

Taming the Evolving Black Box:
Towards Improved Integration and Documentation of
Intelligent Web Services

Alex Cummaudo
BSc Swinburne, BIT(Hons)
<ca@deakin.edu.au>

A thesis submitted in partial fulfilment of the requirements for the
Doctor of Philosophy



Applied Artificial Intelligence Institute
Deakin University
Melbourne, Australia

June 2, 2020

Abstract

Application developers are eager to integrate machine learning (ML) into their software, with a plethora of vendors providing pre-packaged components—typically under the artificial intelligence (AI) banner—to entice them. Such components are marketed as developer ‘friendly’ ML and easy for them to integrate (being ‘just another’ component added to their toolchain). These components are, however, non-trivial: in particular, developers unknowingly add the risk of mixing non-deterministic ML behaviour into their applications that, in turn, impact the quality of their software. Prior research advocates that a developer’s conceptual understanding is critical to effective interpretation of reusable components. However, these ready-made AI components do not present sufficient detail to allow developers to acquire this conceptual understanding. In this thesis, we address the clash of mindsets between an application developers’ deterministic approach to software development and the mindset needed to incorporate probabilistic, intelligent components, namely cloud-based intelligent web services. (We scope our investigation to the most mature subset of these services: those that provide computer vision intelligence via RESTful web APIs.) We explore how this mindset clash ultimately impacts the reliability of the software developers produce and these concerns are addressed via four research perspectives answered by seven sub-research questions.

Firstly, we present a landscape analysis of the nature of these cloud-based intelligent components. What is their runtime behaviour and evolution profile? We performed periodic structured observations against three prominent computer vision services over an 11 month longitudinal study and found inconsistencies in how these services behave and substantial evolution risk in the labels and confidence scores they present. This study is further explored in Chapter 4. Secondly, we explore the sufficiency of the documentation presented in these services. Which attributes of ‘sufficient’ API documentation is explored by literature, what is the efficacy of these attributes according to developers, and which of these attributes are currently missing from these services? We perform a triangulation study by (i) synthesising a taxonomy of 34 facets a ‘complete’ API document should sourced via a systematic mapping study resulting in 21 seminal academic studies, (ii) assessing

the efficacy of this taxonomy using a survey loosely based on the system usability scale instrument that was distributed to 83 software engineers, and (iii) applying the taxonomy (as assessed by engineers) against three popular computer vision services to produce 12 suggested improvements to the service documentation. This study is further explored in Chapter 8. Thirdly, we analyse whether these services are more misunderstood than conventional software engineering domains. Which types of issues do developers face most when working with intelligent services, and which of these issues are developers most frustrated with? Is the distribution of these issues different to more established software development domains? We conduct a mining study of the popular software development discussion forum Stack Overflow resulting in from 1,825 posts to analyse the various pain-points developers face, as classified from two existing classification strategies from the literature. We find that developers raise issues due to a primitive understanding of the underlying concepts within these services—which results in conceptual misunderstanding and confusion in interpreting service errors—and the distribution of the types of frustrations raised are substantially different to more mature domains. This study is further explored in Chapters 5 and 6. Lastly, we propose several strategies targeted at better integrating intelligent components into developer’s software whilst preserving robustness. These strategies are outlined within Chapters 9 to 11.

This thesis presents a substantial contribution to the software engineering discipline by presenting a better understanding how AI-based components have non-trivial implications to software reliability, robustness and completeness which arise due to developer misunderstandings. Further, we present a novel service integration architecture by which developers can use to integrate their applications with these AI-based components to reduce any potential risk to the quality of their software. Lastly, this thesis contributes a key list of attributes that should be documented with any API, chiefly to assist service providers on how better to document their services.

Candidate Declaration

I certify that the thesis entitled "*Taming the Evolving Black Box: Towards Improved Integration and Documentation of Intelligent Web Services*" submitted for the degree of Doctor of Philosophy complies with all statements below.

- (i) I am the creator of all or part of the whole work(s) (including content and layout) and that where reference is made to the work of others, due acknowledgement is given.
- (ii) The work(s) are not in any way a violation or infringement of any copyright, trademark, patent, or other rights whatsoever of any person.
- (iii) That if the work(s) have been commissioned, sponsored or supported by any organisation, I have fulfilled all of the obligations required by such contract or agreement.
- (iv) That any material in the thesis which has been accepted for a degree or diploma by any university or institution is identified in the text.
- (v) All research integrity requirements have been complied with.

I certify that I am the student named below and that the information provided above is correct.

Alex Cummaudo
June 2, 2020

Access to Thesis

I am the author of the thesis entitled “*Taming the Evolving Black Box: Towards Improved Integration and Documentation of Intelligent Web Services*” submitted for the degree of Doctor of Philosophy.

This thesis may be made available for consultation, loan and limited copying in accordance with the *Copyright Act 1968 (Cth)*.

I certify that I am the student named below and that the information provided above is correct.

Alex Cummaudo
June 2, 2020

To my family, friends, and teachers.

Acknowledgements

A long journey of 20 years education has led me to this thesis, and there are so many people who've helped me get to this point that deserve thanking. To start, I must thank my family; you have always been there for me in times good and bad. I'm especially grateful to my mother, Rosa, my father, Paul, my siblings, Marc and Lisa, and my nonna, Michelina, for your love and support. I also thank my nieces Nina and Lucy (though too young to read this now!) for bringing us all so much joy in these last three years. I thank all my teachers, particularly Natalie Heath, whose hard efforts paid off in my tertiary education, and, of course, all those who assisted me along and help shape this PhD. Firstly, to Professor Rajesh Vasa, thank you for your many revisions, patience, ideas and efforts to shape this work: your years of phenomenal guidance—both as a supervisor and as a mentor—has reshaped my worldview to far wider perspectives and I now approach thought and problem-solving in a remarkably new light. Secondly, I thank Professor John Grundy for your efforts and for being such an approachable and hard-working supervisor, always willing to provide feedback and guidance, and help me get over the finish line. I also thank Dr Scott Barnett for the many fruitful discussions shared, for the interest you have shown in my work, and the joint collaborations we achieved in the last two years; Associate Professor Mohamed Abdelrazek for your help in shaping many of the works within this thesis; and, lastly, Associate Professor Andrew Cain, who not only taught me the realm of programming back in undergraduate days, but who first suggested a PhD was within my reach, of which I had never fathomed. I must thank everyone at the Applied Artificial Intelligence Institute for creating such an enjoyable environment to work in, especially Jake Renzella, Rodney Pilgrim, Mahdi Babaei, and Reuben Wilson for their friendship over these years and for all the coffee runs, conversations, and ideas shared. And, lastly, to Tom Fellowes: thank you for being by my side throughout this journey.

This chapter is now over, the next chapter awaits...

— Alex Cummaudo
June 2, 2020

Contents

Abstract	iii
Candidate Declaration	v
Access to Thesis	vii
Acknowledgements	xi
Contents	xi
List of Publications	xviii
List of Abbreviations	xix
List of Figures	xxiii
List of Tables	xxvii
List of Listings	xxx
I Preface	1
1 Introduction	3
1.1 Research Context	4
1.2 Motivating Scenarios	7
1.3 Research Motivation	12
1.4 Research Goals	14
1.5 Research Methodology	16
1.6 Thesis Organisation	16
1.6.1 Part I: Preface	17
1.6.2 Part II: Publications	17

1.6.3	Part III: Postface	20
1.6.4	Part IV: Appendices	20
1.7	Research Contributions	20
1.7.1	Contribution 1: Landscape Analysis & Preliminary Solutions	21
1.7.2	Contribution 2: Improving Documentation Attributes	22
1.7.3	Contribution 3: Service Integration Architecture	23
2	Background	25
2.1	Software Quality	26
2.1.1	Validation and Verification	27
2.1.2	Quality Attributes and Models	29
2.1.3	Reliability in Computer Vision	31
2.2	Probabilistic and Nondeterministic Systems	33
2.2.1	Interpreting the Uninterpretable	34
2.2.2	Explanation and Communication	35
2.2.3	Mechanics of Model Interpretation	36
2.3	Application Programming Interfaces	36
2.3.1	API Usability	37
3	Research Methodology	39
3.1	Research Questions Revisited	39
3.1.1	Empirical Research Questions	40
3.1.2	Non-Empirical Research Questions	41
3.2	Philosophical Stances	41
3.3	Research Methods	43
3.3.1	Review of Relevant Research Methods	43
3.3.2	Review of Data Collection Techniques for Field Studies . . .	45
3.4	Research Design	45
3.4.1	Landscape Analysis of Computer Vision Services	45
3.4.2	Utility of API Documentation in Computer Vision Services	47
3.4.3	Developer Issues concerning Computer Vision Services . .	47
3.4.4	Designing Improved Integration Strategies	48
II	Publications	49
4	Identifying Evolution in Computer Vision Services	51
4.1	Introduction	51
4.2	Motivating Example	53
4.3	Related Work	54
4.3.1	External Quality	54
4.3.2	Internal Quality	55
4.4	Method	57
4.5	Findings	59
4.5.1	Consistency of top labels	59
4.5.2	Consistency of confidence	62

4.5.3	Evolution risk	62
4.6	Recommendations	64
4.6.1	Recommendations for IWS users	64
4.6.2	Recommendations for IWS providers	65
4.7	Threats to Validity	67
4.7.1	Internal Validity	67
4.7.2	External Validity	67
4.7.3	Construct Validity	68
4.8	Conclusions & Future Work	68
5	Interpreting Pain-Points in Computer Vision Services	71
5.1	Introduction	71
5.2	Motivation	73
5.3	Background	75
5.4	Method	76
5.4.1	Data Extraction	76
5.4.2	Data Filtering	78
5.4.3	Data Analysis	79
5.5	Findings	81
5.5.1	Post classification and reliability analysis	81
5.5.2	Developer Frustrations	82
5.5.3	Statistical Distribution Analysis	84
5.6	Discussion	84
5.6.1	Answers to Research Questions	84
5.6.2	The Developer’s Learning Approach	86
5.6.3	Implications	88
5.7	Threats to Validity	91
5.7.1	Internal Validity	91
5.7.2	External Validity	91
5.7.3	Construct Validity	92
5.8	Conclusions	92
6	Ranking Computer Vision Service Issues using Emotion	93
6.1	Introduction	93
6.2	Emotion Mining from Text	95
6.3	Methodology	95
6.3.1	Data Set Extraction from Stack Overflow	96
6.3.2	Question Type & Emotion Classification	98
6.4	Findings	100
6.5	Discussion	101
6.6	Threats to Validity	103
6.6.1	Internal Validity	103
6.6.2	External Validity	103
6.6.3	Construct Validity	104
6.7	Conclusions	104

7 Lessons learnt using a pre-trained AI model	105
7.1 Introduction	105
7.2 Case Study	106
7.2.1 Scope	107
7.2.2 Method	107
7.2.3 Results	107
7.3 Findings and Discussion	108
7.3.1 Limitations of the Text Classifier	108
7.3.2 Peering into the Training Dataset	110
7.3.3 New tools: Model Cards and Datasheets	112
7.3.4 Threats to validity	112
7.4 Conclusion	113
8 Better Documenting Computer Vision Services	115
8.1 Introduction	115
8.2 Related Work	118
8.2.1 API Usability and Documentation Knowledge	118
8.2.2 Adapting the System Usability Scale	119
8.2.3 Computer Vision Services	119
8.3 Taxonomy Development	119
8.3.1 Systematic Mapping Study	120
8.3.2 Development of the Taxonomy	124
8.4 API Documentation Knowledge Taxonomy	126
8.5 Validating our Taxonomy	129
8.5.1 Survey Study	129
8.5.2 Empirical application of the taxonomy against Computer Vision Services	130
8.6 Taxonomy Analysis	131
8.6.1 In-Literature Scores for Taxonomy Categories	131
8.6.2 In-Practice Scores for Taxonomy Categories	132
8.6.3 Contrasting In-Literature to In-Practice Scores	133
8.6.4 Triangulating ILS and IPS with Computer Vision	134
8.6.5 Areas of Improvement for CVS Documentation	135
8.7 Threats to Validity	139
8.7.1 Internal Validity	139
8.7.2 External Validity	139
8.7.3 Construct Validity	140
8.8 Conclusions & Future Work	141
9 Using a Facade Pattern to combine Computer Vision Services	143
9.1 Introduction	143
9.1.1 Motivating Scenario: Intelligent vs Traditional Web Services	144
9.1.2 Research Motivation	145
9.2 Merging API Responses	145
9.2.1 API Facade Pattern	146

9.2.2	Merge Operations	146
9.2.3	Merging Operators for Labels	147
9.3	Graph of Labels	147
9.3.1	Labels and synsets	148
9.3.2	Connected Components	148
9.4	API Results Merging Algorithm	151
9.4.1	Mapping Labels to Synsets	151
9.4.2	Deciding Total Number of Labels	151
9.4.3	Allocating Number of Labels to Connected Components	151
9.4.4	Selecting Labels from Connected Components	153
9.4.5	Conformance to properties	153
9.5	Evaluation	153
9.5.1	Evaluation Method	153
9.5.2	Naive Operators	154
9.5.3	Traditional Proportional Representation Operators	156
9.5.4	New Proposed Label Merge Technique	156
9.5.5	Performance	156
9.6	Conclusions and Future Work	157
10	Supporting Safe Usage of Intelligent Web Services	159
10.1	Introduction	159
10.2	Motivating Example	162
10.3	Threshy	164
10.4	Related work	165
10.4.1	Decision Boundary Estimation	165
10.4.2	Tooling for ML Frameworks	166
10.5	Conclusions & Future Work	166
11	An Integration Architecture Tactic to Guard AI-first Components	169
11.1	Introduction	169
11.2	Motivating Example	172
11.3	Intelligent Services	173
11.3.1	‘Intelligent’ vs ‘Traditional’ Web Services	173
11.3.2	Dimensions of Evolution	173
11.3.3	Limited Configurability	174
11.4	Our Approach	177
11.4.1	Core Components	177
11.4.2	Usage Example	182
11.5	Evaluation	183
11.5.1	Data Collection and Preparation	184
11.5.2	Results	184
11.5.3	Threats to Validity	186
11.6	Discussion	190
11.6.1	Implications	190
11.6.2	Limitations	190

11.6.3 Future Work	191
11.7 Related Work	192
11.8 Conclusions	193
III Postface	195
12 Conclusions & Future Work	197
12.1 Contributions of this Work	197
12.1.1 Answers to Research Questions	198
12.1.2 Limitations to Research Answers & Future Research	202
12.2 Concluding Remarks	204
References	226
List of Online Artefacts	227
IV Appendices	231
A Additional Materials	233
A.1 Development, Documentation and Usage of Web APIs	235
A.2 Additional Figures	238
B Reference Architecture Source Code	253
C Supplementary Materials to Chapter 8	287
C.1 Detailed Overview of Our Proposed Taxonomy	289
C.2 Sources of Documentation	293
C.3 List of Primary Sources	296
C.4 Survey Questions	298
D Authorship Statements	303
E Ethics Clearance	339

List of Publications

Below lists publications arising from work completed in this PhD.

1. A. Cummaudo, R. Vasa, J. Grundy, M. Abdelrazek, and A. Cain, “Losing Confidence in Quality: Unspoken Evolution of Computer Vision Services,” in *Proceedings of the 35th IEEE International Conference on Software Maintenance and Evolution*. Cleveland, OH, USA: IEEE, December 2019. DOI 10.1109/ICSME.2019.00051. ISBN 978-1-72-813094-1 pp. 333–342
2. A. Cummaudo, R. Vasa, and J. Grundy, “What should I document? A preliminary systematic mapping study into API documentation knowledge,” in *Proceedings of the 13th International Symposium on Empirical Software Engineering and Measurement*. Porto de Galinhas, Recife, Brazil: IEEE, October 2019. DOI 10.1109/ESEM.2019.8870148. ISBN 978-1-72-812968-6. ISSN 1949-3789 pp. 1–6
3. A. Cummaudo, R. Vasa, S. Barnett, J. Grundy, and M. Abdelrazek, “Interpreting Cloud Computer Vision Pain-Points: A Mining Study of Stack Overflow,” in *Proceedings of the 42nd International Conference on Software Engineering*. Seoul, Republic of Korea: ACM, July 2020, In Press
4. A. Cummaudo, S. Barnett, R. Vasa, J. Grundy, and M. Abdelrazek, “Beware the evolving ‘intelligent’ web service! An integration architecture tactic to guard AI-first components,” in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. Sacramento, CA, USA: ACM, November 2020, In Press
5. T. Ohtake, A. Cummaudo, M. Abdelrazek, R. Vasa, and J. Grundy, “Merging intelligent API responses using a proportional representation approach,” in *Proceedings of the 19th International Conference on Web Engineering*. Daejeon, Republic of Korea: Springer, June 2019. DOI 10.1007/978-3-030-19274-7_28. ISBN 978-3-03-019273-0. ISSN 1611-3349 pp. 391–406

List of Abbreviations

A²I² Applied Artificial Intelligence Institute. 45, 47

AI artificial intelligence. xvi, 3–5, 8, 12–14, 34, 35, 51, 52, 55, 56, 66, 69, 71–73, 75, 76, 86, 89, 90, 92, 93, 105, 106, 108, 110–113, 145, 146, 169, 170, 173, 186, 193, 200, 202, 204, 238, 240

API application programming interface. xxii, 4–6, 8–16, 18, 19, 22, 23, 25, 26, 28, 29, 36–38, 40–42, 44, 47, 52, 53, 56, 58, 65–68, 71–78, 81–94, 97, 98, 100–103, 105, 107, 109, 112, 115–120, 122–129, 132–141, 143–146, 148, 150, 151, 153, 154, 157, 160, 169, 171, 173, 176–178, 184, 190, 193, 197, 199–201, 235, 237

AWS Amazon web services. 57, 60

BYOML Build Your Own Machine Learning. 5, 6

CC connected component. 148, 151–153, 156

CDSS clinical decision support system. 7, 10

CNN convolutional neural network. 10, 11, 33, 54

CRUD create, read, update, and delete. 237

CVS computer vision service. 7–10, 12, 14–21, 23–25, 27, 28, 31, 37, 40–42, 44, 45, 47, 51–57, 60, 65, 67, 68, 71, 73, 78, 83, 87, 91–93, 95–97, 100, 103, 106, 107, 112, 115–117, 119, 126, 128, 130, 131, 134–141, 143, 145–147, 151, 153, 157, 170, 171, 174, 176, 177, 184, 186, 191, 193, 197–202, 238, 241

DCE distributed computing environment. 235

HITL human-in-the-loop. 11

HTTP Hypertext Transfer Protocol. 6, 181, 182, 185, 190, 235–237

IDL interface definition language. 235, 237

IRR inter-rater reliability. 91

IWS intelligent web service. 5–7, 9–12, 14, 15, 17–20, 25–28, 31, 33, 36, 38, 51–53, 55, 56, 64, 66, 68, 69, 71–77, 79, 81, 82, 84–94, 97, 98, 103, 105, 106, 113, 115, 116, 141, 143–145, 157, 159, 169, 170, 172, 173, 176–179, 190–193, 197, 202–204, 238

JSON JavaScript Object Notation. 7, 174, 182, 184

ML machine learning. 3–6, 8, 9, 12, 13, 15, 18, 22, 29, 34, 37, 51, 52, 55, 56, 66, 68, 72–74, 76, 77, 89, 90, 103, 143, 144, 157, 159

NN neural network. 12, 32, 34, 36

PaaS Platform as a Service. 7, 11, 55

QoS quality of service. 55, 56, 235

RAML RESTful API Modeling Language. 237

REST REpresentational State Transfer. 5, 52, 71, 92, 144, 169, 193, 236, 237

ROI region of interest. 10, 11

RPC remote procedure call. 235

SDK software development kit. 53, 120, 135

SLA service-level agreement. 55, 235

SMS systematic mapping study. 19, 20, 23, 117–120, 125, 131, 140, 141

SO Stack Overflow. 5, 15, 17, 23, 40, 41, 44, 47, 48, 56, 57, 71, 73–77, 79, 81, 82, 84–87, 89–96, 98–101, 103–107, 110, 111, 113, 200, 241

SOA service-oriented architecture. 235

SOAP Simple Object Access Protocol. 5, 235–237

SOLO Structure of the Observed Learning Outcome. 86–88, 90, 91

SQA service quality assurance. 53, 54

SQuaRE Systems and software Quality Requirements and Evaluation. 30

SUS System Usability Scale. 19, 116, 117, 119, 129, 130, 132, 140

SVM support vector machine. 34, 36

URI uniform resource identifier. 237

V&V verification & validation. 25–29

WADL Web Application Description Language. 237

WSDL Web Services Description Language. 235

XML eXtendable markup language. 7, 235

List of Figures

1.1	Differences between data- and rule-driven cloud services	4
1.2	The spectrum of machine learning	5
1.3	Overview of intelligent web services	7
1.4	CancerAssist Context Diagram	11
1.5	Overview publication coherency	22
2.1	Mindset clashes within the development, use and nature of a IWS . .	26
2.2	Leakage of internal and external quality in	29
2.3	Overview of software quality models	30
2.4	Adversarial examples in computer vision	32
2.5	Deterministic versus nondeterministic systems	33
2.6	Theory of AI communication	36
3.1	Review of field study techniques	46
4.1	Consistency of labels in computer vision services is rare	59
4.2	Top labels for images between computer vision services do not intersect	60
4.3	Computer vision services can return multiple top labels	61
4.4	Cumulative distribution of top label confidences	63
4.5	Cumulative distribution of intersecting top label confidences	63
4.6	Agreement of labels between multiple computer vision services do not share similar confidences	64
5.1	Traits of intelligent web services compared to DIY ML	74
5.2	Trend of Stack Overflow posts discussing computer vision services .	75
5.3	Comparing documentation-specific and generalised classifications of Stack Overflow posts	82
5.4	Alignment of Bloom and SOLO taxonomies against computer vision issues	88
6.1	Distribution of Stack Overflow question types	100

6.2 Proportion of emotions per question type	101
7.1 Emotion classifier training data imbalance	110
8.1 Systematic mapping study search results, by years	121
8.2 Filtering steps used in the systematic mapping study	121
8.3 A systematic map of API documentation knowledge studies	125
8.4 Our proposed API documentation knowledge taxonomy	127
8.5 Comparison of ILS and IPS values	133
8.6 Comparison of ILS and IPS values	134
9.1 Overview of the proposed facade	146
9.2 Graph of associated synsets against two different endpoints	149
9.3 Label counts per API assessed	150
9.4 Connected components vs. images	150
9.5 Allocation to connected components	152
9.6 F-measure comparison	154
10.1 Example case study of evaluating model performance in two different models	160
10.2 Request and response for computer vision services provide limited configurability	160
10.3 Threshy supports threshold selection and monitoring	162
10.4 Example pipeline of a computer vision system	163
10.5 UI workflow of Threshy	164
10.6 Architecture of Threshy	167
11.1 Prominent computer vision service evolution	171
11.2 Dimensions of evolution within computer vision services	174
11.3 Example of substantial confidence change	174
11.4 Directly versus indirectly accessing intelligent services	175
11.5 Sample request and response for intelligent services	176
11.6 State diagram of architecture workflows	180
11.7 Precondition failure taxonomy	181
11.8 Histogram of confidence variation	185
11.9 Architecture response to substantial confidence evolution	187
11.10 Architecture response to label set evolution	188
11.11 Architecture response of expected label mismatch	189
12.1 Results from computer vision services can be disparate and non-static	199
12.2 Distribution of issues on Stack Overflow	201
A.1 SOAP versus REST search interest over time	236
A.2 Categorisation of AI-based products and services	240
A.3 Increasing interest in the developer community of computer vision services	241

A.4	Causal factors that may influence understanding of intelligent web services	242
A.5	A proposal technical domain model for intelligent services	243
A.6	Potential questions that can be asked around causal factors of a developer's understanding of an intelligent service	244
A.7	Threshy and developer interaction with decision boundaries	244
A.8	Threshy domain model	245
A.9	Threshy sequence diagram	245
A.10	High-level overview of our method in Chapter 5	246
A.11	Class diagram of architecture implementation	247
A.12	Creation of a benchmark using the architecture tactic	248
A.13	Making a request via the proxy server facade	249
A.14	High-level workflow of the architectural tactic	250
A.15	Handling of evolution using our architecture (i)	251
A.16	Handling of evolution using our architecture (ii)	252

List of Tables

1.1	Differing characteristics of cloud services	6
1.2	Comparison of the machine learning spectrum	6
1.3	Varying confidence changes over time between three computer vision services	9
1.4	Definitions of ‘confidence’ in CV documentation	10
1.5	List of publications resulting from this thesis	21
3.1	Classification of research questions in this thesis	40
4.1	Characteristics of data in computer vision evolution assessment . . .	58
4.2	Ratio of consistent labels in computer vision services	59
4.3	Evolution of top labels and confidence values	62
5.1	Taxonomies used in our Stack Overflow mining study	80
5.2	Example Stack Overflow posts aligning to Bloom’s and SOLO taxonomies	87
6.1	Our interpretations from a Stack Overflow question type taxonomy .	97
6.2	Frequency of emotions per question type.	100
6.3	Assigning emotion to Stack Overflow questions	102
7.1	Results from Inter-Rater Agreements.	108
7.2	Sample questions comparing EmoTxt to manual classification . . .	109
8.1	Summary of search results in API documentation knowledge	122
8.2	Data extraction in API documentation knowledge study	124
8.3	Weighted ILS Scoring	132
8.4	Weighted IPS Scoring	132
8.5	Labels assigned to ILS and IPS values	133
9.1	Statistics for the number of labels	148

9.2	First allocation iteration	153
9.3	Second allocation iteration	153
9.4	Third allocation iteration	153
9.5	Fourth allocation iteration	153
9.6	Matching to human-verified labels	155
9.7	Evaluation results of the facade	155
9.8	Average of evaluation result of the facade	155
11.1	Potential reasons for precondition failure	177
11.2	Rules encoded within behaviour tokens	179
11.3	Variance in ontologies	186

List of Listings

A.1	An example SOAP request	235
A.2	An example SOAP response	236
A.3	An example RESTful request	237
A.4	An example RESTful response	237
B.1	Implementation of the architecture module components	253
B.2	Implementation of the architecture facade API	279

Part I

Preface

CHAPTER 1

3

4

5

6

Introduction

7 Within the last half-decade, we have seen an explosion of cloud-based services
8 typically marketed under an AI banner. Vendors are rapidly pushing out AI-based
9 solutions, technologies and products encapsulating half a century worth of machine-
10 learning research.¹ Application developers are eager to develop the next generation
11 of ‘AI-first’ software, that will reason, sense, think, act, listen, speak and execute
12 every whim in our web browser or smartphone app.

