.1: Mapping between Verification-Aware language names and StackExchange tag names

Language Name	Tag Name
Bandera	bandera
Dafny	dafny
Escher C Verifier	escher
Eiffel	eiffel
Frama-C	frama-c
Java PathFinder	pathfinder
JML	jml
mbeddr	mbeddr
Spec#	spec#
Spark Ada	spark-ada
VeriFast	verifast
VCC	vcc
Whiley	whiley
Why3	why3

Table A.2: List of removed domain-specific words

Removed Words			
acsl	actually	ada	afterwards
also	alternatively	always	answer
anyone	bulls	c	com
coq	could	cows	dafny
dropboxusercontent	eduardo	eiffel	etc
even	exam	firstly	follow
frama	heavily	help!	however
html	ide	inside	instead
java	jml	js	json
jstestpackage	like	lot	might
must	need	non	one
openssl	please	question	second
seem	simple	simulink	since
someone	something	spark	sure
syntax	tell	thank	thanks
things	true	two	understand
way	wonder	work	would

Appendix B

First Model Training Tags by Topics

Table B.1: Frequency of each tag regarding topic 0

Tag Name	Number of Questions
not-suited	5
proof-strategy	4
understanding-behaviour	3
contract-design	2
lacking-documentation	2
verifier-bug	2
dafny-memory	1
heap-understanding	1
logic-error	1
lack-feature-knowledge	1
array-relative-order	1
loop-invariant	1
usability-issues	1

Table B.2: Frequency of each tag regarding topic 1

Tag Name	Number of Questions
not-suited	10
configuration-hardship	4
understand-failures	2
lacking-documentation	2
lack-feature-knowledge	2
contract-design	1
programing-error	1
usability-issues	1
verifier-bug	1

Table B.3: Frequency of each tag regarding topic 2

Tag Name	Number of Questions
not-suited	11
configuration-hardship	2
loop-invariant	2
proof-strategy	2
lack-feature-knowledge	2
understand-failures	1
frama-c-memory	1
heap-understanding	1
lacking-documentation	1
contract-design	1

Table B.4: Frequency of each tag regarding topic 3

Tag Name	Number of Questions
not-suited	7
understanding-behaviour	3
contract-design	3
proof-strategy	3
lacking-documentation	2
understand-failures	1
behaviour-inconsistency	1
performance-issue	1
verifier-bug	1
configuration-hardship	1
usability-issues	1
lack-feature-knowledge	1

Table B.5: Frequency of each tag regarding topic 4

Tag Name	Number of Questions
not-suited	5
understanding-behaviour	5
proof-strategy	3
lack-feature-knowledge	3
configuration-hardship	1
understand-failures	1
lacking-documentation	1
loop-invariant	1
contract-design	1
logic-error	1

Table B.6: Frequency of each tag regarding topic 5

Tag Name	Number of Questions
not-suited	11
proof-strategy	2
understand-failures	2
understanding-behaviour	1
dafny-memory	1
heap-understanding	1
verifier-bug	1
floating-point-arithmetic	1
performance-issue	1
programing-error	1
lacking-documentation	1
lack-feature-knowledge	1
logic-error	1

Table B.7: Frequency of each tag regarding topic 6

Tag Name	Number of Questions
not-suited	6
lack-feature-knowledge	5
configuration-hardship	2
lacking-documentation	2
proof-strategy	2
understanding-behaviour	2
loop-invariant	2
dafny-memory	1
heap-understanding	1
understand-failures	1
contract-design	1
logic-error	1

Table B.8: Frequency of each tag regarding topic 7

Tag Name	Number of Questions
proof-strategy	8
not-suited	4
lack-feature-knowledge	3
loop-invariant	3
configuration-hardship	2
behaviour-inconsistency	2
dafny-memory	2
heap-understanding	2
understand-failures	1
contract-design	1
performance-issue	1
lacking-documentation	1

Table B.9: Frequency of each tag regarding topic 8

Tag Name	Number of Questions
not-suited	8
logic-error	4
proof-strategy	4
dafny-memory	2
heap-understanding	2
understand-failures	2
lack-feature-knowledge	1
configuration-hardship	1
programing-error	1
loop-invariant	1
understanding-behaviour	1
performance-issue	1