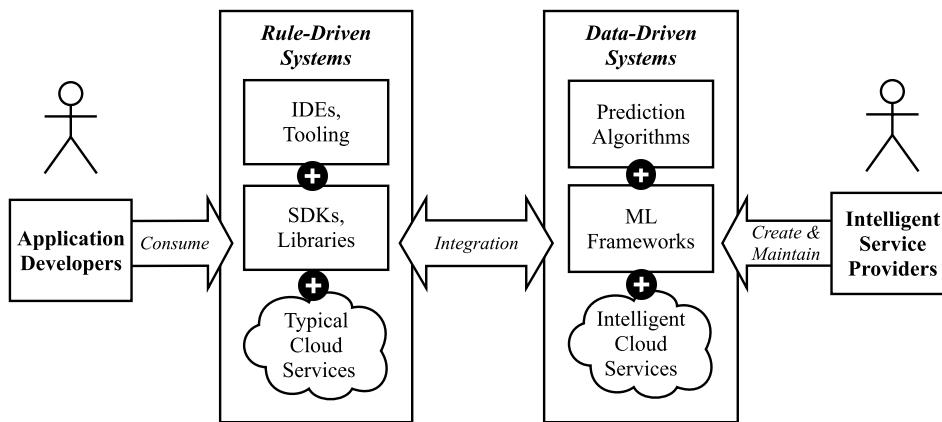
13 However, application developers, accustomed to traditional software engineering
14 paradigms, may not be aware of AI-first’s consequences. Application developers
15 build *rule-driven* applications, where every line of source code evaluates to produce
16 deterministic outcomes. AI-first software is, however, not rule-driven but *data-
driven*. Large datasets train machine learning (ML) models that result in probabilistic
17 confidences of results and nondeterministic behaviour if it continually learns from
18 data with time. Furthermore, developing AI-first applications requires both code
19 and data, and an application developer can approach developing from three (non-
20 traditional) perspectives, further expanded in Section 1.1:

- 22 1. The application developer writes an ML model from scratch and trains it from
23 a handcrafted and curated dataset. This approach is laborious in time and
24 demands formal training in ML and mathematical knowledge, but the tradeoff
25 is that they have full autonomy in the models they creates.
- 26 2. The application developer downloads a pre-trained model and ‘plugs’ it into
27 an existing ML framework, such as Tensorflow [1]. While this approach is
28 less demanding in time, it requires them to revise and understand how to ‘glue’
29 components of the ML framework together² into their application’s code.

¹A 2016 report by market research company Forrester captured such growth into four key areas [219], as reproduced in Figure A.2.

²Thus introducing a verbose list of ML terminology to her developer vocabulary. See a list of 328 terms provided by Google here: <https://developers.google.com/machine-learning/glossary/>. Last accessed 7 December 2018.

Figure 1.1: The application developer’s rule-driven toolchain is distinct from data-driven toolchain. A developer must consume a typical, data-driven cloud service in a different way than an intelligent data-driven cloud service as they are not the same type of system.



- 30 3. The application developer uses a data-driven and cloud-based service. They
31 don’t need to know anything behind the underlying ‘intelligence’ and how it
32 functions. It is fast to integrate into their applications, and the APIs offered
33 abstracts the technical know-how behind a web call.
- 34 The documentation of the service alludes that the data-driven service is as similar to
35 other cloud services offered by the provider. Because this is ‘another’ cloud service,
36 the application developer *assumes* it would act and behave as any other typical
37 service would. But does this assumption—and a lack of appreciation of ML—lead
38 to developer pain-points and miscomprehension? If so, how can the service providers
39 improve their documentation to alleviate this? Do these data-driven services share
40 similarities to the runtime behaviour of traditional cloud services? And if not,
41 how best can the application developer integrate the data-driven service into their a
42 rule-driven application to produce AI-first software?

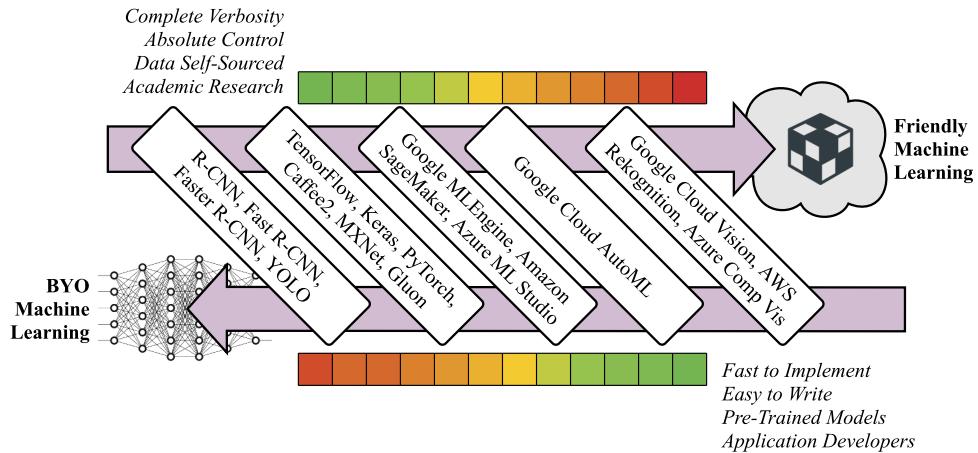
43 Figure 1.1 provides an illustrative overview between the context clashing of rule-
44 driven applications and data-driven cloud services, and we contrast characteristics
45 of typical cloud systems and data-driven ones in Table 1.1.

In this thesis, we advocate that the integration and developer comprehension of data-driven cloud services differ from the rule-driven nature of end-applications. As ‘intelligent’ components these contrast to traditional counterparts, and application developers need to take into account a greater appreciation of these factors.

46 1.1 Research Context

- 47 As described, the application developer has three key approaches in producing AI-
48 first software. This ‘range’ of AI-first integration techniques partially reflects Google

Figure 1.2: Examples within the machine learning spectrum of computer vision. Colour scales indicates the benefits (green) and drawbacks (red) of each end of the spectrum.



49 AI's³ *machine learning spectrum* [204, 232, 263], which encompasses the variety
 50 of skill, effort, users and types of outputs of integration techniques. One extreme
 51 involves the academic research of developing algorithms and self-sourcing data
 52 to achieve intelligence—coined as Build Your Own Machine Learning (BYOML)
 53 [178, 232, 263]. The other extreme involves off-the-shelf, ‘friendlier’ (abstracted)
 54 intelligence with easy-to-use APIs targeted towards applications developers. The
 55 middle-ground involves a mix of the two, with varying levels of automation to assist
 56 in development, that turns custom datasets into predictive intelligence. We illustrate
 57 the slightly varied characteristics within this spectrum in Table 1.2 and Figure 1.2.

58 These data-driven ‘friendly’ services are gaining traction within developer circles:
 59 we show an increasing trend of Stack Overflow posts mentioning a mix of
 60 intelligent computer vision services in Figure A.3.⁴ Academia provides varied
 61 nomenclature for these services, such as *Cognitive Applications* and *Machine Learning*
 62 *Services* [358] or *Machine Learning as a Service* [291]. For the context of this
 63 thesis, we will refer to such services under broader term of **intelligent web services**
 64 (**IWSs**), and diagrammatically express their usage within Figure 1.3.

65 While there are many types of IWSs available to software developers,⁵ the
 66 general workflow of using an IWS is more-or-less the same: a developer accesses
 67 an IWS component via REST/SOAP API(s), which is (typically) available as a

³Google AI was recently rebranded from Google Research, further highlighting how the ‘AI-first’ philosophy is increasingly becoming embedded in companies’ product lines and research and development teams. Spearheaded through work achieved at Google, Microsoft and Facebook, the emphasis on an AI-first attitude we see through Google’s 2018 rebranding of *Google Research* to *Google AI* [163] is evident. A further example includes how Facebook leverage AI *at scale* within their infrastructure and platforms [267].

⁴Query run on 12 October 2018 using StackExchange Data Explorer. Refer to <https://data.stackexchange.com/stackoverflow/query/910188> for full query.

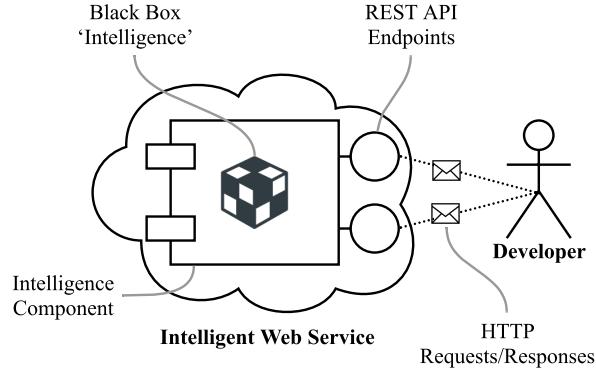
⁵Such as optical character recognition, text-to-speech and speech-to-text transcription, object categorisation, facial analysis and recognition, natural language processing etc.

Table 1.1: Differing characteristics of intelligent and typical web services.

Intelligent web service	Typical web services
Probabilistic	Deterministic
Machine Learnt	Human Engineered
Data-Driven	Rule-Driven
Black-Box	Mostly Transparent

Table 1.2: Comparison of the machine learning spectrum.

Comparator	BYOML	ML F'work	Cloud ML	Auto-Cloud ML	Cloud API
Hosting					
Locally	✓	✓			
Cloud			✓	✓	✓
Output					
Custom Model	✓	✓	✓	✓	
HTTP Response					✓
Autonomy					
Low					✓
Medium				✓	
High		✓	✓		
Highest	✓				
Time To Market					
Medium	✓	✓			
High			✓	✓	
Highest					✓
Data					
Self-Sourced	✓	✓	✓	✓	
Pre-Trained		✓			✓
Intended User					
Academics	✓	✓			
Data Scientist	✓	✓	✓	✓	
Developers				✓	✓

Figure 1.3: Overview of IWSs.

⁶⁸ cloud-based Platform as a Service (PaaS).^{6,7} For a given input, developers receive
⁶⁹ an ‘intelligent’ response and an associated confidence value that represents the
⁷⁰ likelihood of that result. This is typically serialised as a JSON/XML response
⁷¹ object.

☞ *Within this thesis, we scope our investigation to a mature subset of IWSs that provide computer vision intelligence [383, 386, 399, 400, 401, 407, 411, 420, 421, 423, 425, 472, 473]. For the context of this thesis, we will refer to such services as **computer vision services (CVSs)**.*

⁷² 1.2 Motivating Scenarios

⁷³ The market for computer vision services (CVSs) is increasing (Figure A.2) and as
⁷⁴ is developer uptake and enthusiasm in the software engineering community (Figure
⁷⁵ A.3). However, the impact to software quality (internal and external) due to
⁷⁶ a mismatch of the application developer’s deterministic mindset and the service
⁷⁷ provider’s nondeterministic mindset is of concern.

⁷⁸ To illustrate the context of use, we present the two scenarios of varying risk: (i) a
⁷⁹ fictional software developer, named Tom, who wishes to develop an inherently low-
⁸⁰ risk photo labelling application for his friends and family; and (ii) a high-risk cancer
⁸¹ clinical decision support system (CDSS) that uses patient scans to recommend if

⁶We note, however, that a development team may use a similar approach *internally* within a product line or service that may not necessarily reflect a PaaS model.

⁷A number of services provide the platform infrastructure to rapidly begin training from custom datasets, such as Google’s AutoML (<https://cloud.google.com/automl/>, last accessed 7 December 2018). Others provide pre-trained datasets ‘ready-for-use’ in production without the need to train data.

⁸² surgeons should send their patients to surgery. Both describe scenarios where AI-
⁸³ first components has substantiative impact to end-users when the software engineers
⁸⁴ developing with them misunderstand the nuances of ML, ultimately adversely affecting
⁸⁵ external quality. Moreover, due to lack of comprehension, this hinders developer
⁸⁶ experience, productivity, and understanding/appreciation of AI-based components.

⁸⁷ *1.2.0.1 Motivating Scenario I: Tom's PhotoSharer App*

⁸⁸ Tom wants to develop a social media photo-sharing app on iOS and Android, *Photo-*
⁸⁹ *Sharer*, that analyses photos taken on smartphones. Tom wants the app to categorise
⁹⁰ photos into scenes (e.g., day vs. night, landscape vs. indoors), generate brief de-
⁹¹ scriptions of each photo, and catalogue photos of his friends and common objects
⁹² (e.g., photos with his Border Collie dog, photos taken on a beach on a sunny day with
⁹³ his partner). His app will shares this analysed photo intelligence with his friends on
⁹⁴ a social-media platform, where his friends can search and view the photos.

⁹⁵ Instead of building a computer vision engine from scratch, which takes too much
⁹⁶ time and effort, Tom thinks he can achieve this using one of the common CVSs. Tom
⁹⁷ comes from a typical software engineering background and has insufficient knowl-
⁹⁸ edge of key computer vision terminology and no understanding of its underlying
⁹⁹ techniques. However, inspired by easily accessible cloud APIs that offer computer
¹⁰⁰ vision analysis, he chooses to use these. Built upon his experience of using other
¹⁰¹ similar cloud services, he decides on one of the CVS APIs, and expects a static result
¹⁰² always and consistency between similar APIs. Analogously, when Tom invokes the
¹⁰³ iOS Swift substring method "doggy".prefix(3), he expects it to be consistent
¹⁰⁴ with the Android Java equivalent "doggy".substring(0, 2). Consistent, here,
¹⁰⁵ means two things: (i) that calling `substring` or `prefix` on 'dog' will *always*
¹⁰⁶ return in the same way every time he invokes the method; and (ii) that the result is
¹⁰⁷ *always* 'dog' regardless of the programming language or string library used, given
¹⁰⁸ the deterministic nature of the 'substring' construct (i.e., results for `substring` are
¹⁰⁹ API-agnostic).

¹¹⁰ More concretely, in Table 1.3, we illustrate how three (anonymised) CVS
¹¹¹ providers fail to provide similar consistency to that of the substring example above.
¹¹² If Tom uploads a photo of a border collie⁸ to three different providers in August
¹¹³ 2018 and January 2019, he would find that each provider is different in both the vo-
¹¹⁴ cabulary used between. The confidence values and labels within the *same* provider
¹¹⁵ varies within a matter of five months. The evolution of the confidence changes is not
¹¹⁶ explicitly documented by the providers (i.e., when the models change) nor do they
¹¹⁷ document what confidence means. Service providers use a tautological nature when
¹¹⁸ defining what the confidence values are (as presented in the API documentation)
¹¹⁹ provides no insight for Tom to understand why there was a change in confidence,
¹²⁰ which we show in Table 1.4, unless he *knows* that the underlying models change with
¹²¹ them. Furthermore, they do not provide detailed understanding on how to select a
¹²² threshold cut-off for a confidence value. Therefore, he's left with no understanding
¹²³ on how best to tune for image classification in this instance. The deterministic prob-

⁸The image used for these results is <https://www.akc.org/dog-breeds/border-collie/>.

Table 1.3: First six responses of image analysis for a Border Collie sent to three CVS providers five months apart. The specificity (to 3 s.f.) and vocabulary of each label in the response varies between all services, and—except for Provider B—changes over time. Any confidence changes greater than 1 per cent are highlighted in red.

Label	Provider A		Provider B		Provider C	
	Aug 2018	Jan 2019	Aug 2018	Jan 2019	Aug 2018	Jan 2019
Dog	0.990	0.986	0.999	0.999	0.992	0.970
Dog Like Mammal	0.960	0.962	-	-	-	-
Dog Breed	0.940	0.943	-	-	-	-
Border Collie	0.850	0.852	-	-	-	-
Dog Breed Group	0.810	0.811	-	-	-	-
Carnivoran	0.810	0.680	-	-	-	-
Black	-	-	0.992	0.992	-	-
Indoor	-	-	0.965	0.965	-	-
Standing	-	-	0.792	0.792	-	-
Mammal	-	-	0.929	0.929	0.992	0.970
Animal	-	-	0.932	0.932	0.992	0.970
Canine	-	-	-	-	0.992	0.970
Collie	-	-	-	-	0.992	0.970
Pet	-	-	-	-	0.992	0.970

¹²⁴ lem of a substring compared to the nondeterministic nature of the IWS is, therefore,
¹²⁵ non-trivial.

¹²⁶ To make an assessment of these APIs, he tries his best to read through the
¹²⁷ documentation of different CVS APIs, but he has no guiding framework to help him
¹²⁸ choose the right one. A number of questions come to mind:

- ¹²⁹ • What does ‘confidence’ mean?
- ¹³⁰ • Which confidence is acceptable in this scenario?
- ¹³¹ • Are these APIs consistent in how they respond?
- ¹³² • Are the responses in APIs static and deterministic?
- ¹³³ • Would a combination of multiple CVS APIs improve the response?
- ¹³⁴ • How does he know when there is a defect in the response? How can he report
¹³⁵ it?
- ¹³⁶ • How does he know what labels the API knows, and what labels it doesn’t?
- ¹³⁷ • How does it describe his photos and detect the faces?
- ¹³⁸ • Does he understand that the API uses a machine learnt model? Does he know
¹³⁹ what a ML model is?
- ¹⁴⁰ • Does he know when models update? What is the release cycle?

¹⁴¹ Although Tom generally anticipates these CVSs to not be perfect, he has no
¹⁴² prior benchmark to guide him on what to expect. The imperfections appear to be
¹⁴³ low-risk, but may become socially awkward when in use; for instance, if Tom’s
¹⁴⁴ friends have low self-esteem and use the app, they may be sensitive to the app not
¹⁴⁵ identifying them or mislabelling them. Privacy issues come into play especially
¹⁴⁶ if certain friends have access to certain photos that they are (supposedly) in; e.g.,

Table 1.4: Tautological definitions of ‘confidence’ found in the API documentation of three common CVS providers.

API Provider	Definition(s) of Confidence
Provider A	“Score is the confidence score, which ranges from 0 (no confidence) to 1 (very high confidence).” [409]
	“Deprecated. Use score instead. The accuracy of the entity detection in an image. For example, for an image in which the ‘Eiffel Tower’ entity is detected, this field represents the confidence that there is a tower in the query image. Range [0, 1].” [410]
	“The overall score of the result. Range [0, 1]” [410]
Provider B	“Confidence score, between 0 and 1... if there insufficient confidence in the ability to produce a caption, the tags maybe [sic] the only information available to the caller.” [426]
Provider C	“The level of confidence the service has in the caption.” [424]
	“The response shows that the operation detected five labels (that is, beacon, building, lighthouse, rock, and sea). Each label has an associated level of confidence. For example, the detection algorithm is 98.4629% confident that the image contains a building.” [384]
	“[Provider C] also provide[s] a percentage score for how much confidence [Provider C] has in the accuracy of each detected label.” [385]

¹⁴⁷ photos from a holiday with Tom and his partner, however if the API identifies Tom’s
¹⁴⁸ partner as a work colleague, Tom’s partner’s privacy is at risk.

¹⁴⁹ Therefore, the level of risk and the determination of what constitutes an ‘error’ is
¹⁵⁰ dependent on the situation. In the following example, an error caused by the service
¹⁵¹ may be more dangerous.

¹⁵² 1.2.0.2 Motivating Scenario II: Cancer Detection CDSS

¹⁵³ Recent studies in the oncology domain have used deep-learning convolutional neural
¹⁵⁴ networks (CNNs) to detect region of interests (ROIs) in image scans of tissue (e.g.,
¹⁵⁵ [32, 147, 218]), flagging these regions for doctors to review. Trials of such algorithms
¹⁵⁶ have been able to accurately detect cancer at higher rates than humans, and thus
¹⁵⁷ incorporating such capabilities into a CDSS is closer within reach. Studies have
¹⁵⁸ suggested these systems may erode a practitioner’s independent decision-making
¹⁵⁹ [74, 175] due to over-reliance; therefore the risks in developing CDSSs powered by
¹⁶⁰ IWSs become paramount.

¹⁶¹ In Figure 1.4 we present a context diagram for a fictional CDSS named *CancerAssist*. A team of busy pathologists utilise CancerAssist to review patient lymph
¹⁶² node scans and discuss and recommend, on consensus, if the patient requires an
¹⁶³ operation. When the team makes a consensus, the lead pathologist enters the ver-

165 dict into CancerAssist—running passively in the background—to ensure there is
 166 no oversight in the team’s discussions. When a conflict exists between the team’s
 167 verdict and CancerAssist’s verdict, the system produces the scan with ROIs it thinks
 168 the team should review. Where the team overrides the output of CancerAssist, this
 169 reinforces CancerAssist’s internal model as a human-in-the-loop (HITL) learning
 170 process.

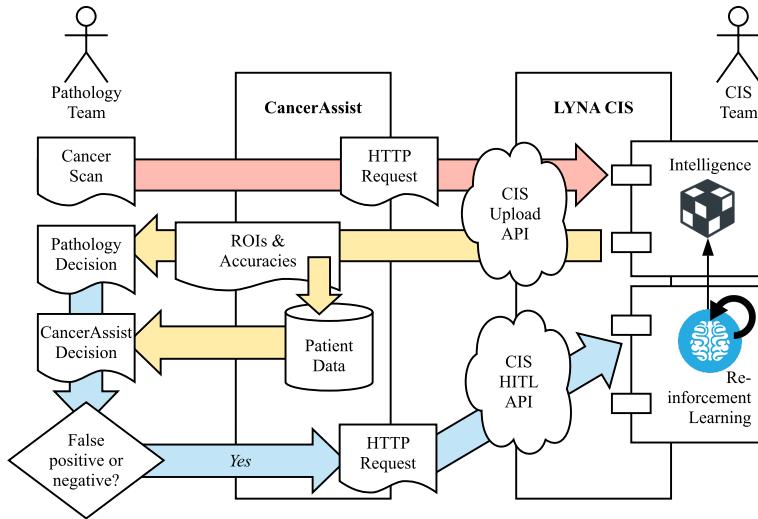


Figure 1.4: CancerAssist Context Diagram. **Key:** Red Arrows = Scan Input; Yellow Arrows = Decision Output; Blue Arrows = HITL Feedback Input.

171 Powering CancerAssist is Google AI’s Lymph Node Assistant (LYNA) [218],
 172 a CNN based on the Inception-v3 model [201, 338]. To provide intelligence to
 173 CancerAssist, the development team decide to host LYNA as an IWS using a cloud-
 174 based PaaS solution. Thus, CancerAssist provides API endpoints integrated with
 175 patient data and medical history, which produces the verdict. In the case of a positive
 176 verdict, CancerAssist highlights the relevant ROIs found are with their respective
 177 bounding boxes and their respective cancer detection accuracies.

178 The developer of CancerAssist has no interaction with the Data Science team
 179 maintaining the LYNA IWS. As a result, they are unaware when updates to the
 180 model occur, nor do they know what training data they provide to test their system.
 181 The default assumptions are that the training data used to power the intelligence
 182 is near-perfect for universal situations; i.e., the algorithm chosen is the correct one
 183 for every assessable ontology tests in the given use case of CancerAssist. Thus,
 184 unlike deterministic systems—where the developer can manually test and validate
 185 the outcomes of the APIs—this is impossible for non-deterministic systems such
 186 as CancerAssist and its underlying IWS. The ramifications of not being able to test
 187 such a system and putting it out into production may prove fatal to patients.

188 Certain questions in the production of CancerAssist and its use of an IWS may
 189 come into mind:

- 190 • When is the model updated and how do the IWS team communicate these

¹⁹¹ updates?

- ¹⁹² • What benchmark test set of data ensures that the changed model doesn't affect other results?
- ¹⁹⁴ • Are assumptions made by the IWS team who train the model correct?

¹⁹⁵ Thus, to improve communication between developers and IWS providers, developers require enhanced documentation, additional metadata, and guidance tooling.

¹⁹⁷ 1.3 Research Motivation

¹⁹⁸ Evermore applications are using IWSs as demonstrated by ubiquitous examples: ¹⁹⁹ aiding the vision-impaired [94, 289], accounting [226], data analytics [173], and ²⁰⁰ student education [100]. As our motivating examples have illustrated, these AI-based ²⁰¹ components—specifically CVSs—are accessible through APIs consisting of ‘black ²⁰² box’ intelligence (Figure 1.3).⁹ Data science teams produce ML algorithms to make ²⁰³ predictions in our datasets and discover patterns within them. As these algorithms ²⁰⁴ are data-dependent, they are therefore inherently probabilistic and stochastic, which ²⁰⁵ results in four critical issues that motivate our thesis: (i) certainty in results, (ii) ²⁰⁶ evolution of datasets, (iii) selecting appropriate decision boundaries, and (iv) the ²⁰⁷ clarity of ML documentation that address items i–iii.

²⁰⁸ There is little room for certainty in these results as the insight is purely statistical ²⁰⁹ and associational [274] against its training dataset. Developers who build these ²¹⁰ applications **do not treat their programs with a stochastic or probabilistic** ²¹¹ **mindset, given that they are trained with a rule-driven mindset that computers** ²¹² **make certain outcomes.** However, CVSs are data-driven, and therefore return the ²¹³ *probability* that a particular object exists in an input images’ pixels via confidence ²¹⁴ values. As an example, consider simple arithmetic representations (e.g., $2 + 2 =$ ²¹⁵ 4). The deterministic (rule-driven) mindset suggests that the result will *always* be ²¹⁶ 4. However, the non-deterministic (data-driven) mindset suggests that results are ²¹⁷ probable: target output (*exactly* 4) and the output inferred (*a likelihood of* 4) matches ²¹⁸ as a probable percentage (or as an error where it does not match).¹⁰ Instead of an ²¹⁹ exact output, there is a *probabilistic* result: $2 + 2$ *may* equal 4 to a confidence of n . ²²⁰ Thus, for a more certain (though not fully certain) distribution of overall confidence ²²¹ returned from the service, a developer must treat the problem stochastically by ²²² testing this case hundreds if not thousands of times to find a richer interpretation of ²²³ the inference made and ensure reliability in its outcome.

²²⁴ Traditional software engineering principles advocate for software systems to be ²²⁵ versioned upon substantial change. Unfortunately **we find that the most prominent** ²²⁶ **cloud vendors providing these intelligent services (e.g., Microsoft Azure, Google** ²²⁷ **Cloud and Amazon Web Services) do not release new versioned endpoints of the**

⁹The ‘black box’ refers to a system that transforms input (or stimulus) to outputs (or response) without any understanding of the internal architecture by which this transformation occurs. This arises from a theory in the electronic sciences and adapted to wider applications since the 1950s–60s [16, 64] to describe “systems whose internal mechanisms are not fully open to inspection” [16].

¹⁰Blake et al. [43] produces a multi-layer perceptron neural network performing arithmetic representation.

228 APIs when the *internal model* changes [87]. In the context of computer vision, new
229 labels may be introduced or dropped, confidence values may differ, entire ontologies
230 or specific training parameters may change, but we hypothesise that is not effectively
231 communicated to developers. Broadly speaking, this can be attributed to a dichotomy
232 of release cycles from the data science and software engineering communities: the
233 data science iterations and work by which new models are trained and released runs
234 at a faster cycle than the maintenance cycle of traditional software engineering. Thus
235 we see cloud vendors integrating model changes without the *need* to update the API
236 version unless substantial code or schema changes are also introduced—the nuance
237 changes in the internal model does not warrant a shift in the API itself, and therefore
238 the version shift in a new model does not always propagate to a version shift in the
239 API endpoint. As demonstrated in Table 1.3, whatever input is uploaded at one time
240 may not necessarily be the same when uploaded at a later time. This again contrasts
241 the rule-driven mindset, where $2 + 2$ *always* equals 4. Therefore, in addition to the
242 certainty of a result in a single instance, the certainty of a result in *multiple instances*
243 may differ with time, which again impacts on the developers notion of reliable
244 software. Currently, it is impossible to invoke requests specific to a particular model
245 that was trained at a particular date in time, and therefore developers need to consider
246 how evolutionary changes of the services may impact their solutions *in production*.
247 Again, whether there is any noticeable behavioural changes from these changes is
248 dependent on the context of the problem domain—unless developers benchmark
249 these changes against their own domain-specific dataset and frequently check their
250 selected service against such a dataset, there is no way of knowing if substantive
251 errors have been introduced.

252 As the only response from these computer vision classifiers are a label and
253 confidence value; **the decision boundaries needs to always be appropriately con-**
254 sidered by client code for each use case and each model selected. The external
255 quality of such software needs to consider reliability in the case of thresholding con-
256 fidence values—that is whether the inference has an appropriate level of confidence
257 to justify a predicted (and reliable) result to end-users. Selecting this confidence
258 threshold is non-trivial; a ML course from Google suggests that “it is tempting
259 to assume that [a] classification threshold should always be 0.5, but thresholds
260 are problem-dependent, and are therefore values that you must tune.” [141]. Ap-
261 proaches to turning these values are considered for data scientists, but are not yet
262 well-understood for application developers with little appreciation of the nuances of
263 ML.

264 Similarly, developers should consider the internal quality of building AI-first
265 software. Reliable API usability and documentation advocate for the accuracy,
266 consistency and completeness of APIs and their documentation [279, 296] and
267 providers should consider mismatches between a developer’s conceptual knowledge
268 of the API its implementation [195]. **Unreliable APIs ultimately hinder developer**
269 **performance and thus reduces productivity**, in addition to producing potentially
270 unreliable software where documentation is not well-understood (or clear to the
271 developer).

272 Ultimately, these four issues present major threats to software reliability if left

273 unresolved. Given that such substantiative software engineering principles on re-
 274 liability, versioning and quality are under-investigated within the context of IWSs,
 275 we aim to explore guidance from the software engineering literature to investigate
 276 what aspects in the development lifecycle could aide in mitigating these issues when
 277 developing using AI-based components. This serves as our core motivation for this
 278 work.

279 1.4 Research Goals

280 This thesis aims to investigate and better understand the nature of cloud-based
 281 computer vision services (CVSs)¹¹ as a concrete exemplar of intelligent web services
 282 (IWSs). We identify the maturity, viability and risks of CVSs through the anchoring
 283 perspective of *reliability* that affects the internal and external quality of software.
 284 We adopt the McCall [230] and Boehm [45] interpretations of reliability via the sub-
 285 characteristics of a service's *consistency* and *robustness* (or fault/error tolerance), and
 286 the *completeness*¹² of its documentation. (A detailed discussion is further provided
 287 in Section 2.1.) This thesis explores and contributes towards *four* key facets regarding
 288 reliability in CVS usage and the completeness of its associated documentation. We
 289 formulate four primary research questions (RQs) with seven sub-RQs, based on
 290 both empirical and non-empirical software engineering methodology [240], further
 291 discussed in Chapter 3.

292 Firstly, we investigate adverse implications that arise when using CVSs that
 293 affects consistency and robustness (**Chapter 4**). We show how CVSs have a non-
 294 deterministic runtime behaviour and evolve with unintended and non-trivial con-
 295 sequences to developers. We demonstrate that these services have inconsistent
 296 behaviour despite offering the same functionality and pose evolution risk that ef-
 297 fects robustness of consuming applications when responses change given the same
 298 (consistent) inputs. Thus, we conclude how the nature of these services (at present)
 299 are not fully robust, consistent, and thus not reliable. Formally, we structure the
 300 following RQs:

301 ② RQ1. What is the nature of cloud-based CVSs?

302 *RQ1.1.* What is their runtime behaviour?

303 *RQ1.2.* What is their evolution profile?

304 Secondly, we investigate the reliability of the documentation these services of-
 305 fer through the lenses of its completeness. We collate prior knowledge of good
 306 API documentation and assess the efficacy of such knowledge against practitioners
 307 (**Chapter 8**). We show that these service's behaviour and evolution is not
 308 reliably documented adequately against this knowledge. Formally, we develop the
 309 following RQs:

¹¹As these services are proprietary, we are unable to conduct source code or model analysis, and hence are not used in the investigation of this thesis.

¹²We treat the API documentation of a CVS as a first-class citizen.

⌚ RQ2. Are CVS APIs sufficiently documented?

- RQ2.1.* What are the dimensions of a ‘complete’ API document, according to both literature and practitioners?
- RQ2.2.* What additional information or attributes do application developers need in CVS API documentation to make it more complete?

307 Thirdly, we investigate how software developers approach using these services
308 and directly assess developer pain-points resulting from the nature of CVSs and
309 their documentation (**Chapter 5**). We show that there is a statistically significant
310 difference in these complaints when contrasted against more established software
311 engineering domains (such as web or mobile development) as expressed as ques-
312 tions asked on Stack Overflow. We provide a number of exploratory avenues for
313 researchers, educators, software engineers and IWS providers to alleviate these com-
314 plaints based on this analysis. Further, using a data set consisting of 1,245 Stack
315 Overflow questions, we explore the emotional state of developers to understand
316 which aspects (i.e., pain-points) developers are most frustrated with (**Chapter 6**)
317 and the types of traps developers can fall into when substantial documentation is not
318 provided for specific ML models (**Chapter 7**). We formulate the following RQs:

**⌚ RQ3. Are CVSs more misunderstood than conventional software en-
gineering domains?**

- RQ3.1.* What types of issues do application developers face most when using CVSs, as expressed as questions on Stack Overflow?
- RQ3.2.* Which of these issues are application developers most frustrated with?
- RQ3.3.* Is the distribution CVS pain-points different to established software engineering domains, such as mobile or web development?

319 Lastly, we explore several strategies to help improve CVSs reliability. Firstly,
320 we investigate whether merging the responses of *multiple* CVSs can improve their
321 reliability and propose a novel algorithm—based on the proportional representation
322 method used in electoral systems—to merge labels and associated confidence values
323 from three providers (**Chapter 9**). Secondly, we develop an integration architec-
324 ture style (or facade) to guard against CVS evolution, and synthesise an integration
325 workflow that addresses the concerns raised by developers in addition to embed-
326 ding ‘complete’ documentation artefacts into the workflow’s design (**Chapters 10**
327 and **11**). Our final RQ is:

**⌚ RQ4. What strategies can developers employ to integrate their appli-
cations with CVSs while preserving robustness and reliability?**

328 1.5 Research Methodology

329 This thesis employs a mixed-methods approach using the concurrent triangulation
330 strategy [57, 229]. The research presented consists of both empirical and non-
331 empirical research design. This section provides a high-level overview of the re-
332 search methodology within this thesis. Further details are provided in Section 1.7
333 and Chapter 3.

334 Firstly, RQ1–RQ3 are all empirical, knowledge-based questions [109, 236] that
335 aim to provide the software engineering community with a greater understanding
336 of the phenomena surrounding CVSs from three perspectives: the nature of the ser-
337 vices themselves, how developers perceive these services and how service providers
338 can improve these services. We answer RQ1 using a longitudinal experiment that
339 assesses both the services’ responses and associated documentation (complement-
340 ing RQ2.2). We adopt qualitative and quantitative data collection; specifically (i)
341 structured observations to quantitatively analyse the results over time, and (ii) docu-
342 mentary research methods to inspect service documentation. Secondly, we perform
343 systematic mapping study following the guidelines of Kitchenham and Charters
344 [191] and Petersen et al. [276] to better understand how API documentation of these
345 services can be improved (i.e., more complete), which targets RQ2. Based on the
346 findings from this study, we use a systematic taxonomy development methodology
347 specifically targeted toward software engineering [351] that structures scattered API
348 documentation knowledge into a taxonomy. We then validate this taxonomy against
349 practitioners using survey research, adopting Brooke well-established Systematic
350 Usability Score [61] surveying instrument and contextualising it within API docu-
351 mentation utility, which answers RQ3.3. To answer RQ2.2, we perform an empirical
352 application of the taxonomy to three CVSs, and therefore assess where improve-
353 ments can be made. Thirdly, we adopt field survey research using repository mining
354 of developer discussion forums (i.e., Stack Overflow) to answer RQ3, and classify
355 these using both manual and automated techniques.

356 The second aspect of our research design involves non-empirical research, which
357 explores a design-based question [240] to answer RQ4. As the answers to our
358 first three RQs establish a greater understanding of the nature behind CVSs from
359 various perspectives, the strategies we design in RQ4 aims at designing more reliable
360 integration methods so that developers can better use these cloud-based services in
361 their applications.

362 1.6 Thesis Organisation

363 We organise the thesis into four parts. **Part I (The Preface)** includes introduc-
364 tory, background and methodology chapters. This is a *PhD by Publication*, and
365 **Part II (Publications)** comprises of seven publications resulting from this work
366 over Chapters 4 to 6 and 8 to 11; publications are included verbatim except for
367 terminology and formatting changes to better fit the suitability of a coherent the-
368 sis. **Part III (The Postface)** includes the conclusion and future works chapter, as
369 well as a list of academic studies and online artefacts referenced within the thesis.

370 **Part IV (Appendices)** includes all supplementary material, including mandatory
371 authorship statements and ethics approval. Details of each chapter following this
372 introductory chapter are provided in the following section.

373 1.6.1 Part I: Preface

374 1.6.1.1 *Chapter 2: Background*

375 This chapter provides an overview of prior studies broadly around three key pillars:
376 the development of an IWS, the usage of an IWS, and the nature of an IWS. We use
377 the three perspectives of software quality (particularly, reliability), probabilistic and
378 non-deterministic systems, and explanation and communication theory to describe
379 prior work.

380 1.6.1.2 *Chapter 3: Research Methodology*

381 This chapter provides a summative review of research methods and philosophical
382 stances relevant to software engineering. We illustrate that the methods used within
383 our publications are sound via an analysis of the methodologies used in seminal
384 works referenced in this thesis.

385 1.6.2 Part II: Publications

386 1.6.2.1 *Chapter 4: Exploring the nature of CVSSs*

387 This chapter was presented at the 2019 International Conference on Software
388 Maintenance and Evolution (ICSME) [87]. We describe an 11-month longitudinal
389 experiment assessing the behavioural (run-time) issues of three popular CVSSs:
390 Google Cloud Vision [411], Amazon Rekognition [386] and Azure Computer Vi-
391 sion [425]. By using three different data sets—two of which we curate as additional
392 contributions—we demonstrate how the services are inconsistent amongst each other
393 and within themselves. This study provides a detailed answer to RQ1: Despite
394 presenting conceptually-similar functionality, each service behaves and produces
395 slightly varied (inconsistent) results and demonstrates non-deterministic runtime
396 behaviour. We discuss potential evolution risks to consumers of such services as the
397 services provide non-static outputs for the same inputs, thereby having significant
398 impact to the robustness of consuming applications. Further details in the study
399 include a brief assessment into the lack of sufficient detail of these concerns in their
400 documentation.

401 1.6.2.2 *Chapter 5: Understanding developer struggles when using CVSSs*

402 This chapter has been accepted for presentation at the 2020 International Conference
403 on Software Engineering (ICSE) [90]. We conduct a mining study of 1,425 Stack
404 Overflow questions that provide indications of the types frustrations that developers
405 face when integrating CVSSs into their applications. To gather what their pain-
406 points are, we use two classification taxonomies that also use Stack Overflow to

⁴⁰⁷ understand generalised and documentation-specific pain-points in mature software
⁴⁰⁸ engineering domains. This study answers RQ3 in detail and provides a validation
⁴⁰⁹ to our motivation of RQ2: we validate that the *completeness* of current CVS API
⁴¹⁰ documentation is a main concern for developers and there is insufficient explanation
⁴¹¹ into the errors and limitations of the service. We find that the documentation does
⁴¹² not adequately cover all aspects of the technical domain. In terms of integrating with
⁴¹³ the service, developers struggle most with simple errors and ways in which to use the
⁴¹⁴ APIs; this is in stark contrast to mature software domains. Our interpretation is that
⁴¹⁵ developers fail to understand the IWS lifecycle and the ‘whole’ system that wraps
⁴¹⁶ such services. We also interpret that developers have a shallower understanding
⁴¹⁷ of the core issues within CVSs (likely due to the nuances of ML as suggested in
⁴¹⁸ a discussion in the paper), which warrants an avenue for future work in software
⁴¹⁹ engineering education.

⁴²⁰ 1.6.2.3 *Chapter 6: Ranking CVS pain-points by frustration*

⁴²¹ This chapter has been published as a technical report pre-print on arXiv and an
⁴²² extended version is in progress for submission to IEEE Software [93]. In this work,
⁴²³ we use our dataset consisting of the 1,425 Stack Overflow questions from [90] to
⁴²⁴ interpret the breakdown of emotions developers express per classification of pain-
⁴²⁵ points conducted in Chapter 5. We find that the distribution of various emotions
⁴²⁶ differ per question type, and developers are most frustrated when the expectations
⁴²⁷ of a CVS does not match the reality of what these services actually provide, which
⁴²⁸ shapes our answer for RQ3.2 and thus RQ3.

⁴²⁹ 1.6.2.4 *Chapter 7: Lessons in applying pre-trained models to Stack Overflow*

⁴³⁰ This chapter has been submitted as an industry track paper for the International
⁴³¹ Conference on Automated Software Engineering (ASE) [142]. This work presents
⁴³² a deeper investigation into the classification model used within Chapter 6 to better
⁴³³ interpret the automation effort we conducted, thereby highlighting valuable lessons
⁴³⁴ we learnt from performing this exercise. Specifically, we find that the classification
⁴³⁵ model we used in this exercise presented substantial data imbalance, which presented
⁴³⁶ unexpected results (namely, a high level of posts that showed the emotion, ‘love’). We
⁴³⁷ identify how novel documentation tooling such as model cards [243] or datasheets
⁴³⁸ [132] could have identified risks to our study earlier, and make suggestions needed
⁴³⁹ into future documentation efforts. This work presents complementary results to
⁴⁴⁰ Item RQ2 to help propose which documentation elements ML models (and thus
⁴⁴¹ IWSs) should provide before diving ‘straight in’.

⁴⁴² 1.6.2.5 *Chapter 8: Investigating improvements to CVS API documentation*

⁴⁴³ This chapter was originally a short paper presented at the 2019 International Sym-
⁴⁴⁴ posium on Empirical Software Engineering and Measurement (ESEM) [90]. To
⁴⁴⁵ understand where to improve CVS documentation, we first need to investigate *what*
⁴⁴⁶ makes a good API document. This short paper initially answered one aspect of

447 RQ2.1: what *academic literature* suggests a good (complete) API document should
448 comprise of. By conducting an systematic mapping study resulting in 21 primary
449 studies, we systematically develop a taxonomy that combines the recommendations
450 of scattered work into a structured framework of 5 dimensions and 34 weighted cat-
451 egorisations. We then extend this work by triangulating the taxonomy with opinions
452 from developers using the System Usability Scale to assess the efficacy of these
453 recommendations (thereby answering the second aspect of RQ2.1). From this, we
454 assess the how well CVS providers document their APIs via a heuristic validation
455 of the taxonomy, using the three services from the ICSME publication to make rec-
456 ommendations where documentation should be more complete, thereby answering
457 RQ2.2 (and thus RQ2). The extended version of this chapter has been submitted to
458 the IEEE Transactions on Software Engineering (TSE) in [91] and is currently in
459 review.

460 *1.6.2.6 Chapter 9: Merging responses of multiple CVSs*

461 This chapter was presented at the 2019 International Conference on Web Engineer-
462 ing (ICWE) [260]. Early exploration of CVSs showed that multiple services use
463 vastly different ontologies for the same input. As an initial strategy to improve
464 the reliability of these services, we explored if merging multiple responses using
465 WordNet [242] and a novel label merging algorithm based on the proportional rep-
466 resentation approach used in political voting could make any improvements. While
467 this approach resulted in a modest improvement to reliability, it did not consider to
468 the evolution issues or developer pain-points we later identified.

469 *1.6.2.7 Chapter 10: Developing a confidence thresholding tool*

470 This chapter has been submitted to the demonstrations track at the 2020 Joint Eu-
471 ropean Software Engineering Conference and Symposium on the Foundations of
472 Software Engineering (ESEC/FSE) [88]. When integrating with a CVS, developers
473 need to select an appropriate confidence threshold suited to their use case and deter-
474 mine whether a decision should be made. An issue, however, is that these CVSs are
475 not calibrated to the specific problem-domain datasets and it is difficult for software
476 developers to determine an appropriate confidence threshold on their problem do-
477 main. This tool presents a workflow and supporting tool for application developers
478 to select decision thresholds suited to their domain that—unlike existing tooling—is
479 designed to be used in pre-development, pre-release and production. This tooling
480 forms part of a solution to RQ4 for developers to maintain robustness and reliability
481 in their systems.

482 *1.6.2.8 Chapter 11: Developing a CVS integration architecture*

483 This chapter has been accepted for presentation at the 2020 Joint European Software
484 Engineering Conference and Symposium on the Foundations of Software Engineer-
485 ing (ESEC/FSE) [89]. Based on the findings, we propose a set of new service error
486 codes for describing the empirically observed error conditions of IWS based on our

⁴⁸⁷ findings in Chapter 4. To achieve this, we propose a proxy server intermediary
⁴⁸⁸ that lies between a client application and a IWS; the proxy server tactic is designed
⁴⁸⁹ to return these error codes when substantial evolution occurs against a benchmark
⁴⁹⁰ dataset that represents the application domain context (similar to that proposed in
⁴⁹¹ Chapter 10). A technical evaluation of our implementation of this architecture iden-
⁴⁹² tifies 1,054 cases of substantial evolution in confidence values and 2,461 cases of
⁴⁹³ evolution in the response label sets when 331 images were sent to a CVS.

⁴⁹⁴ 1.6.3 Part III: Postface

⁴⁹⁵ In Chapter 12, we review the contributions made in this thesis and the relevance
⁴⁹⁶ and significance to identifying and resolving key issues when application developers
⁴⁹⁷ integrate with CVS. We evaluate these outcomes with reference to the research goals,
⁴⁹⁸ and discuss threats to validity of the work. Lastly, we discuss the various avenues
⁴⁹⁹ of research arising from this work. References from literature and a list of online
⁵⁰⁰ artefacts are provided after this concluding chapter.

⁵⁰¹ 1.6.4 Part IV: Appendices

⁵⁰² Appendix A provides additional material referenced within this thesis but not pro-
⁵⁰³ vided in the body. The source code for the reference architecture described in
⁵⁰⁴ Chapter 11 is reproduced in Appendix B. The supplementary materials published
⁵⁰⁵ with Chapter 8 are reproduced in Appendix C, which also describes the list of
⁵⁰⁶ primary sources arising in the systematic mapping study we conducted. We pro-
⁵⁰⁷ vide mandatory coauthor declaration forms describing the contribution breakdown
⁵⁰⁸ for each publication within Appendix D. Appendix E contains copies of the ethics
⁵⁰⁹ clearance for various experiments within this thesis.