Table B.10: Frequency of each tag regarding topic 9

Tag Name	Number of Questions
not-suited	12
understand-failures	4
lack-feature-knowledge	3
proof-strategy	3
lacking-documentation	2
understanding-behaviour	1
behaviour-inconsistency	1

Table B.11: Frequency of each tag regarding topic 10

Tag Name	Number of Questions
not-suited	9
lack-feature-knowledge	7
proof-strategy	3
configuration-hardship	3
verifier-bug	2
loop-invariant	1
behaviour-inconsistency	1
performance-issue	1
contract-design	1
floating-point-arithmetic	1
dafny-memory	1
heap-understanding	1

Table B.12: Frequency of each tag regarding topic 11

Tag Name	Number of Questions
not-suited	6
dafny-memory	6
heap-understanding	6
lack-feature-knowledge	5
proof-strategy	3
understand-failures	2
logic-error	2
contract-design	1
behaviour-inconsistency	1
understanding-behaviour	1
configuration-hardship	1

Table B.13: Frequency of each tag regarding topic 12

Tag Name	Number of Questions
not-suited	7
understanding-behaviour	6
proof-strategy	3
verifier-bug	1
lack-feature-knowledge	1
loop-invariant	1
contract-design	1
array-relative-order	1
behaviour-inconsistency	1
logic-error	1

Table B.14: Frequency of each tag regarding topic 13

Tag Name	Number of Questions
not-suited	8
lack-feature-knowledge	3
verifier-bug	3
configuration-hardship	3
proof-strategy	2
understand-failures	2
lacking-documentation	1
contract-design	1
understanding-behaviour	1

Table B.15: Frequency of each tag regarding topic 14

Tag Name	Number of Questions
not-suited	9
lack-feature-knowledge	5
understanding-behaviour	2
proof-strategy	2
behaviour-inconsistency	1
performance-issue	1
configuration-hardship	1
programing-error	1
verifier-bug	1
understand-failures	1
logic-error	1

Table B.16: Frequency of each tag regarding topic 15

Tag Name	Number of Questions
not-suited	4
understanding-behaviour	4
proof-strategy	3
verifier-bug	3
heap-understanding	3
logic-error	2
frama-c-memory	2
programing-error	1
array-relative-order	1
configuration-hardship	1
loop-invariant	1
understand-failures	1
ada-memory	1
lack-feature-knowledge	1

Appendix C

Second Model Training Tags by Topics

Table C.1: Frequency of each tag regarding topic 0

Tag Name	Number of Questions
not-suited	6
heap-understanding	5
dafny-memory	4
understand-failures	3
lacking-documentation	2
proof-strategy	1
contract-design	1
configuration-hardship	1
logic-error	1
verifier-bug	1
lack-feature-knowledge	1
frama-c-memory	1
understanding-behaviour	1

Table C.2: Frequency of each tag regarding topic 1

Tag Name	Number of Questions
lack-feature-knowledge	9
understanding-behaviour	3
not-suited	3
proof-strategy	2
contract-design	1
lacking-documentation	1
understand-failures	1
behaviour-inconsistency	1
frama-c-memory	1
heap-understanding	1
logic-error	1
loop-invariant	1
needs-review	1

Table C.3: Frequency of each tag regarding topic 2

Tag Name	Number of Questions
not-suited	7
understand-failures	3
configuration-hardship	3
lack-feature-knowledge	2
lacking-documentation	2
proof-strategy	2
contract-design	1
logic-error	1
understanding-behaviour	1
verifier-bug	1
floating-point-arithmetic	1
performance-issue	1
usability-issues	1

Table C.4: Frequency of each tag regarding topic 3

Tag Name	Number of Questions
configuration-hardship	6
understand-failures	3
understanding-behaviour	3
proof-strategy	3
lack-feature-knowledge	2
lacking-documentation	2
not-suited	2
usability-issues	1
performance-issue	1
verifier-bug	1
contract-design	1