⁵¹⁰ 1.7 Research Contributions

⁵¹¹ The outcomes of answering the four primary research questions elaborated in Sec-
⁵¹² tion 1.4 shapes three primary contributions this thesis offers to software engineering
⁵¹³ knowledge:

- ⁵¹⁴ • An **improved understanding in the landscape of CVSs**, with respect to their
⁵¹⁵ runtime behaviour and evolutionary profiles.
- ⁵¹⁶ • A **novel service integration architecture** that helps developers with integrat-
⁵¹⁷ ing their applications with CVSs.
- ⁵¹⁸ • A **key list of attributes that should be documented**, to assist CVS providers
⁵¹⁹ to better document their services.

¹³Conference publications ranking measured using the CORE Conference Ranks (<http://www.core.edu.au/conference-portal>) and Journal publications rankings using the Scimago Ranking (<https://www.scimagojr.com/>). Rankings retrieved January 2020.

¹⁴Date of publication, if applicable.

¹⁵The extended version of this conference proceeding is provided in Chapter 8.

¹⁶We abbreviate this with an added 'd' (for the demonstrations track) to distinguish this paper from our full FSE 2020 paper.

Table 1.5: List of publications resulting from this thesis, separated by phenomena exploration (above) and solution design (below).

Ref.	Venue	Acronym	Rank ¹³	Published ¹⁴	Chapter	RQs
[87]	35 th International Conference on Software Maintenance and Evolution	ICSME	A	05 Dec 2019	Chapter 4	RQ1
[86]	13 th International Symposium on Empirical Software Engineering and Measurement	ESEM	A	17 Oct 2019	Excluded ¹⁵	RQ2.1
[90]	42 nd International Conference on Software Engineering	ICSE	A*	In Press	Chapter 5	RQ3
[93]	IEEE Software	–	Q2	In Progress	Chapter 6	RQ3.2
[142]	35 th IEEE/ACM International Conference on Automated Software Engineering	ASE	A	In Progress	Chapter 7	RQ3.2
[91]	IEEE Transactions on Software Engineering	TSE	Q1	In Review	Chapter 8	RQ2
[260]	13 th International Conference on Web Engineering	ICWE	B	26 Apr 2019	Chapter 9	RQ4
[88]	28 th Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering	FSE(d) ¹⁶	A*	In Review	Chapter 10	RQ4
[89]	28 th Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering	FSE	A*	In Press	Chapter 11	RQ4

⁵²⁰ In this section, we detail how each publication forms a coherent body of work
⁵²¹ and how each publication relates to the primary contributions made.

⁵²² After our exploratory analysis on the nature of CVSs (Chapter 4), we proposed
⁵²³ two sets of recommendations targeted towards two stakeholders: (i) the service
⁵²⁴ *consumers* (i.e., application developers) and (ii) the service *providers*. Our sub-
⁵²⁵ sequent publications arose as a two-fold investigation to develop two strategies in
⁵²⁶ which developers and providers can, respectively, (i) better integrate these intelli-
⁵²⁷ gent components into their applications, and (ii) how these services can be better
⁵²⁸ documented. Table 1.5 provides a tabulated form of the publications and research
⁵²⁹ questions addressed within this thesis; for ease of reference, we refer to the publica-
⁵³⁰ tions in within this section in their abbreviated form as listed in Table 1.5. We also
⁵³¹ provide abbreviations for easier reference in this section. A high-level overview of
⁵³² the cohesiveness of our publications is provided in Figure 1.5.

⁵³³ 1.7.1 Contribution 1: Landscape Analysis & Preliminary Solutions

⁵³⁴ The first two bodies of work in this paper are the ICSME and ICWE papers. These
⁵³⁵ two works investigated a landscape analysis CVSs from two perspectives: firstly, we
⁵³⁶ conducted a longitudinal study to better understand the attributes associated with
⁵³⁷ these services (ICSME)—particularly their evolution and behavioural profiles, and
⁵³⁸ their potential impacts to software reliability—and tackled a preliminary solution
⁵³⁹ facade to ‘merge’ responses of the services together (ICWE).

⁵⁴⁰ The ICSME paper confirmed our hypotheses that the services have a non-

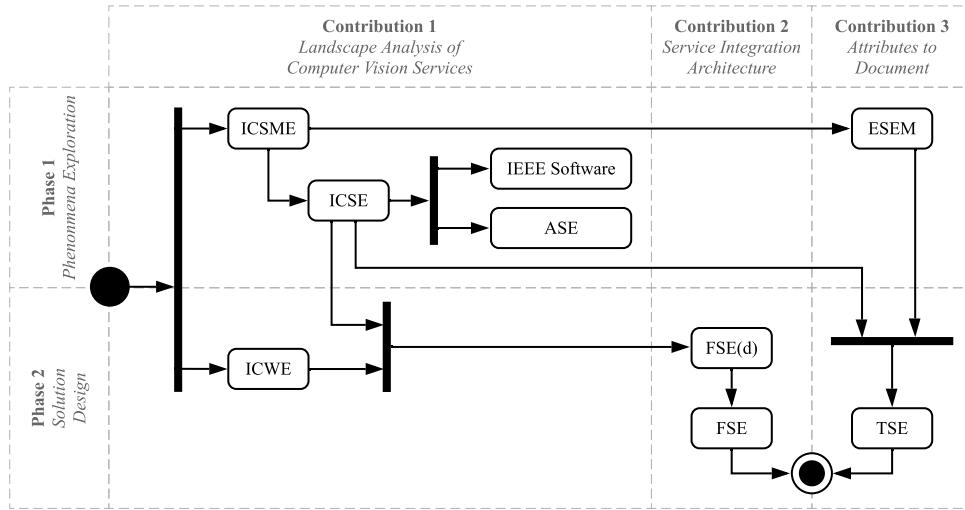


Figure 1.5: Activity diagram of the coherency of our publications, how our research was conducted, and relevant connections between publications. Our two-phase structure initial phenomena exploration and a proposed solutions to issues identified from the exploration. We map the contributions within each publication to the three primary contributions of the thesis.

541 deterministic behavioural profile, and that the evolution occurring within the ML
 542 models powering these services are not sufficiently communicated to software en-
 543 gineers. This therefore led to follow up investigation into how developers perceive
 544 these services, and thereby determine if they are frustrated due to this lack of com-
 545 munication.

546 Our ICWE paper explored one aspect identified from the ICSME paper that
 547 we identified early on: that different services use different vocabularies to describe
 548 semantically similar objects but in different ways (e.g., ‘border collie’ vs. ‘collie’),
 549 despite offering functionally similar capabilities. We attempted to merge the re-
 550 sponse labels from these services using a proportional representation approach, and
 551 upon comparison with more naive merge approaches, we improved label-merge per-
 552 formance by an F-measure of 0.015. However, while this was an interesting outcome
 553 for a preliminary solution design, investigation from our following work suggested
 554 that standardising ontologies between service providers becomes challenging and
 555 normalising the entire ontological hierarchy of response labels would need to fall
 556 under the responsibility of a certain body (that does not exist). Further, we did
 557 not find sufficient evidence that developers would frequently switch between service
 558 providers. Therefore, we opted for a shielded relay architecture in our later design
 559 work.

560 1.7.2 Contribution 2: Improving Documentation Attributes

561 As mentioned, our ICSME paper found that evolutionary and non-deterministic
 562 behavioural profile of are not adequately documented in pre-trained ML model APIs
 563 documentation, and further developers find this frustrating (Chapter 6) and potential

564 issues can arise as a result (Chapter 7). A recommendation concluding from this
565 work was that service providers should improve their documentation, however there
566 lacked a strategy by which they could do this, and our hypotheses that developers
567 were actually frustrated by this lack of communication was yet to be tested. This led
568 to two follow-up further investigations as presented in our ICSE and ESEM papers.

569 One aspect of our ICSE paper was to confirm whether developers are actually
570 frustrated with the service’s limited API documentation. By mining Stack Overflow
571 posts with reference to documentation issues, we adopted a 2019 documentation-
572 related taxonomy by Aghajani et al. [3] to classify posts, and found that 47.87%
573 of posts classified fell under the ‘completeness’ dimension of Aghajani et al.’s
574 taxonomy. This interpretation, therefore, warranted the recommendation proposed
575 in the ICSME paper to improve service documentation.

576 However, though improvements to more complete documentation was justified
577 from the ICSE paper, we needed to explore exactly *what* makes a ‘complete’ API
578 document. By conducting a systematic mapping study resulting in 4,501 results, we
579 curated 21 primary studies that outline the facets of API documentation knowledge.
580 From these studies, we distilled a documentation framework describing a priori-
581 tised order of the documentation assets API’s should document that is described
582 in our ESEM short paper. After receiving community feedback, we extended this
583 short paper with a follow-up experiment submitted to TSE. By conducting a sur-
584 vey with developers, we assessed our API documentation taxonomy’s efficacy with
585 practitioner opinions, thereby producing a weighted taxonomy against *both* literature
586 and developer sources. Lastly, we triangulated both weightings against a heuristic
587 evaluation against common CVS providers’ documentation. This allowed us to de-
588 duce which specific areas in existing CVS providers’ API documentation needed
589 improvement, which was a primary contribution from our TSE article.

590 1.7.3 Contribution 3: Service Integration Architecture

591 Two recommendations from our ICSME study encouraged developers to test their
592 applications with a representative ontology for their problem domain and to incorpo-
593 rate a specialised testing and monitoring techniques into their workflow. Strategies
594 on *how* to achieve this were explored in later studies. Following a similar approach
595 to our solution of improved API documentation, we validated the substantiveness of
596 our recommendations using our mining study of Stack Overflow (our ICSE paper)
597 to help inform us of generalised issues developers face whilst integrating CVSs into
598 their applications. To achieve this, we used a Stack Overflow post classification tax-
599 onomy proposed by Beyer et al. [39] into seven categories, where 28.9% and 20.37%
600 of posts asked issues regarding how to use the CVS API and conceptual issues be-
601 hind CVSs, respectively. Developers presented an insufficient understanding of the
602 non-deterministic runtime behaviour, functional capability, and limitations of these
603 services and are not aware of key computer vision terminology. When contrasted
604 to more conventional domains such as mobile-app development, the spread of these
605 issues vary substantially.

606 We proposed two technical solutions in our two FSE papers to help alleviate

607 this issue. Firstly, our FSE demonstrations paper—FSE(d) for short—provides a
608 workflow for developers to better select an appropriate confidence threshold, and
609 thus decision boundary, calibrated for their particular use case. In our ESEC/FSE
610 paper, we provide a reference architecture for developers to guard against the non-
611 deterministic issues that may ‘leak’ into their applications. This architecture tactic
612 proposes a client-server intermediary proxy server, similar to the style proposed in
613 our ICWE paper. However, unlike the ICWE paper that uses proportional repre-
614 sentation approach to modify multiple sources, our FSE paper proposes a guarded
615 relay, whereby a single service is used, and the proxy server maintains a lifecycle to
616 monitor evolution issues identified in ICSME and should be benchmarked against
617 the developer’s dataset (i.e., against the particular application domain) as suggested
618 in FSE(d). For robust component composition, this architecture tactic handles four
619 key requirements: (i) it clearly defines erroneous conditions that occur when evo-
620 lution occurs in CVSs; (ii) it notifies of behavioural changes in the service; (iii) it
621 monitors the service for change and substantial impact this may have to the client
622 application; and (iv) is flexible enough to be implemented and adaptable to any client
623 application or specific intelligent service to facilitate reuse. Both FSE papers serve
624 as two primary contributions to RQ4.

CHAPTER 2

625

626

627

Background

628

629 In Chapter 1, we defined a common set of (artificial) intelligence-based cloud ser-
630 vices that we label intelligent web services (IWSs). Specifically, we scope the
631 primary body of this study’s work on computer vision services (CVSs) (e.g., Google
632 Cloud Vision [411], AWS Rekognition [386], Azure Computer Vision [425], Watson
633 Visual Recognition [421] etc.). We claim developers have a distinctly determinis-
634 tic mindset (*2 + 2 always equals 4*) whereas an IWS’s ‘intelligence’ component (a
635 black box) may return probabilistic results (*2 + 2 might equal 4 with a confidence*
636 *of 95%*). Thus, there is a mindset mismatch between probabilistic results (from the
637 API provider) and results interpreted with certainty (from the API consumer).

638 What affect does this mindset mismatch have on the developer’s approach to-
639 wards building probabilistic software? What can we learn from common software
640 engineering practices (e.g., [282, 327]) that apply to resolve this mismatch and
641 thereby improve quality, such as verification & validation (V&V)? Chiefly, we an-
642 chor this question around three lenses of software engineering: creating an IWS,
643 using an IWS, and the nature of IWSs themselves.

644 Our chief concern lies with interaction and integration between IWS providers
645 and consumers, the nature of applications built using an IWS, and the impact this
646 has on software quality. We triangulate this around three pillars, which we diagram-
647 matically represent in Figure 2.1.

- 648 (1) **The development of the IWS.** We investigate the internal quality attributes
649 of creating an IWS from the IWS *provider’s* perspective. That is, we ask if
650 existing verification techniques are sufficient enough to ensure that the IWS
651 being developed actually satisfies the IWS consumer’s needs and if the internal
652 perspective of creating the system with a non-deterministic mindset clashes
653 with the outside perspective (i.e., pillar 2).
- 654 (2) **The usage of the IWS.** We investigate the external quality attributes of using
655 an IWS from the IWS *consumer’s* perspective. That is, we ask if existing
656 validation techniques are sufficient enough to ensure that the end-users can

657 actually use an IWS to build their software in the ways they expect the IWS to
 658 work.

659 **(3) The nature of an IWS.** We investigate what standard software engineering
 660 practices apply when developing non-deterministic systems. That is, we
 661 tackle what best practices exist when developing systems that are inherently
 662 stochastic and probabilistic, i.e., the ‘black box’ intelligence itself.

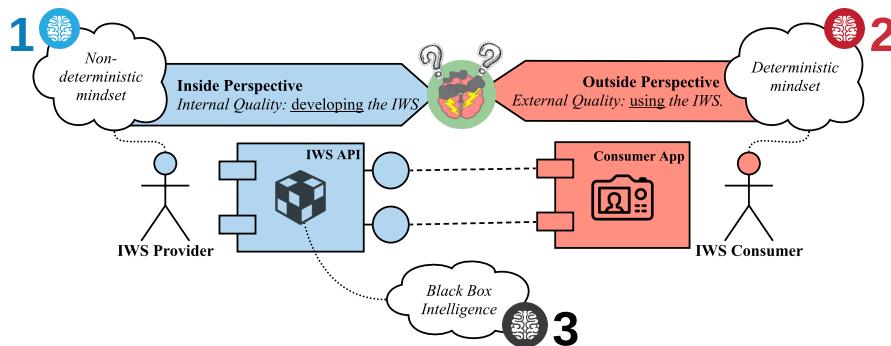


Figure 2.1: The three pillars by which we anchor the background: (1) developing an IWS with a non-deterministic mindset by the IWS provider; (2) the use of a IWS with a deterministic mindset by the IWS consumer; (3) the nature of a IWS itself.

663 Does a clash of deterministic consumer mindsets who use a IWS and the non-
 664 deterministic provider mindsets who develop them exist? And what impact does
 665 this have on the inside and outside perspective? Throughout this chapter, we will
 666 review these three core pillars due to such mindset mismatch from the anchoring per-
 667 spective of software quality, particularly around V&V and related quality attributes,
 668 probabilistic and nondeterministic software and the nature of APIs.

669 2.1 Software Quality

670 *Quality... you know what it is, yet you don't know what it is.*

ROBERT PIRSIG, 1974 [280]

671 The philosophical viewpoint of ‘quality’ remains highly debated and there are mul-
 672 tiple facets to perceive this complex concept [131]. Transcendentally, a viewpoint
 673 like that of Pirsig’s above shows that quality is not tangible but still recognisable; it’s
 674 hard to explicitly define but you know when it’s missing. The International Orga-
 675 nization for Standardization provides a breakdown of seven universally-applicable
 676 principles that defines quality for organisations, developers, customers and training
 677 providers [170]. More pertinently, the 1986 ISO standard for quality was simply
 678 “the totality of characteristics of an entity that bear on its ability to satisfy stated or
 679 implied needs” [169].

680 Using this sentence, what characteristics exist for non-deterministic IWSs like
681 that of a CVS? How do we know when the system has satisfied its ‘stated or implied
682 needs’ when the system can only give us uncertain probabilities in its outputs? Such
683 answers can be derived from related definitions—such as ‘conformance to specifica-
684 tion or requirements’ [85, 137], ‘meeting or exceeding customer expectation’ [36],
685 or ‘fitness for use’ [182]—but these then still depend on the solution description or
686 requirements specification, and thus the same questions still apply.

687 *Software* quality is somewhat more concrete. Pressman [282] adapted the
688 manufacturing-oriented view of quality from [37] and phrased software quality
689 under three core pillars:

- 690 • **effective software processes**, where the infrastructure that supports the cre-
691 ation of quality software needs is effective, i.e., poor checks and balances,
692 poor change management and a lack of technical reviews (all that lie in the
693 *process* of building software, rather than the software itself) will inevitably
694 lead to a poor quality product and vice-versa;
- 695 • **building useful software**, where quality software has fully satisfied the end-
696 goals and requirements of all stakeholders in the software (be it explicit or
697 implicit requirements) *in addition to* delivering these requirements in reliable
698 and error-free ways; and lastly
- 699 • **adding value to both the producer and user**, where quality software provides
700 a tangible value to the community or organisation using it to expedite a
701 business process (increasing profitability or availability of information) *and*
702 provides value to the software producers creating it whereby customer support,
703 maintenance effort, and bug fixes are all reduced in production.

704 In the context of a non-deterministic IWS, however, are any of the above actually
705 guaranteed? Given that the core of a system built using an IWS is fully dependent
706 on the *probability* that an outcome is true, what assurances must be put in place to
707 provide developers with the checks and balances needed to ensure that their software
708 is built with quality? For this answer, we re-explore the concept of verification &
709 validation (V&V).

710 2.1.1 Validation and Verification

711 To explain V&V, we analogously recount a tale given by Pham [278] on his works
712 on reliability. A high-school student sat a standardised test that was sent to 350,0000
713 students [339]. A multiple-choice algebraic equation problem used a variable, a ,
714 and intended that students *assume* that the variable was non-negative. Without
715 making this assumption explicit, there were two correct answers to the multiple
716 choice answer. Up to 45,000 students had their scores retrospectively boosted by up
717 to 30 points for those who ‘incorrectly’ answered, however, outcomes of a student’s
718 higher education were, thereby, affected by this one oversight in quality assessment.
719 The examiners wrote a poor question due to poor process standards to check if
720 their ‘correct’ answers were actually correct. The examiners “didn’t build the right
721 product” nor did they “build the product right” by writing an poor question and
722 failing to ensure quality standards, in the phrases Boehm [47] coined.

This story describes the issues with the cost of quality [46] and the importance of V&V: just as the poorly written exam question had such a high toll the 45,000 unlucky students, so does poorly written software in production. As summarised by Pressman [282], data sourced from Digital [79] in a large-scale application showed that the difference in cost to fix a bug in development versus system testing is \$6,159 per error. In safety-critical systems, such as self-driving cars or clinical decision support systems, this cost skyrockets due to the extreme discipline needed to minimise error [342].

Formally, we refer to the IEEE Standard Glossary of Software Engineering Terminology [166] for to define V&V:

<p>733 verification</p> <p>734</p> <p>735</p>	<p>The process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.</p>
<p>736 validation</p> <p>737</p> <p>738</p>	<p>The process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements.</p>

Thus, in the context of an IWS, we have two perspectives on V&V: that of the API provider and consumer (Figure 2.2).

The verification process of API providers ‘leak’ out to the context of the developer’s project dependent on the IWS. Poor verification in the *internal quality* of the IWS will entail poor process standards, such as poor definitions and terminology used, support tooling and description of documentations [327]. Though it is commonplace for providers to have a ‘ship-first-fix-later’ mentality of ‘good-enough’ software [354], the consequence of doing so leads to consumers absorbing the cost. Thus API providers must ensure that their verification strategies are rigorous enough for the consumers in the myriad contexts they wish to use it in. Studies have considered V&V in the context of web services on the cloud [20, 69, 70, 119, 155, 250, 252, 376], though little have recently considered how adding ‘intelligence’ to these services affects existing proposed frameworks and solutions. For a CVS, what might this entail? Which assurances are given to the consumers, and how is that information communicated? To verify if the service is working correctly, does that mean that we need to deploy the system first to get a wider range of data, given the stochastic nature of the black box?

Likewise, the validation perspective comes from that of the consumer. While the former perspective is of creation, this perspective comes from end-user (developer) expectation. As described in Chapter 1, a developer calls the IWS component using an API endpoint. Again, the mindset problem arises; does the developer know what to expect in the output? What are their expectations for their specific context? In the area of non-deterministic systems of probabilistic output, can the developer be assured that what they enter in a testing phase outcome the same result when in production?

Therefore, just as the test answers with were both correct and incorrect at the same time, so is the same with IWSs returning a probabilistic result: no result is

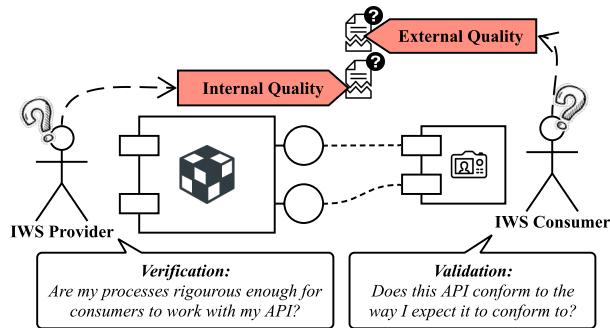


Figure 2.2: The ‘leakage’ of internal quality into the API consumer’s product and external quality imposing on the API provider.

766 certain. While V&V has been investigated in the area of mathematical and earth
 767 sciences for numerical probabilistic models and natural systems [262, 305], from
 768 the software engineering literature, little work has been achieved to look at the
 769 surrounding area of probabilistic systems hidden behind API calls.

770 Now that a developer is using a probabilistic system behind a deterministic API
 771 call, what does it mean in the context of V&V? Do current verification approaches
 772 and tools suffice, and if not, how do we fix it? From a validation perspective of
 773 ML and end-users, after a model is trained and an inference is given and if the
 774 output data point is incorrect, how will end users report a defect in the system?
 775 Compared to deterministic systems where such tooling as defect reporting forms are
 776 filled out (i.e., given input data in a given situation and the output data was X), how
 777 can we achieve similar outputs when the system is not non-deterministic? A key
 778 problem with the probabilistic mindset is that once a model is ‘fixed’ by retraining
 779 it, while one data-point may be fixed, others may now have been effected, thereby
 780 not ensuring 100% validation. Thus, due to the unpredictable and blurry nature of
 781 probabilistic systems, V&V must be re-thought out extensively.

782 2.1.2 Quality Attributes and Models

783 Similarly, quality models are used to capture internal and external quality attributes
 784 via measurable metrics. Is a similar issue reflected from that of V&V due to
 785 nondeterministic systems? As there is no ‘one’ definition of quality, there have been
 786 differing perspectives with literature placing varying value on disparate attributes.

787 Quality attribute assessment models (like those shown in Figure 2.3) are an early
 788 concept in software engineering, and systematically evaluating software quality
 789 appears as early as 1968 [304]. Rubey and Hartwick’s 1968 study introduced the
 790 phrase ‘attributes’ as a “prose expression of the particular quality of desired software”
 791 (as worded by Boehm et al. [45]) and ‘metrics’ as mathematical parameters on a
 792 scale of 0 to 100. Early attempts to categorise wider factors under a framework was
 793 proposed by McCall, Richards, and Walters in the late 1970s [73, 230]. This model
 794 described quality from the three perspectives of product revision (*how can we keep*

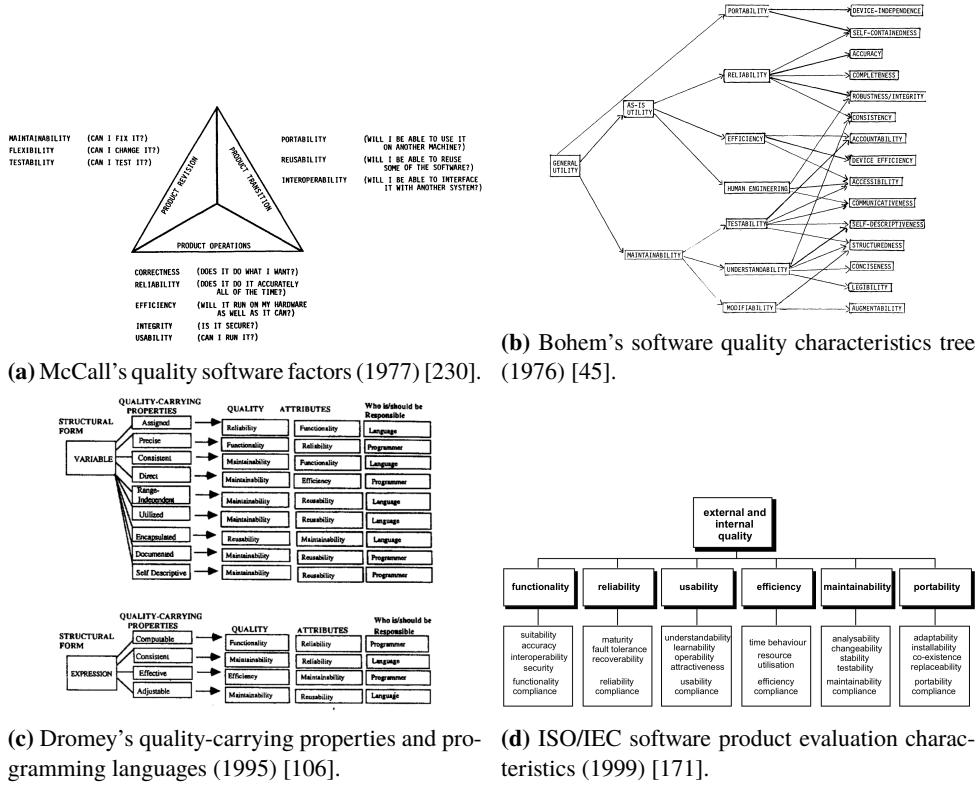


Figure 2.3: A brief overview of the development of software quality models since 1977.

795 *the system operational?), transition (how can we migrate the system as needed?)* and operation (*how effective is the system at achieving its tasks?*) (Figure 2.3a).
 796 The model also introduced 11 attributes alongside numerous direct and indirect
 797 measures to help quantify quality. This model was further developed by Boehm
 798 et al. [45] who independently developed a similar model, starting with an initial set
 800 of 11 software characteristics. It further defined candidate measurements of Fortran
 801 code to such characteristics, taking shape in a tree-like structure as in Figure 2.3b.
 802 In the mid-1990s, Dromey's interpretation [106] defined a set of quality-carrying
 803 properties with structural forms associated to specific programming languages and
 804 conventions (Figure 2.3c). The model also supported quality defect identification
 805 and proposed an improved auditing method to automate defect detection for code
 806 editors in IDEs. As the need for quality models became prevalent, the International
 807 Organization for Standardization standardised software quality under ISO/IEC-9126
 808 [171] (the Software Product Evaluation Characteristics, Figure 2.3d), which has since
 809 recently been revised to ISO/IEC-25010 with the introduction of the Systems and
 810 software Quality Requirements and Evaluation (SQuaRE) model [168], separating
 811 quality into *Product Quality* (consisting of eight quality characteristics and 31 sub-
 812 characteristics) and *Quality In Use* (consisting of five quality characteristics and 9
 813 sub-characteristics). An extensive review on the development of quality models in
 814 software engineering is given in [6].

815 Of all the models described, there is one quality attribute that relates most
 816 with our narrative of IWS quality: reliability. Reliability is the primary quality
 817 factor investigated within this thesis (see Section 1.4). Both McCall and Boehm's
 818 quality models have sub-characteristics of reliability relating to the primary research
 819 questions that investigate the *robustness*, *consistency* and *completeness*¹ of CVSs
 820 and its associated documentation. Moreover, the definition of reliability is similar
 821 among all quality models:

- 822 **McCall et al.** Extent to which a program can be expected to perform its in-
 823 tended function with required precision [230].
- 824 **Boehm et al.** Code possesses the characteristic *reliability* to the extent that
 825 it can be expected to perform its intended functions satisfac-
 826 torily [45].
- 827 **Dromey** Functionality implies reliability. The reliability of software is
 828 therefore dependent on the same properties as functionality, that
 829 is, the correctness properties of a program [106].
- 830 **ISO/IEC-9126** The capability of the software product to maintain a specified
 831 level of performance when used under specified conditions [171].

832 These definitions strongly relate to the system's solution description in that
 833 reliability is the ability to maintain its *functionality* under given conditions. But what
 834 defines reliability when the nature of an IWS in itself is inherently unpredictable
 835 due to its probabilistic implementation? Can a non-deterministic system ever be
 836 considered reliable when the output of the system is uncertain? How do developers
 837 perceive these quality aspects of reliability in the context of such systems? A system
 838 cannot be perceived as 'reliable' if the system cannot reproduce the same results due
 839 to a probabilistic nature. Therefore, we believe the literature of quality models does
 840 not suffice in the context of IWS reliability; a CVS can interpret an image of a dog
 841 as a 'Dog' one day, but what if the next it interprets such image more specifically to
 842 the breed, such as 'Border Collie'? Does this now mean the system is unreliable?

843 Moreover, defining these systems in themselves is challenging when require-
 844 ments specifications and solution descriptions are dependent on nondeterministic
 845 and probabilistic algorithms. We discuss this further in Section 2.2.

846 2.1.3 Reliability in Computer Vision

847 Testing computer vision deep-learning reliability is an area explored typically
 848 through the use of adversarial examples [337]. These input examples are where
 849 images are slightly perturbed to maximise prediction error but are still interpretable
 850 to humans. Refer to Figure 2.4.

¹In McCall's model, completeness is a sub-characteristic of the 'correctness' quality factor; however in Boehm's model it is a sub-characteristic of reliability. For consistency in this thesis, *completeness* is referred in the Boehm interpretation.

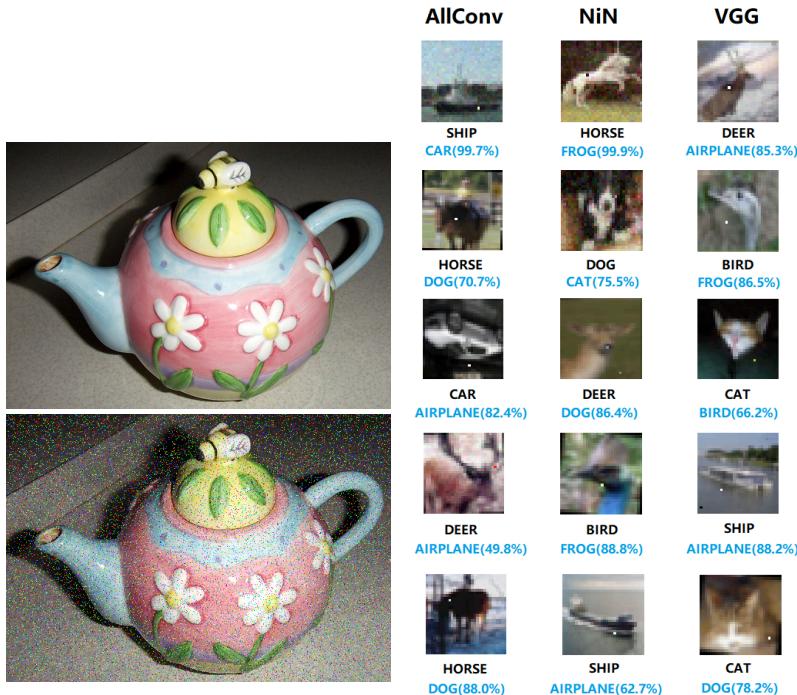


Figure 2.4: Sample adversarial examples in state-of-the-art computer vision studies.

851 Google Cloud Vision, for instance, fails to correctly classify adversarial examples
 852 when noise is added to the original images [161]. Rosenfeld et al. [302] illustrated
 853 that inserting synthetic foreign objects to input images (e.g., a cartoon elephant)
 854 can alter classification output. Wang et al. [357] performed similar attacks on a
 855 transfer-learning approach of facial recognition by modifying pixels of a celebrity's
 856 face to be recognised as a different celebrity, all while still retaining the same human-
 857 interpretable original celebrity. Su et al. [332] used the ImageNet database to show
 858 that 41.22% of images drop in confidence when just a *single pixel* is changed in the
 859 input image; and similarly, Eykholt et al. [113] recently showed similar results that
 860 made a CNN interpret a stop road-sign (with mimicked graffiti) as a 45mph speed
 861 limit sign.

862 Thus, the state-of-the-art computer vision techniques may not be reliable enough
 863 for safety critical applications (such as self-driving cars) as they do not handle inten-
 864 tional or unintentional adversarial attacks. Moreover, as such adversarial examples
 865 exist in the physical world [113, 203], "the real world may be adversarial enough"
 866 [277] to fool such software.

867 2.2 Probabilistic and Nondeterministic Systems

868 Probabilistic and nondeterministic systems are those by which, for the same given
 869 input, different outcomes may result. The underlying models that power an IWS
 870 are treated as though they are nondeterministic; Chapter 2 introduces IWSs as
 871 essentially black-box behaviour that can change over time. As such, we adopt the
 872 nondeterministic behaviour that they present.

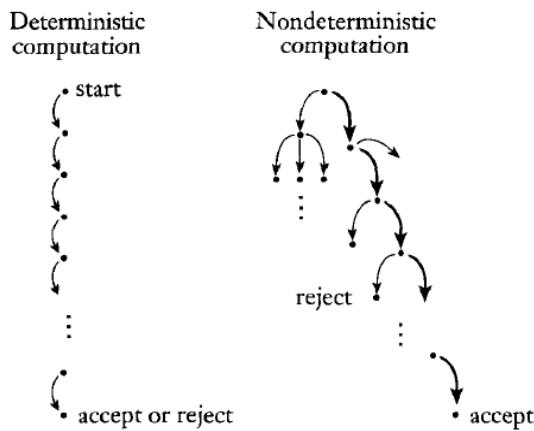


Figure 2.5: A deterministic system (left) always returns the same result in the same amount of steps. A nondeterministic system does not guarantee the same outcome, even with the same input data. Source: [117].

873 2.2.1 Interpreting the Uninterpretable

874 As the rise of applied AI increases, the need for engineering interpretability around
 875 models becomes paramount, chiefly from an external quality perspective that the
 876 *reliability* of the system can be inspected by end-users. Model interpretability has
 877 been stressed since early machine learning research in the late 1980s and 1990s (such
 878 as Quinlan [284] and Michie [241]), and although there has since been a significant
 879 body of work in the area [18, 34, 53, 66, 97, 115, 125, 135, 180, 212, 216, 227, 272,
 880 290, 303, 324, 352, 355], it is evident that ‘accuracy’ or model ‘confidence’ is still
 881 used as a primary criterion for AI evaluation [164, 174, 326]. Much research into
 882 neural network (NN) or support vector machine (SVM) development stresses that
 883 ‘good’ models are those with high accuracy. However, is accuracy enough to justify
 884 a model’s quality?

885 To answer this, we revisit what it means for a model to be accurate. Accuracy
 886 is an indicator for estimating how well a model’s algorithm will work with future
 887 or unforeseen data. It is quantified in the AI testing stage, whereby the algorithm
 888 is tested against cases known by humans to have ground truth but such cases are
 889 unknown by the algorithm. In production, however, all cases are unknown by both
 890 the algorithm *and* the humans behind it, and therefore a single value of quality is “not
 891 reliable if the future dataset has a probability distribution significantly different from
 892 past data” [121], a problem commonly referred to as the *datashift* problem [309].
 893 Analogously, Freitas [121] provides the following description of the problem:

894 *The military trained [a NN] to classify images of tanks into enemy
 895 and friendly tanks. However, when the [NN] was deployed in the field
 896 (corresponding to “future data”), it had a poor accuracy rate. Later,
 897 users noted that all photos of friendly (enemy) tanks were taken on a
 898 sunny (overcast) day. I.e., the [NN] learned to discriminate between
 899 the colors of the sky in sunny vs. overcast days! If the [NN] had
 900 output a comprehensible model (explaining that it was discriminating
 901 between colors at the top of the images), such a trivial mistake would
 902 immediately be noted.* [121]

903 So, why must we interpret models? While the formal definition of what it means
 904 to be *interpretable* is still somewhat disparate (though some suggestions have been
 905 proposed [216]), what is known is (i) there exists a critical trade-off between accuracy
 906 and interpretability [102, 120, 144, 179, 186, 378], and (ii) a single quantifiable value
 907 cannot satisfy the subjective needs of end-users [121]. As ever-growing domains
 908 ML become widespread², these applications engage end-users for real-world goals,
 909 unlike the aims in early ML research where the aim was to get AI working in the
 910 first place. In safety-critical systems where AI provide informativeness to humans
 911 to make the final call (see [71, 165, 189]), there is often a mismatch between the
 912 formal objectives of the model (e.g., to minimise error) and complex real-world
 913 goals, where other considerations (such as the human factors and cognitive science

²In areas such as medicine [33, 66, 111, 175, 180, 207, 273, 292, 352, 374, 381], bioinformatics [101, 122, 177, 185, 336], finance [18, 99, 165] and customer analytics [212, 355].

behind explanations³) are not realised: model optimisation is only worthwhile if they “actually solve the original [human-centred] task of providing explanation” [251] to end-users. **Therefore, when human-decision makers must be interpretable themselves [293], any AI they depend on must also be interpretable.**

Recently, discussion behind such a notion to provide legal implications of interpretability is topical. Doshi-Velez et al. [105] discuss when explanations are not provided from a legal stance—for instance, those affected by algorithmic-based decisions have a ‘right to explanation’ [224, 356] under the European Union’s GDPR⁴. But, explanations are not the only way to ensure AI accountability: theoretical guarantees (mathematical proofs) or statistical evidence can also serve as guarantees [105], however, in terms of explanations, what form they take and how they are proven correct are still open questions [216].

2.2.2 Explanation and Communication

From a software engineering perspective, explanations and interpretability are, by definition, inherently communication issues: what lacks here is a consistent interface between the AI system and the person using it. The ability to encode ‘common sense reasoning’ [231] into programs today has been achieved, but *decoding* that information is what still remains problematic. At a high level, Shannon and Weaver’s theory of communication [317] applies, just as others have done with similar issues in the software engineering realm [244, 368] (albeit to the domain of visual notations). Humans map the world in higher-level concepts easily when compared to AI systems: while we think of a tree first (not the photons of light or atoms that make up the tree), an algorithm simply sees pixels, and not the concrete object [105] and the AI interprets the tree inversely to humans. Therefore, the interpretation or explanation is done inversely: humans do not explain the individual neurons fired to explain their predictions, and therefore the algorithmic transparent explanations of AI algorithms (“*which neurons were fired to make this AI think this tree is a tree?*”) do not work here.

Therefore, to the user (as mapped using Shannon and Weaver’s theory), an AI pipeline (the communication *channel*) begins with a real-world concept, y , that acts as an *information source*. This information source is fed in as a *message*, x , (as pixels) to an AI system (the *transmitter*). The transmitter encodes the pixels to a prediction, \hat{y} , the *signal* of the message. This signal is decoded by the *receiver*, an explanation system, $e_x(x, \hat{y})$, that tailors the prediction with the given input data to the intended end user (the *destination*) as an explanation, \tilde{y} , another type of *message*. Therefore, the user only sees the channel as an input/output pipeline of real-world objects, y , and explanations, \tilde{y} , tailored to *them*, without needing to see the inner-mechanics of a prediction \hat{y} . We present this diagrammatically in Figure 2.6.

³Interpretations and explanations are often used interchangeably.

⁴<https://www.eugdpr.org> last accessed 13 August 2018.

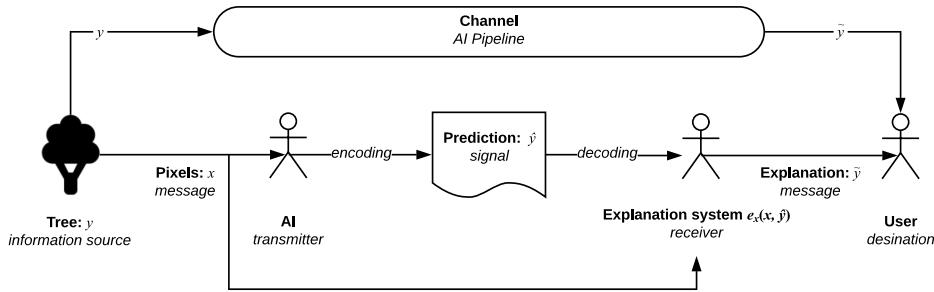


Figure 2.6: Theory of AI communication from information source, y , to intended user as explanations \tilde{y} .

952 2.2.3 Mechanics of Model Interpretation

953 How do we interpret models? Methods for developing interpretation models include:
 954 decision trees [59, 83, 152, 221, 285], decision tables [19, 212] and decision sets
 955 [205, 251]; input gradients, gradient vectors or sensitivity analysis [18, 209, 290,
 956 303, 314]; exemplars [123, 190]; generalised additive models [71]; classification
 957 (*if-then*) rules [55, 80, 265, 346, 371] and falling rule lists [324]; nearest neighbours
 958 [227, 286, 315, 366, 379] and Naïve Bayes analysis [33, 75, 114, 124, 156, 197, 207,
 959 381].

960 Cross-domain studies have assessed the interpretability of these techniques
 961 against end-users, measuring response time, accuracy in model response and user
 962 confidence [7, 122, 153, 165, 227, 308, 333, 355], although it is generally agreed
 963 that decision rules and decision tables provide the most interpretation in non-linear
 964 models such as SVMs or NNs [122, 227, 355]. For an extensive survey of the benefits
 965 and fallbacks of these techniques, we refer to Freitas [121], Doshi-Velez et al. [105]
 966 and Doshi-Velez and Kim [104].

967 2.3 Application Programming Interfaces

968 Application programming interfaces (APIs) are the interface between a developer
 969 needs and the software components at their disposal [13] by abstracting the underlying-
 970 component behind a subroutine, protocol or specific tool. Therefore, it is natural
 971 to assess internal quality (and external quality if the software is in itself a service to
 972 be used by other developers—in this case an IWS) is therefore directly related to the
 973 quality the API offers [196].

974 Good APIs are known to be intuitive and require less documentation browsing
 975 [279], thereby increasing developer productivity. Conversely, poor APIs are those
 976 that are hard to interpret, thereby reducing developer productivity and product quality.
 977 The consequences of this have shown a higher demand of technical support (as
 978 measured in [157]) that, ultimately, causes the maintenance to be far more expensive,
 979 a phenomenon widely known in software engineering economics (see Section 2.1.1).

980 While there are different types of APIs, such as software library/framework

981 APIs for building desktop software, operating system APIs for interacting with the
982 operating system, remote APIs for communication of varying technologies through
983 common protocols, we focus on web APIs for communication of resources over
984 the web (being the common architecture of cloud-based services). Further infor-
985 mation on the development, usage and documentation of web APIs is provided in
986 Appendix A.1.

987 **2.3.1 API Usability**

988 If a developer doesn't understand the overarching concepts of the context behind
989 the API they wish to use, then they cannot formulate what gaps in their knowledge
990 is missing. For example, a developer that knows nothing about ML techniques in
991 computer vision cannot effectively formulate queries to help bridge those gaps in
992 their understanding to figure out more about the CVS they wish to use.

993 Balancing the understanding of the information need (both conscious and un-
994 conscious), how to phrase that need and how to query it in an information retrieval
995 system is concept long studied in the information sciences [344]. In API design,
996 the most common form to convey knowledge to developers is through annotated
997 code examples and overviews to a platform's architectural and design decisions
998 [56, 103, 248, 297] though these studies have not effectively communicated *why*
999 these artefacts are important. What makes the developer *conceptually understand*
1000 these artefacts?

1001 Robillard and Deline [297] conducted a multi-phase, mixed-method approach to
1002 create knowledge grounded in the professional experience of 440 software engineers
1003 at Microsoft of varying experience to determine what makes APIs hard to learn,
1004 the results of which previously published in an earlier report [296]. Their results
1005 demonstrate that 'documentation-related obstacles' are the biggest hurdle in learning
1006 new APIs. One of these implications are the *intent documentation* of an API (i.e.,
1007 *what is the intent for using a particular API?*) and such documentation is required
1008 only where correct API usage is not self-evident, where advanced uses of the API are
1009 documented (but not the intent), and where performance aspects of the API impact
1010 the application developed using it. They conclude that professional developers do
1011 not struggle with learning the *mechanics* of the API, but in the *understanding* of how
1012 the API fits in upwards to its problem domain and downward to its implementation:

1013 *In the upwards direction, the study found that developers need help*
1014 *mapping desired scenarios in the problem domain to the content of the*
1015 *API, and in understanding what scenarios or usage patterns the API*
1016 *provider intends and does not intend to support. In the downwards*
1017 *direction, developers want to understand how the API's implementation*
1018 *consumes resources, reports errors and has side effects. [297]*

1019 These results particularly corroborate to that of previous studies where devel-
1020 opers quote that they feel that existing learning content currently focuses on "how
1021 to do things, not necessarily why" [259]. This thereby reiterates the conceptual
1022 understanding of an API as paramount.

¹⁰²³ A later study by Ko and Riche [195] assessed the importance of a programmer's
¹⁰²⁴ conceptual understanding of the background behind the task before implementing the
¹⁰²⁵ task itself, a notion that we find most relevant for users of IWS APIs. While the study
¹⁰²⁶ did not focus on developing web APIs (rather implementing a Bluetooth application
¹⁰²⁷ using platform-agnostic terminology), the study demonstrated how developers show
¹⁰²⁸ little confidence in their own metacognitive judgements to understand and assess the
¹⁰²⁹ feasibility of the intent of the API and understand the vocabulary and concepts within
¹⁰³⁰ the domain (i.e., wireless connectivity). This indecision over what search results
¹⁰³¹ were relevant in their searches ultimately hindered their progress implementing the
¹⁰³² functionality, again decreasing productivity. Ko and Riche suggest to improve API
¹⁰³³ usability by introducing the background of the API and its relevant concepts using
¹⁰³⁴ glossaries linked to tutorials to each of the major concepts, and then relate it back to
¹⁰³⁵ how to implement the particular functionality.

¹⁰³⁶ Thus, an analysis of the conceptual understanding of IWS APIs by a range of
¹⁰³⁷ developers (from beginner to professional) is critical to best understand any differ-
¹⁰³⁸ ences between existing studies and those that are nondeterministic. Our proposal is
¹⁰³⁹ to perform similar survey research (see Chapter 3) in the search for further insight
¹⁰⁴⁰ into the developer's approach toward existing IWS APIs.

CHAPTER 3

1041

1042

1043

Research Methodology

1044

1045 Investigating software engineering practices is often a complex task as it is imper-
1046 ative to understand the social and cognitive processes around software engineers
1047 and not just the tools and processes used [109]. This chapter explores our research
1048 methodology by exploring five key elements of empirical software engineering re-
1049 search: firstly, (i) we provide an extended focus to the study by reviewing our research
1050 questions (see Section 1.4) anchored under the context of an existing research ques-
1051 tion classification taxonomy, (ii) characterise our research goals through an explicit
1052 philosophical stance, (iii) explain how the stance selected impacts our selection of
1053 research methods and data collection techniques (by dissecting our choice of meth-
1054 ods used to reach these research goals), (iv) discuss a set of criteria for assessing the
1055 validity of our study design and the findings of our research, and lastly (v) discuss
1056 the practical considerations of our chosen methods.

1057 The foundations for developing this research methodology has been expanded
1058 from that proposed by Easterbrook et al. [109], Wohlin and Aurum [372], Wohlin
1059 et al. [373] and Shaw [319].

1060 3.1 Research Questions Revisited

1061 To discuss our research strategy, we revisit our four primary and seven secondary
1062 research questions (RQs) through the classification technique discussed by Easter-
1063 brook et al. [109], a technique originally proposed in the field of psychology by
1064 Meltzoff and Cooper [236] but adapted to software engineering. A summary of the
1065 classifications made to our research questions are presented in Table 3.1.