Table C.5: Frequency of each tag regarding topic 4

Tag Name	Number of Questions
proof-strategy	5
lack-feature-knowledge	5
not-suited	4
contract-design	3
understanding-behaviour	3
understand-failures	2
array-relative-order	1
lacking-documentation	1
dafny-memory	1
heap-understanding	1
logic-error	1
loop-invariant	1

Table C.6: Frequency of each tag regarding topic 5

Tag Name	Number of Questions
configuration-hardship	3
lack-feature-knowledge	2
understanding-behaviour	2
verifier-bug	2
not-suited	2
logic-error	2
usability-issues	2
contract-design	2
proof-strategy	2
programing-error	1
frama-c-memory	1
heap-understanding	1
lacking-documentation	1
floating-point-arithmetic	1

Table C.7: Frequency of each tag regarding topic 6

Tag Name	Number of Questions
proof-strategy	7
lack-feature-knowledge	5
loop-invariant	3
not-suited	3
heap-understanding	2
understand-failures	2
verifier-bug	1
dafny-memory	1
usability-issues	1
performance-issue	1
programing-error	1
configuration-hardship	1
contract-design	1
understanding-behaviour	1
ada-memory	1

Table C.8: Frequency of each tag regarding topic 7

Tag Name	Number of Questions
proof-strategy	8
understand-failures	3
understanding-behaviour	2
configuration-hardship	2
not-suited	2
logic-error	2
loop-invariant	2
lack-feature-knowledge	1
usability-issues	1
programing-error	1
floating-point-arithmetic	1
contract-design	1

Appendix D

Mentioned Questions

To navigate to the following hyperlinks it is necessary to prepend the following URL: <https://stackoverflow.com/questions/>

Question 16

- **Title:** DafnyLanguageServer: Can the function syntax version be configured?
- **URL:** [77306575/dafnylanguageserver-can-the-function-syntax-version-be-configured](https://stackoverflow.com/questions/77306575/dafnylanguageserver-can-the-function-syntax-version-be-configured)
- **Last Accessed:** 9th of July 2024

Question 24

- **Title:** How to prove properties of non-linear operators involved in LEB128 algorithm in Dafny?
- **URL:** [77593925/how-to-prove-properties-of-non-linear-operators-involved-in-leb128-algorithm-in](https://stackoverflow.com/questions/77593925/how-to-prove-properties-of-non-linear-operators-involved-in-leb128-algorithm-in)
- **Last Accessed:** 9th of July 2024

Question 30

- **Title:** Proving non-linear traversal terminates in Dafny
- **URL:** [77480353/proving-non-linear-traversal-terminates-in-dafny](https://stackoverflow.com/questions/77480353/proving-non-linear-traversal-terminates-in-dafny)
- **Last Accessed:** 9th of July 2024

Question 119

- **Title:** Why does this Dafny function return an empty sequence?
- **URL:** [74834633/why-does-this-dafny-function-return-an-empty-sequence](#)
- **Last Accessed:** 16th of July 2024

Question 123

- **Title:** Defining a surjective predicate for maps and functions
- **URL:** [74632153/defining-a-surjective-predicate-for-maps-and-functions](#)
- **Last Accessed:** 9th of July 2024

Question 135

- **Title:** Proving a covariance inequality in Dafny, use contradiction?
- **URL:** [74156826/proving-a-covariance-inequality-in-dafny-use-contradiction](#)
- **Last Accessed:** 9th of July 2024

Question 138

- **Title:** Cannot prove in Dafny that $\text{variance}(x) \geq 0$
- **URL:** [74220812/cannot-prove-in-dafny-that-variancex-0](#)
- **Last Accessed:** 9th of July 2024

Question 179

- **Title:** How do I describe the invariants for this simple algorithm in Dafny?
- **URL:** [72438846/how-do-i-describe-the-invariants-for-this-simple-algorithm-in-dafny](#)
- **Last Accessed:** 9th of July 2024

Question 188

- **Title:** Can I allow preconditions on the argument to a higher-order function in Dafny?
- **URL:** [70468186/can-i-allow-preconditions-on-the-argument-to-a-higher-order-function-in-dafny](#)
- **Last Accessed:** 9th of July 2024