1066 Our research study involves a mix of nine *empirical*¹ RQs, that focus on observ-
1067 ing and analysing existing phenomena, and two *non-empirical* RQs, that focuses
1068 on designing better approaches to solve software engineering tasks [240]. The use

¹Or ‘knowledge’ questions, that extend our *knowledge* on certain phenomena.

¹⁰⁶⁹ of empirical *and* non-empirical RQs are best combined in long-term software engineering research studies where the phenomena are under-explored, as is the case ¹⁰⁷⁰ with CVSs. Further, these approaches help propose solutions to issues found in the ¹⁰⁷¹ phenomena studied [369]. We discuss both our empirical and non-empirical RQs in ¹⁰⁷² Sections 3.1.1 and 3.1.2 below. ¹⁰⁷³

Table 3.1: A summary of our research questions classified using the strategies presented by Easterbrook et al. [109] and Meltzoff and Cooper [236].

#	RQ	Primary/ Secondary	RQ Classification
RQ1	What is the nature of cloud-based CVSs?	Primary Secondary Secondary	EMPIRICAL ↪ Exploratory ↪ Description/Classification
RQ1.1	What is their runtime behaviour?		EMPIRICAL ↪ Exploratory ↪ Description/Classification
RQ1.2	What is their evolution profile?		EMPIRICAL ↪ Exploratory ↪ Description/Classification
RQ2	Are CVS APIs sufficiently documented?	Primary	EMPIRICAL ↪ Exploratory ↪ Existence
RQ2.1	What are the dimensions of a ‘complete’ API document, according to both literature and practitioners?	Secondary	EMPIRICAL ↪ Exploratory ↪ Composition
RQ2.2	What additional information or attributes do application developers need in CVS API documentation to make it more complete?	Secondary	NON-EMPIRICAL ↪ Design
RQ3	Are CVSs more misunderstood than conventional software engineering domains?	Primary	EMPIRICAL ↪ Exploratory ↪ Descriptive-Comparative
RQ3.1	What types of issues do application developers face most when using CVSs, as expressed as questions on Stack Overflow?	Secondary	EMPIRICAL ↪ Base-Rate ↪ Frequency/Distribution
RQ3.2	Which of these issues are application developers most frustrated with?	Secondary	EMPIRICAL ↪ Exploratory ↪ Description/Classification
RQ3.3	Is the distribution CVS pain-points different to established software engineering domains, such as mobile or web development?	Secondary	EMPIRICAL ↪ Base-Rate ↪ Frequency/Distribution
RQ4	What strategies can developers employ to integrate their applications with CVSs while preserving robustness and reliability?	Primary	NON-EMPIRICAL ↪ Design

¹⁰⁷⁴ 3.1.1 Empirical Research Questions

¹⁰⁷⁵ In total, we pose nine empirically-based RQs to help us understand the way developers ¹⁰⁷⁶ currently interact and work with web services that provide computer vision. The ¹⁰⁷⁷ majority of these questions are *exploratory* questions that contribute to a landscape ¹⁰⁷⁸ analysis of these services (RQ1, RQ1.1 and RQ1.2), how well they are documented ¹⁰⁷⁹ (RQ2), and the issues developers currently face when using them (RQ3). Our other

1080 exploratory questions complement the answers to these questions. For instance, to
1081 understand if CVSs are sufficiently documented (an *existence* exploratory question
1082 posed in RQ2), we need to understand the components of a ‘sufficient’ or ‘com-
1083 plete’ API document via RQ2.1 as proposed in both the literature and by software
1084 developers. While RQ2.1 does not directly relate to CVSs, answering it gives us
1085 an understanding the components of complete API documentation, and therefore,
1086 we can assess what aspects they are missing and where improvements can be made
1087 (RQ2.2). These questions are *descriptive and classification* questions that help de-
1088 scribe and classify what practices are in use for existing CVS API documentation
1089 and the nature behind these services. Answering these exploratory questions assists
1090 in refining preciser terms of the phenomena, ways in which we find evidence for
1091 them, and ensuring the data found is valid.

1092 By answering these questions, we have a clearer understanding of the phenom-
1093 ena; we then follow up by posing two additional *base-rate questions* that helps
1094 provide a basis to confirm that the phenomena occurring is normal (or unusual)
1095 behaviour by investigating the patterns of phenomena’s occurrence against other
1096 phenomena. RQ3.1 is a *frequency and distribution* question to help us understand
1097 what types of issues developers often encounter most, given a lack of formal extended
1098 training in artificial intelligence. This achieves us an insight into the developer’s
1099 mindset and regular thought patterns toward these APIs. We can then contrast
1100 this distribution using our second base-rate question (RQ3.3), that assesses the
1101 distributional differences between these intelligent components and non-intelligent
1102 (conventional) software components. Combined, these two questions can help us
1103 answer how the issues raised against CVSs are different to normal Stack Overflow
1104 issues—our *descriptive-comparative* question posed in RQ3—and, similarly, we can
1105 classify and rank which issues developers find most frustrating (RQ3.2).

1106 3.1.2 Non-Empirical Research Questions

1107 RQ2.2 and RQ4 are both non-empirically-based *design questions*; they are con-
1108 cerned with ways in which we can improve a CVS by investigating what additional
1109 attributes are needed in both the documentation of CVSs and in the integration
1110 architectures developers can employ to improve reliability and robustness in their
1111 applications. They are not classified as empirical questions as we investigate what
1112 *will be* and not *what is*. By understanding the process by which developers desire
1113 additional attributes of documentation and integration strategies, we can help shape
1114 improvements to the existing designs of using CVSs.

1115 3.2 Philosophical Stances

1116 Philosophical stances guide the researcher’s action by fortifying what constitutes
1117 ‘valid truth’ against a fundamental set of core beliefs [295]. In software engineer-
1118 ing, four dominant philosophical stances are commonly characterised [84, 275]:
1119 positivism (or post-positivism), constructivism (or interpretivism), pragmatism, and
1120 critical theory (or advocacy/participatory). To construct such a ‘validity of truth’,

1121 we will review these four philosophical stances in this section, and state the stance
1122 that we explicitly adopt and our reasoning for this.

1123 **3.2.0.1 Positivism**

1124 Positivists claim truth to be all observable facts, reduced piece-by-piece to smaller
1125 components which is incrementally verifiable to form truth. We do not base our
1126 work on the positivistic stance as the theories governing verifiable hypothesis must
1127 be precise from the start of the research. Moreover, due to its reductionist approach,
1128 it is difficult to isolate these hypotheses and study them in isolation from context.
1129 As our hypotheses are not context-agnostic, we steer clear from this stance.

1130 **3.2.0.2 Constructivism**

1131 Constructivists see knowledge embedded within the human context; truth is the
1132 *interpretive* observation by understanding the differences in human thought between
1133 meaning and action [194]. That is, the interpretation of the theory is just as important
1134 to the empirical observation itself. We partially adopt a constructivist stance as we
1135 attempt to model the developer's mindset, being an approach that is rich in qualitative
1136 data on human activity.

1137 **3.2.0.3 Pragmatism**

1138 Pragmatism is a less dogmatic approach that encourages the incomplete and approx-
1139 imate nature of knowledge and is dependent on the methods in which the knowledge
1140 was extracted. The utility of consensually agreed knowledge is the key outcome, and
1141 is therefore relative to those who seek utility in the knowledge—what is the useful
1142 for one person is not so for the other. While we value the utility of knowledge, it is
1143 difficult to obtain consensus especially on an ill-researched topic such as ours, and
1144 therefore we do not adopt this stance.

1145 **3.2.0.4 Critical Theory**

1146 This study chiefly adopts the philosophy of critical theory [10]. A key outcome of
1147 the study is to shift the developer's restrictive deterministic mindset and shed light
1148 on developing a new framework actively with the developer community that seeks
1149 to improve the process of using such APIs. In software engineering, critical theory
1150 is used to “actively [seek] to challenge existing perceptions about software practice”
1151 [109], and this study utilises such an approach to shift the mindset of CVS consumers
1152 and providers alike on how the documentation and metadata should not be written
1153 with the ‘traditional’ deterministic mindset at heart. Thus, our key philosophical
1154 approach is critical theory to seek out *what-can-be* using partial constructivism to
1155 model the current *what-is*.

3.3 Research Methods

1157 Research methods are “a set of organising principles around which empirical data is
1158 collection and analysed” [109]. Creswell [84] suggests that strong research design
1159 is reflected when the weaknesses of multiple methods complement each other. Us-
1160 ing a mixed-methods approach is therefore commonplace in software engineering
1161 research, typically due to the human-oriented nature investigating how software en-
1162 gineers work both individually (where methods from psychology may be employed)
1163 and together (where methods from sociology may be employed).

1164 Therefore, studies in software engineering are typically performed as field studies
1165 where researchers and developers (or the artefacts they produce) are analysed either
1166 directly or indirectly [323]. The mixed-methods approach combines five classes
1167 of field study methods (or empirical strategies/studies) most relevant in empirical
1168 software engineering research [109, 184, 373]: controlled experiments, case studies,
1169 survey research, ethnographies, and action research. We chiefly adopt a mixed-
1170 methods approach to our work using the *concurrent triangulation* mixed-methods
1171 strategy [229] as it best compensates for weaknesses that exist in all research methods,
1172 and employs the best strengths of others [84].

3.3.1 Review of Relevant Research Methods

1173 Below we review some of the research methods most relevant to our research ques-
1174 tions as refined in Section 3.1 as presented by Easterbrook et al. [109].

3.3.1.1 Controlled Experiments

1175 A controlled experiment is an investigation of a clear, testable hypothesis that guides
1176 the researcher to decide and precisely measure how at least one independent variable
1177 can be manipulated and effect at least one other dependent variable. They determine
1178 if the two variables are related and if a cause-effect relationship exists between
1179 them. The combination of independent variable values is a *treatment*. It is common
1180 to recruit human subjects to perform a task and measure the effect of a randomly
1181 assigned treatment on the subjects, though it is not always possible to achieve
1182 full randomisation in real-life software engineering contexts, in which case a *quasi-*
1183 *experiment* may be employed where subjects are not randomly assigned to treatments.

1184 While we have well-defined RQs, refining them into precise, *measurable* vari-
1185 ables is challenging due to the qualitative nature they present. A well-defined
1186 population is also critical and must be easily accessible; the varied range of begin-
1187 ner to expert software engineers with varied understanding of artificial intelligence
1188 concepts is required to perform controlled experiments, and thus recruitment may
1189 prove challenging. Lastly, the controlled experiment is essentially reductionist by
1190 affecting a small amount of variables of interest and controlling all others. This
1191 approach is too clinical for the practical outcomes by which our research goals aim
1192 for, and is therefore closely tied to the positivist stance.

1195 3.3.1.2 Case Studies

1196 Case studies investigate phenomena in their real-life context and are well-suited
1197 when the boundary between context and phenomena is unknown [377]. They offer
1198 understanding of how and why certain phenomena occur, thereby investigating ways
1199 cause-effect relationships can occur. They can be used to test existing theories
1200 (*confirmatory case studies*) by refuting theories in real-world contexts instead of
1201 under laboratory conditions or to generate new hypotheses and build theories during
1202 the initial investigation of some phenomena (*exploratory case studies*).

1203 Case studies are well-suited where the context of a situation plays a role in
1204 the phenomenon being studied. They also lend themselves to purposive sampling
1205 rather than random sampling, and thus it is possible to selectively choose cases that
1206 benefit our research goals and (using our critical theorist stance) select cases that
1207 will actively benefit our participant software engineering audience most to draw
1208 attention to situations regarded as problematic in CVS.

1209 3.3.1.3 Survey Research

1210 Survey research identifies characteristics of a broad population of individuals through
1211 direct data collection techniques such as interviews and questionnaires or indepen-
1212 dent techniques such as data logging. Defining that well-defined population is
1213 critical, and selecting a representative sample from it to generalise the data gathered
1214 usually assists in answering base-rate questions.

1215 By identifying representative sample of the population, from beginner to ex-
1216 perienced developers with varying understanding of CVS APIs, we can use survey
1217 research to assist in answering our exploratory and base-rate RQs (see Section 3.1.1)
1218 in determining the qualitative aspects of how individual developers perceive and
1219 work with the existing APIs, either by directly asking them, or by mining third-party
1220 discussion websites such as Stack Overflow (SO). Similarly, we can use this strategy
1221 to assess the developer’s understanding on what makes API documentation sufficient
1222 by assessing whether specific factors suggested from literature are useful according
1223 to developers. However, with direct survey research techniques, low response rates
1224 may prove challenging, especially if no inducements can be offered for participation.

1225 3.3.1.4 Ethnographies

1226 Ethnographies investigates the understanding of social interaction within community
1227 through field observation [299]. Resulting ethnographies help understand how soft-
1228 ware engineering technical communities build practices, communication strategies
1229 and perform technical work collaboratively.

1230 Ethnographies require the researcher to be highly trained in observational and
1231 qualitative data analysis, especially if the form of ethnography is participant observa-
1232 tion, whereby the researcher is embedded of the technical community for observation.
1233 This may require the longevity of the study to be far greater than a couple of weeks,
1234 and the researcher must remain part of the project for its duration to develop enough
1235 local theories about how the community functions. While it assists in revealing

1236 subtle but important aspects of work practices within software teams, this study
1237 does not focus on the study of teams, and is therefore not a research method relevant
1238 to this project.

1239 **3.3.1.5 Action Research**

1240 Action researchers simultaneously solve real-world problems while studying the
1241 experience of solving the problem [95] by actively seeking to intervene in the
1242 situation for the purpose of improving it. A precondition is to engage with a
1243 *problem owner* who is willing to collaborate in identifying and solving the problem
1244 faced. The problem must be authentic (a problem worth solving) and must have
1245 new knowledge outcomes for those involved. It is also characterised as an iterative
1246 approach to problem solving, where the knowledge gained from solving the problem
1247 has a desirable solution that empowers the problem owner and researcher.

1248 This research is most associated to our adopted philosophical stance of critical
1249 theory. As this project is being conducted under the Applied Artificial Intelligence
1250 Institute (A^2I^2) collaboratively with engaged industry clients, we have identified a
1251 need for solving an authentic problem that industry faces. The desired outcome
1252 of this project is to facilitate wider change in the usage and development of CVSs;
1253 thus, engaging action research as a potential method throughout the mixed-methods
1254 approach is used in this research.

1255 **3.3.2 Review of Data Collection Techniques for Field Studies**

1256 Singer et al. developed a taxonomy [210, 323] showcasing data collection techniques
1257 in field studies that are used in conjunction with a variety of methods based on the
1258 level of interaction between researcher and software engineer, if any. This taxonomy
1259 is reproduced in Figure 3.1.

1260 **3.4 Research Design**

1261 This section discusses an overview of the design of methods used within the experi-
1262 ments conducted under this thesis. For each experiment, we describe an overview of
1263 the experiment grounded known methods and techniques (Sections 3.3.1 and 3.3.2)
1264 and our approach to analysing the data, as well as relating the selecting method back
1265 to a specific RQ. Details of each experiment presented in this thesis, the coherency
1266 between them, and where they can be found are given in Sections 1.6 and 1.7.

1267 **3.4.1 Landscape Analysis of Computer Vision Services**

1268 To understand the behavioural and evolutionary profiles of CVSs (i.e., RQ1), we em-
1269 ployed a longitudinal study based around a dynamic system analysis [323]. Specif-
1270 ically, we used structured observations of three services using the same dataset to
1271 understand how the responses from these services change with time. Lastly, we

Figure 3.1: Questions asked by software engineering researchers (column 2) that can be answered by field study techniques. (From [323].)

Technique	Used by researchers when their goal is to understand:	Volume of data	Also used by software engineers for
Direct techniques			
Brainstorming and focus groups	Ideas and general background about the process and product, general opinions (also useful to enhance participant rapport)	Small	Requirements gathering, project planning
Interviews and questionnaires	General information (including opinions) about process, product, personal knowledge etc.	Small to large	Requirements and evaluation
Conceptual modeling	Mental models of product or process	Small	Requirements
Work diaries	Time spent or frequency of certain tasks (rough approximation, over days or weeks)	Medium	Time sheets
Think-aloud sessions	Mental models, goals, rationale and patterns of activities	Medium to large	UI evaluation
Shadowing and observation	Time spent or frequency of tasks (intermittent over relatively short periods), patterns of activities, some goals and rationale	Small	Advanced approaches to use case or task analysis
Participant observation (joining the team)	Deep understanding, goals and rationale for actions, time spent or frequency over a long period	Medium to large	
Indirect techniques			
Instrumenting systems	Software usage over a long period, for many participants	Large	Software usage analysis
Fly on the wall	Time spent intermittently in one location, patterns of activities (particularly collaboration)	Medium	
Independent techniques			
Analysis of work databases	Long-term patterns relating to software evolution, faults etc.	Large	Metrics gathering
Analysis of tool use logs	Details of tool usage	Large	
Documentation analysis	Design and documentation practices, general understanding	Medium	Reverse engineering
Static and dynamic analysis	Design and programming practices, general understanding	Large	Program comprehension, metrics, testing, etc.

1272 utilised documentation analysis to assess the overall ‘picture’ of how these services are documented. Further details on this experiment is given in **Chapter 4, Section 4.4.**

1275 3.4.2 Utility of API Documentation in Computer Vision Services

1276 To assess whether these services are sufficiently documented (i.e., RQ2), we conducted a systematic mapping study [191, 276] of the various academic sources 1277 detailing API documentation knowledge. We then consolidated this information 1278 into a structured taxonomy following a systematic taxonomy development method 1279 specific to software engineering studies [351].

1280 We then followed the triangulation approach proposed by Mayring [229] to validate the taxonomy by use of a personal opinion survey. Kitchenham and Pfleeger 1281 [192] provide an introduction on methods used to conduct personal opinion surveys 1282 which we adopted as an initial reference in (i) shaping our survey objectives around 1283 our research goals, (ii) designing a cross-sectional survey, (iii) developing and evaluating 1284 our survey instrument, (iv) evaluating our instruments, (v) obtaining the data 1285 and (vi) analysing the data. We adapted Brooke’s systematic usability scale [61] 1286 technique by basing our research questions against a known surveying instrument. 1287

1288 As is good practice in developing questionnaire instruments to evaluate their reliability and validity [217], we evaluated our instrument design by asking colleagues 1289 to critique it via pilot studies within A²I². This assisted in identifying any problems 1290 with the questionnaire itself and with any issues that may have occurred with the 1291 response rate and follow-up procedures.

1292 Findings from the pilot study helped inform us for a widely distributed questionnaire 1293 using snow-balling sampling. Ethics approval from the Faculty of Science, 1294 Engineering and Built Environment Human Ethics Advisory Group (SEBE HEAG) 1295 was approved to externally conduct this survey research (see Appendix E). Further 1296 details on these methods are detailed within **Chapter 8, Section 8.3.**

1299 3.4.3 Developer Issues concerning Computer Vision Services

1300 Developers typically congregate in search of discourses on issues they face in online 1301 forums, such as Stack Overflow (SO) and Quora, as well as writing their experiences 1302 in personal blogs such as Medium. The simplest of these platforms is SO (a sub- 1303 community of the Stack Exchange family of targeted communities) that specifically 1304 targets developer issues on using a simple Q&A interface, where developers can 1305 discuss technical aspects and general software development topics. Moreover, SO 1306 is often acknowledged as *the ‘go-to’ place* for developers to find high-quality code 1307 snippets that assist in their problems [334].

1308 Thus, to begin understanding the issues developers face when using CVSs and 1309 whether there is a substantial difference to conventional domains (i.e., RQ3), we 1310 used repository mining on SO to help answer RQ3. Specifically, we selected SO 1311 due to its targeted community of developers² and the availability of its publicly

2We also acknowledge that there are other targeted software engineering Stack Exchange

¹³¹² available dataset released as ‘data dumps’ on the Stack Exchange Data Explorer³
¹³¹³ and Google BigQuery⁴. Studies conducted have also used SO to mine developer
¹³¹⁴ discourse [8, 21, 28, 77, 215, 256, 266, 288, 300, 325, 340, 359]. Further details on
¹³¹⁵ how we approached the design for this study can be found in **Chapter 5, Section 5.4,**
¹³¹⁶ **Chapter 6, Section 6.3, and Chapter 7, Section 7.2.2**

¹³¹⁷ **3.4.4 Designing Improved Integration Strategies**

¹³¹⁸ Our improved integration strategies (i.e., RQ4) evolved organically over the duration
¹³¹⁹ of this research through the use of industry case studies and action research. We
¹³²⁰ developed several iterative prototypes to the integration strategies and used a mix
¹³²¹ of statistical and technical evaluations to analyse whether our improved integration
¹³²² strategies can prove useful. Further details about these approaches are detailed in
¹³²³ **Chapter 9, Section 9.5.1 and Chapter 10, Section 10.3 and Chapter 11, Sec-**
¹³²⁴ **tion 11.5.**

communities such as Stack Exchange Software Engineering (<https://softwareengineering.stackexchange.com>), though (as of January 2019) this much smaller community consists of only 52,000 questions versus SO’s 17 million.

³<https://data.stackexchange.com/stackoverflow> last accessed 17 January 2017.

⁴<https://console.cloud.google.com/marketplace/details/stack-exchange/stack-overflow> last accessed 17 January 2017.

1325

Part II

1326

Publications

CHAPTER 4

1327

1328

1329

Identifying Evolution in Computer Vision Services[†]

1330

1331 **Abstract** Recent advances in artificial intelligence (AI) and machine learning (ML), such
1332 as computer vision, are now available as intelligent web services (IWSs) and their acces-
1333 sibility and simplicity is compelling. Multiple vendors now offer this technology as cloud
1334 services and developers want to leverage these advances to provide value to end-users. How-
1335 ever, there is no firm investigation into the maintenance and evolution risks arising from use
1336 of these IWSs; in particular, their behavioural consistency and transparency of their function-
1337 ality. We evaluated the responses of three different IWSs (specifically computer vision) over
1338 11 months using 3 different data sets, verifying responses against the respective documenta-
1339 tion and assessing evolution risk. We found that there are: (1) inconsistencies in how these
1340 services behave; (2) evolution risk in the responses; and (3) a lack of clear communication
1341 that documents these risks and inconsistencies. We propose a set of recommendations to
1342 both developers and IWS providers to inform risk and assist maintainability.

4.1 Introduction

1343 The availability of intelligent web services (IWSs) has made artificial intelligence
1344 (AI) tooling accessible to software developers and promises a lower entry barrier for
1345 their utilisation. Consider state-of-the-art computer vision analysers, which require
1346 either manually training a deep-learning classifier, or selecting a pre-trained model
1347 and deploying these into an appropriate infrastructure. Either are laborious in time,
1348 and require non-trivial expertise along with a large data set when training or customisa-
1349 tion is needed. In contrast, IWSs providing computer vision (i.e., computer vision
1350 services or CVSs such as [386, 398, 399, 400, 407, 411, 419, 420, 421, 425, 438,

[†]This chapter is originally based on A. Cummaudo, R. Vasa, J. Grundy, M. Abdelrazek, and A. Cain, “Losing Confidence in Quality: Unspoken Evolution of Computer Vision Services,” in *Proceedings of the 35th IEEE International Conference on Software Maintenance and Evolution*. Cleveland, OH, USA: IEEE, December 2019. DOI 10.1109/ICSME.2019.00051. ISBN 978-1-72-813094-1 pp. 333–342. Terminology has been updated to fit this thesis.

439, 472, 473]) abstract these complexities behind a web application programming
1352 interface (API) call. This removes the need to understand the complexities required
1353 of machine learning (ML), and requires little more than the knowledge on how to
1354 use RESTful endpoints. The ubiquity of these services is exemplified through their
1355 rapid uptake in applications such as aiding the vision-impaired [94, 289].
1356

1357 While IWSs have seen quick adoption in industry, there has been little work
1358 that has considered the software quality perspective of the risks and impacts posed
1359 by using such services. In relation to this, there are three main challenges: (1)
1360 incorporating stochastic algorithms into software that has traditionally been deter-
1361 ministic; (2) the general lack of transparency associated with the ML models; and
1362 (3) communicating to application developers.

1363 ML typically involves use of statistical techniques that yield components with
1364 a non-deterministic external behaviour; that is, for the same given input, different
1365 outcomes may result. However, developers, in general, are used to libraries and small
1366 components behaving predictably, while systems that rely on ML techniques work
1367 on confidence intervals¹ and probabilities. For example, the developer’s mindset
1368 suggests that an image of a border collie—if sent to three intelligent computer vision
1369 services (CVSs)—would return the label ‘dog’ consistently with time regardless
1370 of which service is used. However, one service may yield the specific dog breed,
1371 ‘border collie’, another service may yield a permutation of that breed, ‘collie’, and
1372 another may yield broader results, such as ‘animal’; each with results of varying
1373 confidence values.² Furthermore, the third service may evolve with time, and
1374 thus learn that the ‘animal’ is actually a ‘dog’ or even a ‘collie’. The outcomes
1375 are thus behaviourally inconsistent between services providing conceptually similar
1376 functionality. As a thought exercise, consider if the sub-string function were created
1377 using ML techniques—it would perform its operation with a confidence where the
1378 expected outcome and the AI inferred output match as a *probability*, rather than a
1379 deterministic (constant) outcome. How would this affect the developers’ approach
1380 to using such a function? Would they actively take into consideration the non-
1381 deterministic nature of the result?

1382 Myriad software quality models and software engineering practices advocate
1383 maintainability and reliability as primary characteristics; stability, testability, fault
1384 tolerance, changeability and maturity are all concerns for quality in software com-
1385 ponents [160, 282, 327] and one must factor these in with consideration to soft-
1386 ware evolution challenges [139, 140, 238, 239, 345]. However, the effect this
1387 non-deterministic behaviour has on quality when masked behind an IWS is still
1388 under-explored to date in software engineering literature, to our knowledge. Where
1389 software depends on IWSs to achieve functionality, these quality characteristics may
1390 not be achieved, and developers need to be wary of the unintended side effects and
1391 inconsistency that exists when using non-deterministic components. A CVS may
1392 encapsulate deep-learning strategies or stochastic methods to perform image analy-

¹Varied terminology used here. Probability, confidence, accuracy and score may all be used interchangeably.

²Indeed, we have observed this phenomenon using a picture of a border collie sent to various CVSs.

1393 sis, but developers are more likely to approach IWSs with a mindset that anticipates
1394 consistency. Although the documentation does hint at this non-deterministic be-
1395 haviour (i.e., the descriptions of ‘confidence’ in various CVSs suggest the they are
1396 not always confident, and thus not deterministic [384, 409, 426]), the integration
1397 mechanisms offered by popular vendors do not seem to fully expose the nuances,
1398 and developers are not yet familiar with the trade-offs.

1399 Do popular CVSs, as they currently stand, offer consistent behaviour, and if not,
1400 how is this conveyed to developers (if it is at all)? If CVSs are to be used in production
1401 services, do they ensure quality under rigorous service quality assurance (SQA)
1402 frameworks [160]? What evolution risk [139, 140, 238, 239] do they pose if these
1403 services change? To our knowledge, few studies have been conducted to investigate
1404 these claims. This paper assesses the consistency, evolution risk and consequent
1405 maintenance issues that may arise when developers use IWSs. We introduce a
1406 motivating example in Section 4.2, discussing related work and our methodology
1407 in Sections 4.3 and 4.4. We present and interpret our findings in Section 4.5. We
1408 argue with quantified evidence that these IWSs can only be considered with a mature
1409 appreciation of risks, and we make a set of recommendations in Section 4.6.

1410 4.2 Motivating Example

1411 Consider Rosa, a software developer, who wants to develop a social media photo-
1412 sharing mobile app that analyses her and her friends photos on Android and iOS.
1413 Rosa wants the app to categorise photos into scenes (e.g., day vs. night, outdoors
1414 vs. indoors), generate brief descriptions of each photo, and catalogue photos of her
1415 friends as well as common objects (e.g., all photos with a dog, all photos on the
1416 beach).

1417 Rather than building a computer vision engine from scratch, Rosa thinks she
1418 can achieve this using one of the popular CVSs (e.g., [386, 398, 399, 400, 407, 411,
1419 419, 420, 421, 425, 438, 439, 472, 473]). However, Rosa comes from a typical
1420 software engineering background with limited knowledge of the underlying deep-
1421 learning techniques and implementations as currently used in computer vision. Not
1422 unexpectedly, she internalises a mindset of how such services work and behave based
1423 on her experience of using software libraries offered by various SDKs. This mindset
1424 assumes that different cloud vendor image processing APIs more-or-less provide
1425 similar functionality, with only minor variations. For example, cloud object storage
1426 for Amazon S3 is both conceptually and behaviourally very similar to that of Google
1427 Cloud Storage or Azure Storage. Rosa assumes the CVSs of these platforms will,
1428 therefore, likely be very similar. Similarly, consider the string libraries Rosa will
1429 use for the app. The conceptual and behavioural similarities are consistent; a string
1430 library in Java (Android) is conceptually very similar to the string library she will
1431 use in Swift (iOS), and likewise both behave similarly by providing the same results
1432 for their respective sub-string functionality. However, **unlike the cloud storage and**
1433 **string libraries, different CVSs often present conceptually similar functionality**
1434 **but are behaviourally very different.** IWS vendors also hide the depth of knowledge
1435 needed to use these effectively—for instance, the training data set and ontologies

¹⁴³⁶ used to create these services are hidden in the documentation. Thus, Rosa isn't even
¹⁴³⁷ exposed to this knowledge as she reads through the documentation of the providers
¹⁴³⁸ and, thus, Rosa makes the following assumptions:

- ¹⁴³⁹ • **"I think the responses will be consistent amongst these CVSSs."** When Rosa
¹⁴⁴⁰ uploads a photo of a dog, she would expect them all to respond with 'dog'. If
¹⁴⁴¹ Rosa decides to switch which service she is using, she expects the ontologies
¹⁴⁴² to be compatible (all CVSSs *surely* return dog for the same image) and therefore
¹⁴⁴³ she can expect to plug-in a different service should she feel like it making only
¹⁴⁴⁴ minor code modifications such as which endpoints she is relying on.
- ¹⁴⁴⁵ • **"I think the responses will be constant with time."** When Rosa uploads the
¹⁴⁴⁶ photo of a dog for testing, she expects the response to be the same in 10 weeks
¹⁴⁴⁷ time once her app is in production. Hence, in 10 weeks, the same photo of the
¹⁴⁴⁸ dog should return the same label.

¹⁴⁴⁹ 4.3 Related Work

¹⁴⁵⁰ If we were to view CVSSs through the lenses of an SQA framework, robustness,
¹⁴⁵¹ consistency, and maintainability often feature as quality attributes in myriad soft-
¹⁴⁵² ware quality models (e.g., [171]). Software quality is determined from two key
¹⁴⁵³ dimensions: (1) in the evaluation of the end-product (external quality) and (2) the
¹⁴⁵⁴ assurances in the development processes (internal quality) [282]. We discuss both
¹⁴⁵⁵ perspectives of quality within the context of our work in this section.

¹⁴⁵⁶ 4.3.1 External Quality

¹⁴⁵⁷ 4.3.1.1 Robustness for safety-critical applications

¹⁴⁵⁸ A typical focus of recent work has been to investigate the robustness of deep-
¹⁴⁵⁹ learning within computer vision technique implementation, thereby informing the
¹⁴⁶⁰ effectiveness in the context of the end-product. The common method for this has
¹⁴⁶¹ been via the use of adversarial examples [337], where input images are slightly
¹⁴⁶² perturbed to maximise prediction error but are still interpretable to humans.

¹⁴⁶³ Google Cloud Vision, for instance, fails to correctly classify adversarial examples
¹⁴⁶⁴ when noise is added to the original images [161]. Rosenfeld et al. [302] illustrated
¹⁴⁶⁵ that inserting synthetic foreign objects to input images (e.g., a cartoon elephant)
¹⁴⁶⁶ can completely alter classification output. Wang et al. [357] performed similar
¹⁴⁶⁷ attacks on a transfer-learning approach of facial recognition by modifying pixels of
¹⁴⁶⁸ a celebrity's face to be recognised as a completely different celebrity, all while still
¹⁴⁶⁹ retaining the same human-interpretable original celebrity. Su et al. [332] used the
¹⁴⁷⁰ ImageNet database to show that 41.22% of images drop in confidence when just a
¹⁴⁷¹ *single pixel* is changed in the input image; and similarly, Eykholt et al. [113] recently
¹⁴⁷² showed similar results that made a convolutional neural network (CNN) interpret a
¹⁴⁷³ stop road-sign (with mimicked graffiti) as a 45mph speed limit sign.

¹⁴⁷⁴ The results suggest that current state-of-the-art computer vision techniques may
¹⁴⁷⁵ not be robust enough for safety critical applications as they do not handle intentional

1476 or unintentional adversarial attacks. Moreover, as such adversarial examples exist in
1477 the physical world [113, 203], “the natural world may be adversarial enough” [277]
1478 to fool AI software. Though some limitations and guidelines have been explored
1479 in this area, the perspective of *Intelligent Web Services* is yet to be considered and
1480 specific guidelines do not yet exist when using CVSS.

1481 **4.3.1.2 Testing strategies in ML applications**

1482 Although much work applies ML techniques to automate testing strategies, there is
1483 only a growing emphasis that considers this in the opposite sense; that is, testing
1484 to ensure the ML product works correctly. There are few reliable test oracles
1485 that ensure if an ML has been implemented to serve its algorithm and use case
1486 purposefully; indeed, “the non-deterministic nature of many training algorithms
1487 makes testing of models even more challenging” [15]. Murphy et al. [247] proposed
1488 a software engineering-based testing approach on ML ranking algorithms to evaluate
1489 the ‘correctness’ of the implementation on a real-world data set and problem domain,
1490 whereby discrepancies were found from the formal mathematical proofs of the ML
1491 algorithm and the implementation.

1492 Recently, Braiek and Khomh [54] conducted a comprehensive review of testing
1493 strategies in ML software, proposing several research directions and recommendations
1494 in how best to apply software engineering testing practices in ML programs.
1495 However, much of the area of this work specifically targets ML engineers, and not
1496 application developers. Little has been investigated on how application developers
1497 perceive and understand ML concepts, given a lack of formal training; we note that
1498 other testing strategies and frameworks proposed (e.g., [58, 246, 255]) are targeted
1499 chiefly to the ML engineer, and not the application developer.

1500 However, Arpteg et al. [15] recently demonstrated (using real-world ML projects)
1501 the developmental challenges posed to developers, particularly those that arise when
1502 there is a lack of transparency on the models used and how to troubleshoot ML
1503 frameworks using traditional software engineering debugging tools. This said, there
1504 is no further investigations into challenges when using the higher, ‘ML friendly’
1505 layers (e.g., IWSs) of the ‘machine learning spectrum’ [263], rather than the ‘lower
1506 layers’ consisting of existing ML frameworks and algorithms targeted toward the
1507 ML community.

1508 **4.3.2 Internal Quality**

1509 **4.3.2.1 Quality metrics for cloud services**

1510 CVSS are based on cloud computing fundamentals under a subset of the Platform as
1511 a Service (PaaS) model. There has been work in the evaluation of PaaS in terms of
1512 quality attributes [128]: these attributes are exposed using service-level agreements
1513 (SLAs) between vendors and customers, and customers denote their demanded
1514 quality of service (QoS) to ensure the cloud services adhere to measurable KPI
1515 attributes.

1516 Although, popular services, such as cloud object storage, come with strong QoS

1517 agreement, to date IWSs do not come with deep assurances around their performance
1518 and responses, but do offer uptime guarantees. For example, how can Rosa demand
1519 a QoS that ensures all photos of dogs uploaded to her app guarantee the specific dog
1520 breeds are returned so that users can look up their other friend's 'border collie's?
1521 If dog breeds are returned, what ontologies exist for breeds? Are they consistent
1522 with each other, or shortened? ('Collie' versus 'border collie'; 'staffy' versus
1523 'staffordshire bull terrier')? For some applications, these unstated QoS metrics
1524 specific to the ML service may have significant legal ramifications.

1525 4.3.2.2 *Web service documentation and documenting ML*

1526 From the *developer's* perspective, little has been achieved to assess IWS quality
1527 or assure quality of these CVSs. Web services and their APIs are the bridge be-
1528 tween developers' needs and the software components [13]; therefore, assessing
1529 such CVSs from the quality of their APIs is thereby directly related to the develop-
1530 ment quality [196]. Good APIs should be intuitive and require less documentation
1531 browsing [279], thereby increasing productivity. Conversely, poor APIs that are
1532 hard to understand and work with reduce developer productivity, thereby reducing
1533 product quality. This typically leads to developers congregating on forums such as
1534 Stack Overflow, leading to a repository of unstructured knowledge likely to concern
1535 API design [361]. The consequences of addressing these concerns in development
1536 leads to a higher demand in technical support (as measured in [157]) that, ultimately,
1537 causes the maintenance to be far more expensive, a phenomenon widely known in
1538 software engineering economics [47]. Rosa, for instance, isn't aware of technical ML
1539 concepts; if she cannot reason about what search results are relevant when brows-
1540 ing the service and understanding functionality, her productivity is significantly
1541 decreased. Conceptual understanding is critical for using APIs, as demonstrated by
1542 Ko and Riche, and the effects of maintenance this may have in the future of her
1543 application is unknown.

1544 Recent attempts to document attributes and characteristics on ML models have
1545 been proposed. Model cards were introduced by Mitchell et al. [243] to describe how
1546 particular models were trained and benchmarked, thereby assisting users to reason
1547 if the model is right for their purposes and if it can achieve its stated outcomes.
1548 Gebru et al. [132] also proposed datasheets, a standardised documentation format to
1549 describe the need for a particular data set, the information contained within it and
1550 what scenarios it should be used for, including legal or ethical concerns.

1551 However, while target audiences for these documents may be of a more technical
1552 AI level (i.e., the ML engineer), there is still no standardised communication format
1553 for application developers to reason about using particular IWSs, and the ramifica-
1554 tions this may have on the applications they write is not fully conveyed. Hence, our
1555 work is focused on the application developer perspective.

4.4 Method

1557 This study organically evolved by observing phenomena surrounding CVSs by as-
1558 sessing both their documentation and responses. We adopted a mixed methods
1559 approach, performing both qualitative and quantitative data collection on these two
1560 key aspects by using documentary research methods for inspecting the documen-
1561 tation and structured observations to quantitatively analyse the results over time.
1562 This, ultimately, helped us shape the following research hypotheses which this paper
1563 addresses:

1564 [RH1] CVSs do not respond with consistent outputs between services, given the
1565 same input image.

1566 [RH2] The responses from CVSs are non-deterministic and evolving, and the same
1567 service can change its top-most response over time given the same input
1568 image.

1569 [RH3] CVSs do not effectively communicate this evolution and instability, intro-
1570 ducing risk into engineering these systems.

1571 We conducted two experiments to address these hypotheses against three popular
1572 CVSs: AWS Rekognition [386], Google Cloud Vision [411], Azure Computer
1573 Vision [425]. Specifically, we targeted the AWS DetectLabels endpoint [384],
1574 the Google Cloud Vision annotate:images endpoint [409] and Azure's analyze
1575 endpoint [426]. For the remainder of this paper, we de-identify our selected CVSs
1576 by labelling them as services A, B and C but do not reveal mapping to prevent
1577 any implicit bias. Our selection criteria for using these particular three services
1578 are based on the weight behind each service provider given their prominence in
1579 the industry (Amazon, Google and Microsoft), the ubiquity of their hosting cloud
1580 platforms as industry leaders of cloud computing (i.e., AWS, Google Cloud and
1581 Azure), being in the top three most adopted cloud vendors in enterprise applications
1582 in 2018 [294] and the consistent popularity of discussion amongst developers in
1583 developer communities such as Stack Overflow. While we choose these particular
1584 cloud CVSs, we acknowledge that similar services [399, 400, 407, 420, 421, 472, 473]
1585 also exist, including other popular services used in Asia [398, 419, 438, 439] (some
1586 offering 3D image analysis [397]). We reflect on the impacts this has to our study
1587 design in Section 4.7.

1588 Our study involved an 11-month longitudinal study which consisted of two 13
1589 week and 17 week experiments from April to August 2018 and November 2018 to
1590 March 2019, respectively. Our investigation into documentation occurred on August
1591 28 2018. In total, we assessed the services with three data sets; we first ran a pilot
1592 study using a smaller pool of 30 images to confirm the end-points remain stable,
1593 re-running the study with a larger pool of images of 1,650 and 5,000 images. Our
1594 selection criteria for these three data sets were that the images had to have varying
1595 objects, taken in various scenes and various times. Images also needed to contain
1596 disparate objects. Our small data set was sourced by the first author by taking photos
1597 of random scenes in an afternoon, whilst our second data set was sourced from
1598 various members of our research group from their personal photo libraries. We also

Table 4.1: Characteristics of our datasets and responses.

Data set	Small	Large	COCOVal17
# Images/data set	30	1,650	5000
# Unique labels found	307	3506	4507
Number of snapshots	9	22	22
Avg. days b/n requests	12 Days	8 Days	8 Days

wanted to include a data set that was publicly available prior to running our study, so for this data set we chose the COCO 2017 validation data set [214]. We have made our other two data sets available online ([402]). We collected results and their responses from each service’s API endpoint using a python script [406] that sent requests to each service periodically via cron jobs. Table 4.1 summarises various characteristics about the data sets used in these experiments.

We then performed quantitative analyses on each response’s labels, ensuring all labels were lowercased as case changed for services A and C over the evaluation period. To derive at the consistency of responses for each image, we considered only the ‘top’ labels per image for each service and data set. That is, for the same image i over all images in data set D where $i \in D$ and over the three services, the top labels per image (T_i) of all labels per image L_i (i.e., $T_i \subseteq L_i$) is that where the respective label’s confidences are consistently the highest of all labels returned. Typically, the top labels returned is a set containing only one element—that is, only one unique label consistently returned with the highest label ($|T_i| = 1$)—however there are cases where the top labels contains multiple elements as their respective confidences are *equal* ($|T_i| > 1$).

We measure response consistency under 6 aspects:

- (1) **Consistency of the top label between each service.** Where the same image of, for example, a dog is sent to the three services, the top label for service A may be ‘animal’, B ‘canine’ and C ‘animal’. Therefore, service B is inconsistent.
- (2) **Semantic consistency of the top labels.** Where a service has returned multiple top labels ($|T_i| > 1$), there may lie semantic differences in what the service thinks the image best represents. Therefore, there is conceptual inconsistency in the top labels for a service even when the confidences are equal.
- (3) **Consistency of the top label’s confidence per service.** The top label for an image does not guarantee a high confidence. Therefore, there may be inconsistencies in how confident the top labels for all images in a service is.
- (4) **Consistency of confidence in the intersecting top label between each service.** The spread of a top intersecting label, e.g., ‘cat’, may not have the same confidences per service even when all three services agree that ‘cat’ is the top label. Therefore, there is inconsistency in the confidences of a top label even where all three services agree.
- (5) **Consistency of the top label over time.** Given an image, the top label in one week may differ from the top label the following week. Therefore, there is inconsistency in the top label itself due to model evolution.



Figure 4.1: The only consistent label for the above image is ‘people’ for services C and B. The top label for A is ‘conversation’ and this label is not registered amongst the other two services.

Table 4.2: Ratio of the top labels (to images) that intersect in each data set for each permutation of service.

Service	Small	Large	COCOVal17	μ	σ
$A \cap B \cap C$	3.33%	2.73%	4.68%	2.75%	0.0100
$A \cap B$	6.67%	11.27%	12.26%	10.07%	0.0299
$A \cap C$	20.00%	13.94%	17.28%	17.07%	0.0304
$B \cap C$	6.67%	12.97%	20.90%	13.51%	0.0713

1635 (6) **Consistency of the top label’s confidence over time.** The top label of an
 1636 image may remain static from one week to the next for the same service, but
 1637 its confidence values may change with time. Therefore, there is inconsistency
 1638 in the top label’s confidence due to model evolution.

1639 For the above aspects of consistency, we calculated the spread of variation for the
 1640 top label’s confidences of each service for every 1 percent point; that is, the frequency
 1641 of top label confidences within 100–99%, 99–98% etc. The consistency of top label’s
 1642 and their confidences between each service was determined by intersecting the labels
 1643 of each service per image and grouping the intersecting label’s confidences together.
 1644 This allowed us to determine relevant probability distributions. For reproducibility,
 1645 all quantitative analysis is available online [403].

1646 4.5 Findings

1647 4.5.1 Consistency of top labels

1648 4.5.1.1 Consistency across services

1649 Table 4.2 presents the consistency of the top labels between data sets, as measured
 1650 by the cardinality of the intersection of all three services’ set of top labels divided
 1651 by the number of images per data set. A combination of services present varied
 1652 overlaps in their top labels; services A and C provide the best overlap for all three
 1653 data sets, however the intersection of all three irrespective of data sets is low.

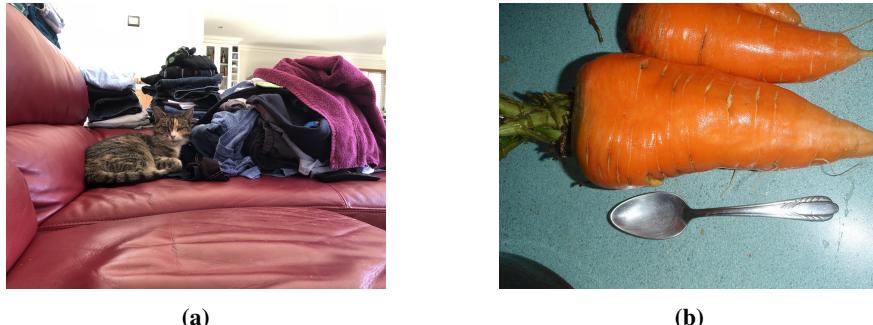


Figure 4.2: *Left:* The top labels for each service do not intersect, with each having a varied ontology: $T_i = \{ A = \{ \text{'black'} \}, B = \{ \text{'indoor'} \}, C = \{ \text{'slide'}, \text{'toy'} \} \}$. (Service C returns both ‘slide’ and ‘toy’ with equal confidence.) *Right:* The top labels for each service focus on disparate subjects in the image: $T_i = \{ A = \{ \text{'carrot'} \}, B = \{ \text{'indoor'} \}, C = \{ \text{'spoon'} \} \}$.

1654 The implication here is that, without semantic comparison (see Section 4.7),
 1655 service vendors are not ‘plug-and-play’. If Rosa uploaded the sample images in
 1656 this paper to her application to all services, she would find that only Figure 4.1
 1657 responds with ‘person’ for services B and C in their respective set of top labels.
 1658 However, if she decides to then adopt service A, then Figure 4.1’s top label becomes
 1659 ‘conversation’; the ‘person’ label does not appear within the top 15 labels for service
 1660 A and, conversely, the ‘conversation’ label does not appear in the other services top
 1661 15.

1662 Should she decide if the performance of a particular service isn’t to her needs,
 1663 then the vocabulary used for these labels becomes inconsistent for all other images;
 1664 that is, the top label sets per service for Figure 4.2a shows no intersection at all.
 1665 Furthermore, the part of the image each service focuses on may not be consistent
 1666 for their top labels; in Figure 4.2b, service A’s top label focuses on the vegetable
 1667 (‘carrot’), service C focuses on the ‘spoon’, while service B’s focus is that the image
 1668 is ‘indoor’s. It is interesting to note that service B focuses on the scene matter
 1669 (indoors) rather than the subject matter. (Furthermore, we do not actually know if
 1670 the image in Figure 4.2b was taken indoors.)

1671 Hence, developers should ensure that the vocabulary used by a particular service
 1672 is right for them before implementation. As each service does not work to the
 1673 same standardised model, trained with disparate training data, and tuned differently,
 1674 results will differ despite the same input. This is unlike deterministic systems: for
 1675 example, switching from AWS Object Storage to Google Cloud Object storage will
 1676 conceptually provide the same output (storing files) for the same input (uploading
 1677 files). However, CVSs do not agree on the top label for images, and therefore
 1678 developers are likely to be vendor locked, making changes between services non-
 1679 trivial.

1680 4.5.1.2 Semantic consistency where $|T_i| > 1$

1681 Service C returns two top labels for Figure 4.2a; ‘slide’ and ‘toy’. More than one
 1682 top label is typically returned in service C (80.00%, 56.97%, and 81.66% of all



Figure 4.3: *Left:* Service C is 98.49% confident of the following labels: { ‘beverage’, ‘chocolate’, ‘cup’, ‘dessert’, ‘drink’, ‘food’, ‘hot chocolate’ }. However, it is up to the developer to decide which label to persist with as all are returned. *Right:* Service B persistently returns a top label set of { ‘book’, ‘several’ }. Both are semantically correct for the image, but disparate in what the label is to describe.