Question 221

- **Title:** Dafny prove lemmas in a high-order polymorphic function
- **URL:** [67860852/dafny-prove-lemmas-in-a-high-order-polymorphic-function](#)
- **Last Accessed:** 9th of July 2024

Question 236

- **Title:** Dafny cannot prove function-method equivalence, with High-Order-Polymorphic Recursive vs Linear Iterative
- **URL:** [66563130/dafny-cannot-prove-function-method-equivalence-with-high-order-polymorphic-relu](#)
- **Last Accessed:** 9th of July 2024

Question 243

- **Title:** While loop termination with null references in Dafny linked list implementation
- **URL:** [66172163/while-loop-termination-with-null-references-in-dafny-linked-list-implementation](#)
- **Last Accessed:** 9th of July 2024

Question 250

- **Title:** Understanding the :fuel attribute in Dafny
- **URL:** [65165650/understanding-the-fuel-attribute-in-dafny](#)
- **Last Accessed:** 9th of July 2024

Question 259

- **Title:** Dafny suggests that multiplying odd numbers doesn't always give an odd result
- **URL:** [64634094/dafny-suggests-that-multiplying-odd-numbers-doesnt-always-give-an-odd-result](https://stackoverflow.com/questions/64634094/dafny-suggests-that-multiplying-odd-numbers-doesnt-always-give-an-odd-result)
- **Last Accessed:** 9th of July 2024

Question 302

- **Title:** Dafny Loop Invariant Not Maintained By The Loop
- **URL:** [49504596/dafny-loop-invariant-not-maintained-by-the-loop](https://stackoverflow.com/questions/49504596/dafny-loop-invariant-not-maintained-by-the-loop)
- **Last Accessed:** 9th of July 2024

Question 305

- **Title:** assignment may update an object not in the enclosing context's modifies clause
- **URL:** [62485809/assignment-may-update-an-object-not-in-the-enclosing-contexts-modifies-clause](https://stackoverflow.com/questions/62485809/assignment-may-update-an-object-not-in-the-enclosing-contexts-modifies-clause)
- **Last Accessed:** 9th of July 2024

Question 346

- **Title:** How to modify an array element and a class member in the same method (Dafny)?
- **URL:** [58607417/how-to-modify-an-array-element-and-a-class-member-in-the-same-method-dafny](https://stackoverflow.com/questions/58607417/how-to-modify-an-array-element-and-a-class-member-in-the-same-method-dafny)
- **Last Accessed:** 9th of July 2024

Question 353

- **Title:** Different "sorted" predicates should be equivalent in Dafny
- **URL:** [48331442/different-sorted-predicates-should-be-equivalent-in-dafny](https://stackoverflow.com/questions/48331442/different-sorted-predicates-should-be-equivalent-in-dafny)
- **Last Accessed:** 9th of July 2024

Question 396

- **Title:** How to read dafny counterexamples
- **URL:** [52618673/how-to-read-dafny-counterexamples](#)
- **Last Accessed:** 16th of July 2024

Question 441

- **Title:** trying witness null: result of operation might violate subset type constraint
- **URL:** [49041017/trying-witness-null-result-of-operation-might-violate-subset-type-constraint](#)
- **Last Accessed:** 9th of July 2024

Question 444

- **Title:** What does code outside a Dafny class verify, but not when encapsulated?
- **URL:** [49033664/what-does-code-outside-a-dafny-class-verify-but-not-when-encapsulated](#)
- **Last Accessed:** 9th of July 2024

Question 451

- **Title:** Dafny: Using "forall" quantifiers with the "reads" or "modifies" clauses
- **URL:** [48429725/dafny-using-forall-quantifiers-with-the-reads-or-modifies-clauses](#)
- **Last Accessed:** 9th of July 2024

Question 452

- **Title:** How can I modify an overwritten array correctly?
- **URL:** [48034153/how-can-i-modify-an-overwritten-array-correctly](#)
- **Last Accessed:** 9th of July 2024

Question 476

- **Title:** Dafny method return reversed character array
- **URL:** [44094115/dafny-method-return-reversed-character-array](#)
- **Last Accessed:** 9th of July 2024

Question 489

- **Title:** Dafny context modifies clause error
- **URL:** [43591924/dafny-context-modifies-clause-error](#)
- **Last Accessed:** 9th of July 2024

Question 552

- **Title:** How to compile using EiffelStudio? VD43 Precompiled file is missing or unreadable (...)/driver.exe - External C compilation failed
- **URL:** [65530756/how-to-compile-using-eiffelstudio-vd43-precompiled-file-is-missing-or-unreadabl](#)
- **Last Accessed:** 9th of July 2024

Question 746

- **Title:** Incompatible library version running EiffelStudio
- **URL:** [37127975/incompatible-library-version-running-eiffelstudio](#)
- **Last Accessed:** 9th of July 2024

Question 854

- **Title:** Why is my ACSL contract failing on my copy function?
- **URL:** [73449193/why-is-my-acsl-contract-failing-on-my-copy-function](#)
- **Last Accessed:** 9th of July 2024

Question 938

- **Title:** ACSL Logic Struct Declarations Not Working as in Reference Manual
- **URL:** [59456820/acsl-logic-struct-declarations-not-working-as-in-reference-manual](#)
- **Last Accessed:** 9th of July 2024

Question 963

- **Title:** Coq file generated by WP does not compile
- **URL:** [55066369/coq-file-generated-by-wp-does-not-compile](#)
- **Last Accessed:** 9th of July 2024

Question 973

- **Title:** Specifying Referential transparency in ACSL
- **URL:** [53257844/specifying-referential-transparency-in-acsl](#)
- **Last Accessed:** 9th of July 2024

Question 994

- **Title:** Cannot prove euclidean division in frama-c
- **URL:** [50967169/cannot-prove-euclidean-division-in-frama-c](#)
- **Last Accessed:** 9th of July 2024

Question 995

- **Title:** Frama-C multiline macro definition syntax error
- **URL:** [50957918/frama-c-multiline-macro-definition-syntax-error](#)
- **Last Accessed:** 9th of July 2024

Question 1003

- **Title:** E-ACSL logic function call error - Unbound function
- **URL:** [50179575/e-acsl-logic-function-call-error-unbound-function](#)
- **Last Accessed:** 9th of July 2024

Question 1037

- **Title:** How to prove an iterative loop with computations in frama-c wp?
- **URL:** [45915333/how-to-prove-an-iterative-loop-with-computations-in-frama-c-wp](#)
- **Last Accessed:** 9th of July 2024

Question 1053

- **Title:** ACSL "assigns" annotation for inner structs and fields of C code
- **URL:** [43648733/acsl-assigns-annotation-for-inner-structs-and-fields-of-c-code](#)
- **Last Accessed:** 9th of July 2024

Question 1111

- **Title:** The exact mechanism of mapping WhyML into SMT logic
- **URL:** [31426510/the-exact-mechanism-of-mapping-whyml-into-smt-logic](#)
- **Last Accessed:** 9th of July 2024

Question 1129

- **Title:** ACSL specification of a function that appends a string to a dynamic character array
- **URL:** [33682909/acsl-specification-of-a-function-that-appends-a-string-to-a-dynamic-character-array](#)
- **Last Accessed:** 9th of July 2024

Question 1144

- **Title:** trying to make frama-c work on Windows 7, using Perl or MinGW or
- **URL:** [32012175/trying-to-make-frama-c-work-on-windows-7-using-perl-or-mingw-or](#)
- **Last Accessed:** 9th of July 2024

Question 1327

- **Title:** How to install Microsoft Spec# 2010
- **URL:** [5309173/how-to-install-microsoft-spec-2010](#)
- **Last Accessed:** 9th of July 2024

Question 1359

- **Title:** Failed assert that libsparkcrypto SHA256 results are equal
- **URL:** [66279968/failed-assert-that-libsparkcrypto-sha256-results-are-equal](#)
- **Last Accessed:** 9th of July 2024

Question 1374

- **Title:** How to perform arithmetic contract operations on function taking in 2D array type as parameter in Ada
- **URL:** [56956796/how-to-perform-arithmetic-contract-operations-on-function-taking-in-2d-array-typ](#)
- **Last Accessed:** 9th of July 2024