1683 images for all three data sets, respectively) though this also occurs in B in the large
 1684 (4.97% of all images) and COCOVal17 data sets (2.38%). Semantic inconsistencies
 1685 of what this label conceptually represents becomes a concern as these labels have
 1686 confidences of *equal highest* consistency. Thus, some services are inconsistent in
 1687 themselves and cannot give a guaranteed answer of what exists in an image; services
 1688 C and B have multiple top labels, but the respective services cannot ‘agree’ on
 1689 what the top label actually is. In Figure 4.3a, service C presents a reasonably high
 1690 confidence for the set of 7 top labels it returns, however there is too much diversity
 1691 ranging from a ‘hot chocolate’ to the hypernym ‘food’. Both are technically correct,
 1692 but it is up to the developer to decide the level of hypernymy to label the image as.
 1693 We also observe a similar effect in Figure 4.3b, where the image is labelled with
 1694 both the subject matter and the number of subjects per image.

1695 Thus, a taxonomy of ontologies is unknown; if a ‘border collie’ is detected in
 1696 an image, does this imply the hypernym ‘dog’ is detected, and then ‘mammal’, then
 1697 ‘animal’, then ‘object’? Only service B documents a taxonomy for capturing what
 1698 level of scope is desired, providing what it calls the ‘86-category’ concept as found
 1699 in its how-to guide:

1700 “*Identify and categorize an entire image, using a category taxonomy*
 1701 *with parent/child hereditary hierarchies. Categories can be used alone,*
 1702 *or with our new tagging models.”* [427]

1703 Thus, even if Rosa implemented conceptual similarity analysis for the image, the
 1704 top label set may not provide sufficient information to derive at a conclusive answer,
 1705 and if simply relying on only one label in this set, information such as the duplicity
 1706 of objects (e.g., ‘several’ in Figure 4.3b) may be missed.

Table 4.3: Ratio of the top labels (to images) that remained the top label but changed confidence values between intervals.

Service	Small	Large	COCOVal17	$\mu(\delta_c)$	$\sigma(\delta_c)$	Median(δ_c)	Range(δ_c)
A	53.33%	59.19%	44.92%	9.62e-8	6.84e-8	5.96e-8	[5.96e-8, 6.56e-7]
B	0.00%	0.00%	0.02%	-	-	-	-
C	33.33%	41.36%	15.60%	5.35e-7	8.76e-7	3.05e-7	[1.27e-7, 1.13e-5]

4.5.2 Consistency of confidence

4.5.2.1 Consistency of top label's confidence

In Figure 4.4, we see that there is high probability that top labels have high confidences for all services. In summary, one in nine images uploaded to any service will return a top label confident to at least 97%. However, there is higher probability for service A returning a lower confidence, followed by B. The best performing service is C, with 90% of requests having a top label confident to $\gtrsim 95\%$, when compared to $\gtrsim 87\%$ and $\gtrsim 93\%$ for services A and B, respectively.

Therefore, Rosa could generally expect that the top labels she receives in her images do have high confidence. That is, each service will return a top label that they are confident about. This result is expected, considering that the ‘top’ label is measured by the highest confidence, though it is interesting to note that some services are generally more confident than others in what they present back to users.

4.5.2.2 Consistency of intersecting top label's confidence

Even where all three services do agree on a set of top labels, the disparity of how much they agree by is still of importance. Just because three services agree that an image contains consistent top labels, they do not always have a small spread of confidence. In Figure 4.6, the three services agree with $\sigma = 0.277$, significantly larger than that of all images in general $\sigma = 0.0831$. Figure 4.5 displays the cumulative distribution of all intersecting top labels’ confidence values, presenting slightly similar results to that of Figure 4.4.

4.5.3 Evolution risk

4.5.3.1 Label Stability

Generally, the top label(s) did not evolve in the evaluation period. 16.19% and 5.85% of images did change their top label(s) in the Large and COCOVal17 data sets in service A. Thus, top labels are stable but not guaranteed to be constant.

4.5.3.2 Confidence Stability

Similarly, where the top label(s) remained the same from one interval to the next, the confidence values were stable. Table 4.3 displays the proportion of images that changed their top label’s confidence values with various statistics on the confidence

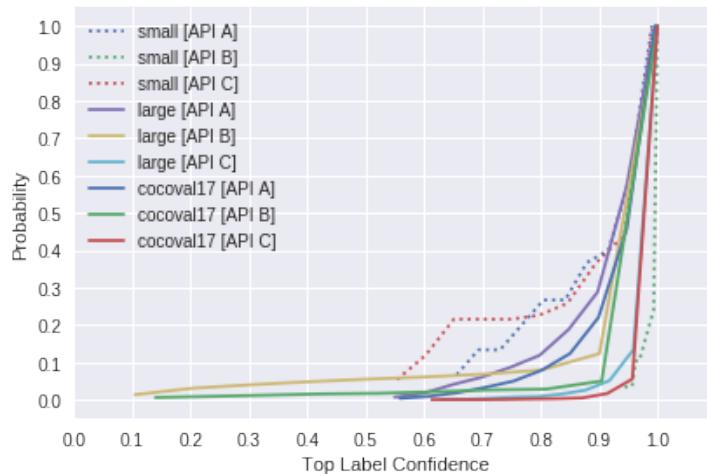


Figure 4.4: Cumulative distribution of the top labels' confidences. One in nine images return a top label(s) confident to $\gtrsim 97\%$, though there is a wider distribution for service A.

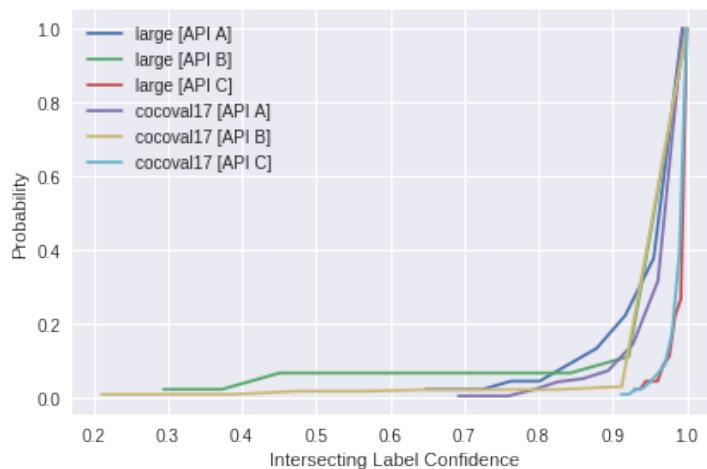


Figure 4.5: Cumulative distribution of intersecting top labels' confidences. The small data set is intentionally removed due to low intersections of labels (see Table 4.2).



Figure 4.6: All three services agree the top label for the above image is ‘food’, but the confidences to which they agree by vary significantly. Service C is most confident to 94.93% (in addition with the label ‘bread’); service A is the second most confident to 84.32%; service B is the least confident with 41.39%.

¹⁷³⁷ deltas between snapshots (δ_c). However, this delta is so minuscule that we attribute
¹⁷³⁸ such changes to statistical noise.

¹⁷³⁹ 4.6 Recommendations

¹⁷⁴⁰ 4.6.1 Recommendations for IWS users

¹⁷⁴¹ 4.6.1.1 *Test with a representative ontology for the particular use case*

¹⁷⁴² Rosa should ensure that in her testing strategies for the app she develops, there is an
¹⁷⁴³ ontology focus for the types of vocabulary that are returned. Additionally, we noted
¹⁷⁴⁴ that there was a sudden change in case for services A and C; for all comparative
¹⁷⁴⁵ purposes of labels, each label should be lower-cased.

¹⁷⁴⁶ 4.6.1.2 *Incorporate a specialised IWS testing methodology into the development 1747 lifecycle*

¹⁷⁴⁸ Rosa can utilise the different aspects of consistency as outlined in this paper as
¹⁷⁴⁹ part of her quality strategy. To ensure results are correct over time, we recommend
¹⁷⁵⁰ developers create a representative data set of the intended application’s data set
¹⁷⁵¹ and evaluate these changes against their chosen service frequently. This will help
¹⁷⁵² identify when changes, if any, have occurred if vendors do not provide a line of
¹⁷⁵³ communication when this occurs.

¹⁷⁵⁴ 4.6.1.3 *IWSs are not ‘plug-and-play’*

¹⁷⁵⁵ Rosa will be locked into whichever vendor she chooses as there is inherent incon-
¹⁷⁵⁶ sistency between these services in both the vocabulary and ontologies that they use.
¹⁷⁵⁷ We have demonstrated that very few services overlap in their vocabularies, chiefly
¹⁷⁵⁸ because they are still in early development and there is yet to be an established,
¹⁷⁵⁹ standardised vocabulary that can be shared amongst the different vendors. Issues
¹⁷⁶⁰ such as those shown in Section 4.5.1 can therefore be avoided.

1761 Throughout this work, we observed that the terminologies used by the vari-
1762 ous vendors are different. Documentation was studied, and we note that there is
1763 inconsistency between the ways techniques are described to users. We note the
1764 disparity between the terms ‘detection’, ‘recognition’, ‘localisation’ and ‘analysis’.
1765 This applies chiefly to object- and facial-related techniques. Detection applies to
1766 facial detection, which gives bounding box coordinates around all faces in an image.
1767 Similarly, localisation applies the same methodology to disparate objects in an image
1768 and labels them. In the context of facial ‘recognition’, this term implies that a face
1769 is *recognised* against a known set of faces. Lastly, ‘analysis’ applies in the context
1770 of facial analysis (gender, eye colour, expression etc.); there does not exist a similar
1771 analysis technique on objects.

1772 We notice similar patterns with object ‘tagging’, ‘detection’ and ‘labelling’.
1773 Service A uses ‘Entity Detection’ for object categorisation, service B uses ‘Image
1774 Tagging’, and service C uses the term ‘Detect Labels’ : conceptually, these provide
1775 the same functionality but the lack of consistency used between all three providers is
1776 concerning and leaves room for confusion with developers during any comparative
1777 analyses. Rosa may find that she wants to label her images into day/night scenes, but
1778 this in turn means the ‘labelling’ of varying objects. There is therefore no consistent
1779 standards to use the same terminology for the same concepts, as there are in other
1780 developer areas (such as Web Development).

1781 4.6.1.4 *Avoid use in safety-critical systems*

1782 We have demonstrated in this paper that both labels and confidences are stable but not
1783 constant; there is still an evolution risk posed to developers that may cause unknown
1784 consequences in applications dependent on these CVSs. Developers should avoid
1785 their use in safety critical systems due to the lack of visible changes.

1786 4.6.2 **Recommendations for IWS providers**

1787 4.6.2.1 *Improve the documentation*

1788 Rosa does not know that service A returns back ‘carrot’ for its top response, with
1789 service C returning ‘spoon’ (Figure 4.2b). She is unable to tell the service’s API
1790 where to focus on the image. Moreover, how can she toggle the level of specificity
1791 in her results? She is frustrated that service C can detect ‘chocolate’, ‘food’ and also
1792 ‘beverage’ all as the same top label in Figure 4.3a: what label is she to choose when
1793 the service is meant to do so for her, and how does she get around this? Thus, we
1794 recommend vendors to improve the documentation of services by making known
1795 the boundary set of the training data used for the algorithms. By making such
1796 information publicly available, developers would be able to review the service’s
1797 specificity for their intended use case (e.g., maybe Rosa is satisfied her app can
1798 catalogue ‘food’ together, and in fact does not want specific types of foods (‘hot
1799 chocolate’) catalogued). We also recommend that vendors publish usage guidelines
1800 should that include details of priors and how to evaluate the specific service results.

1801 Furthermore, we did not observe that the vendors documented how some images

may respond with multiple labels of the exact same confidence value. It is not clear from the documentation that response objects can have duplicate top values, and tutorials and examples provided by the vendors do not consider this possibility. It is therefore left to the developer to decide which label from this top set of labels best suits for their particular use case; the documentation should describe that a rule engine may need to be added in the developer's application to verify responses. The implications this would have on maintenance would be significant.

1809 4.6.2.2 *Improve versioning*

1810 We recommend introducing a versioning system so that a model can be used from a
1811 specific date in production systems: when Rosa tests her app today, she would like
1812 the service to remain *static* the same for when her app is deployed in production
1813 tomorrow. Thus, in a request made to the vendor, Rosa could specify what date she
1814 ran her app's QA testing on so that she knows that henceforth these model changes
1815 will not affect her app.

1816 4.6.2.3 *Improve Metadata in Response*

1817 Much of the information in these services is reduced to a single confidence value
1818 within the response object, and the details about training data and the internal AI
1819 architecture remains unknown; little metadata is provided back to developers that
1820 encompass such detail. Early work into model cards and datasheets [132, 243]
1821 suggests more can be done to document attributes about ML systems, however at a
1822 minimum from our work, we recommend including a reference point via the form
1823 of an additional identifier. This identifier must also permit the developers to submit
1824 the identifier to another API endpoint should the developer wish to find further
1825 characteristics about the AI empowering the IWS, reinforcing the need for those
1826 presented in model cards and datasheets. For example, if Rosa sends this identifier
1827 she receives in the response object to the IWS descriptor API, she could find out
1828 additional information such as the version number or date when the model was
1829 trained, thereby resolving potential evolution risk, and/or the ontology of labels.

1830 4.6.2.4 *Apply constraints for predictions on all inputs*

1831 In this study, we used some images with intentionally disparate, and noisy objects. If
1832 services are not fully confident in the responses they give back, a form of customised
1833 error message should be returned. For example, if Rosa uploads an image of 10
1834 various objects on a table, rather than returning a list of top labels with varying
1835 confidences, it may be best to return a 'too many objects' exception. Similarly, if
1836 Rosa uploads a photo that the model has had no priors on, it might be useful to return
1837 an 'unknown object' exception than to return a label it has no confidence of. We do
1838 however acknowledge that current state of the art computer vision techniques may
1839 have limits in what they can and cannot detect, but this limitation can be exposed in
1840 the documentation to the developers.

1841 A further example is sending a one pixel image to the service, analogous to
1842 sending an empty file. When we uploaded a single pixel white image to service A,
1843 we received responses such as ‘microwave oven’, ‘text’, ‘sky’, ‘white’ and ‘black’
1844 with confidences ranging from 51–95%. Prior checks should be performed on all
1845 input data, returning an ‘insufficient information’ error where any input data is below
1846 the information of its training data.

1847 4.7 Threats to Validity

1848 4.7.1 Internal Validity

1849 Not all CVSs were assessed. As suggested in Section 4.4, we note that there are
1850 other CVSs such as IBM Watson. Many services from Asia were also not considered
1851 due to language barriers (of the authors) in assessing these services. We limited our
1852 study to the most popular three providers (outside of Asia) to maintain focus in this
1853 body of work.

1854 A custom confidence threshold was not set. All responses returned from each of
1855 the services were included for analysis; where confidences were low, they were still
1856 included for analysis. This is because we used the default thresholds of each API to
1857 hint at what real-world applications may be like when testing and evaluating these
1858 services.

1859 The label string returned from each service was only considered. It is common
1860 for some labels to respond back that are conceptually similar (e.g., ‘car’ vs. ‘automobile’)
1861 or grammatically different (e.g., ‘clothes’ vs. ‘clothing’). While we could have
1862 employed more conceptual comparison or grammatical fixes in this study, we chose
1863 only to compare lowercased labels and as returned. We leave semantic comparison
1864 open to future work.

1865 Only introductory analysis has been applied in assessing the documentation of
1866 these services. Further detailed analysis of documentation quality against a rigorous
1867 documentation quality framework would be needed to fortify our analysis of the
1868 evolution of these services’ documentation.

1869 4.7.2 External Validity

1870 The documentation and services do change over time and evolve, with many allowing
1871 for contributions from the developer community via GitHub. We note that our
1872 evaluation of the documentation was conducted on a single date (see Section 4.4)
1873 and acknowledge that the documentation may have changed from the evaluation date
1874 to the time of this publication. We also acknowledge that the responses and labelling
1875 may have evolved too since the evaluation period described and the date of this
1876 publication. Thus, this may have an impact on the results we have produced in this
1877 paper compared to current, real-world results. To mitigate this, we have supplied the
1878 raw responses available online [404].

1879 Moreover, in this paper we have investigated *computer vision* services. Thus,
1880 the significance of our results to other domains such as natural language processing

1881 or audio transcription is, therefore, unknown. Future studies may wish to repeat our
1882 methodology on other domains to validate if similar patterns occur; we remain this
1883 open for future work.

1884 4.7.3 Construct Validity

1885 It is not clear if all the recommendations proposed in Section 4.6 are feasible
1886 or implementable in practice. Construct validity defines how well an experiment
1887 measures up to its claims; the experiments proposed in this paper support our three
1888 hypotheses but these have been conducted in a clinical condition. Real-world case
1889 studies and feedback from developers and providers in industry would remove the
1890 controlled nature of our work.

1891 4.8 Conclusions & Future Work

1892 This study explored three popular CVSs over an 11 month longitudinal experiment
1893 to determine if these services pose any evolution risk or inconsistency. We find that
1894 these services are generally stable but behave inconsistently; responses from these
1895 services do change with time and this is not visible to the developers who use them.
1896 Furthermore, the limitations of these systems are not properly conveyed by vendors.
1897 From our analysis, we present a set of recommendations for both IWS vendors and
1898 developers.

1899 Standardised software quality models (e.g., [171]) target maintainability and
1900 reliability as primary characteristics. Quality software is stable, testable, fault
1901 tolerant, easy to change and mature. These CVSs are, however, in a nascent stage,
1902 difficult to evaluate, and currently are not easily interchangeable. Effectively, the
1903 IWS response objects are shifting in material ways to developers, albeit slowly, and
1904 vendors do not communicate this evolution or modify API endpoints; the endpoint
1905 remains static but the content returned does not despite the same input.

1906 There are many potential directions stemming from this work. To start, we plan
1907 to focus on preparing a more comprehensive datasheet specifically targeted at what
1908 should be documented to application developers, and not data scientists. Reapplying
1909 this work in real-world contexts, that is, to get real developer opinions and study
1910 production grade systems, would also be beneficial to understand these phenomena
1911 in-context. This will help us clarify if such changes are a real concern for developers
1912 (i.e., if they really need to change between services, or the service evolution has real
1913 impact on their applications). We also wish to refine and systematise the method
1914 used in this study and develop change detectors that can be used to identify evolution
1915 in these services that can be applied to specific ML domains (i.e., not just computer
1916 vision), data sets, and API endpoints, thereby assisting application developers in their
1917 testing strategies. Moreover, future studies may wish to expand the methodology
1918 applied by refining how the responses are compared. As there does not yet exist a
1919 standardised list of terms available between services, labels could be *semantically*
1920 compared instead of using exact matches (e.g., by using stem words and synonyms
1921 to compare similar meanings of these labels), similar to previous studies [260].

1922 This paper has highlighted only some high-level issues that may be involved
1923 in using these evolving services. The laws of software evolution suggest that for
1924 software to be useful, it must evolve [239, 345]. There is, therefore, a trade-off, as
1925 we have shown, between consistency and evolution in this space. For a component
1926 to be stable, any changes to dependencies it relies on must be communicated. We
1927 are yet to see this maturity of communication from IWS providers. Thus, developers
1928 must be cautious between integrating intelligent components into their applications
1929 at the expense of stability; as the field of AI is moving quickly, we are more likely to
1930 see further instability and evolution in IWSs as a consequence.

CHAPTER 5

1931

1932

1933

Interpreting Pain-Points in Computer Vision Services[†]

1934

1935 **Abstract** Intelligent web services (IWSs) are becoming increasingly more pervasive; application developers want to leverage the latest advances in areas such as computer vision to provide new services and products to users, and large technology firms enable this via RESTful APIs. While such APIs promise an easy-to-integrate on-demand machine intelligence, their current design, documentation and developer interface hides much of the underlying machine learning techniques that power them. Such APIs look and feel like conventional APIs but abstract away data-driven probabilistic behaviour—the implications of a developer treating these APIs in the same way as other, traditional cloud services, such as cloud storage, is of concern. The objective of this study is to determine the various pain-points developers face when implementing systems that rely on the most mature of these intelligent web services, specifically those that provide computer vision. We use Stack Overflow to mine indications of the frustrations that developers appear to face when using computer vision services, classifying their questions against two recent classification taxonomies (documentation-related and general questions). We find that, unlike mature fields like mobile development, there is a contrast in the types of questions asked by developers. These indicate a shallow understanding of the underlying technology that empower such systems. We discuss several implications of these findings via the lens of learning taxonomies to suggest how the software engineering community can improve these services and comment on the nature by which developers use them.

5.1 Introduction

1955 The availability of recent advances in artificial intelligence (AI) over simple RESTful
1956 end-points offers application developers new opportunities. These new intelligent

[†]This chapter is originally based on A. Cummaudo, R. Vasa, S. Barnett, J. Grundy, and M. Abdellrazek, “Interpreting Cloud Computer Vision Pain-Points: A Mining Study of Stack Overflow,” in *Proceedings of the 42nd International Conference on Software Engineering*. Seoul, Republic of Korea: ACM, July 2020, In Press. Terminology has been updated to fit this thesis.

1957 web services (IWSs) are AI components that abstract complex machine learning
1958 (ML) and AI techniques behind simpler API calls. In particular, they hide (either
1959 explicitly or implicitly) any data-driven and non-deterministic properties inherent
1960 to the process of their construction. The promise is that software engineers can
1961 incorporate complex machine learnt capabilities, such as computer vision, by simply
1962 calling an API end-point.

1963 The expectation is that application developers can use these AI-powered services
1964 like they use other conventional software components and cloud services (e.g., object
1965 storage like AWS S3). Furthermore, the documentation of these AI components is
1966 still anchored to the traditional approach of briefly explaining the end-points with
1967 some information about the expected inputs and responses. The presupposition
1968 is that developers can reason and work with this high level information. These
1969 services are also marketed to suggest that application developers do not need to fully
1970 understand how these components were created (i.e., assumptions in training data
1971 and training algorithms), the ways in which the components can fail, and when such
1972 components should and should not be used.

1973 The nuances of ML and AI powering IWSs have to be appreciated, as there are
1974 real-world consequences to software quality for applications that depend on them if
1975 they are ignored [87]. This is especially true when ML and AI are abstracted and
1976 masked behind a conventional-looking API call, yet the mechanisms behind the API
1977 are data-dependent, probabilistic and potentially non-deterministic [260]. We are
1978 yet to discover what long-term impacts exist during development and production due
1979 to poor documentation that do not capture these traits, nor do we know the depth of
1980 understanding application developers have for these components. Given the way AI-
1981 powered services are currently presented, developers are also likely to reason about
1982 these new services much like a string library or a cloud data storage service. That
1983 is, they may not fully consider the implications of the underlying statistical nature
1984 of these new abstractions or the consequent impacts on productivity and quality.

1985 Typically, when developers are unable to correctly align to the mindset of the
1986 API designer, they attempt to resolve issues by (re-)reading the API documentation.
1987 If they are still unable to resolve these issues on their own after some internet
1988 searching, they consider online discussion platforms (e.g., Stack Overflow, GitHub
1989 Issues, Mailing Lists) where they seek technological advice from their peers [4].
1990 Capturing what developers discuss on these platforms offers an insight into the
1991 frustrations developers face when using different software components as shown
1992 by recent works [38, 187, 300, 329, 360]. However, to our knowledge, no studies
1993 have yet analysed what developers struggle with when using the new generation of
1994 *intelligent* services. Given the re-emergent interest in AI and the anticipated value
1995 from this technology [219], a better understanding of issues faced by developers
1996 will help us improve the quality of services. Our hypothesis is that application
1997 developers do not fully appreciate the probabilistic nature of these services, nor do
1998 they have sufficient appreciation of necessary background knowledge—however, we
1999 do not know the specific areas of concern. The motivation for our study is to inform
2000 API designers on which aspects to focus in their documentation, education, and
2001 potentially refine the design of the end-points.

2002 This study involves an investigation of 1,825 Stack Overflow (SO) posts regarding
2003 one of the most mature types of IWSs—computer vision services (CVSs)—dating
2004 from November 2012 to June 2019. We adapt existing methodologies of prior SO
2005 analyses [38, 340] to extract posts related to CVSs. We then apply two existing SO
2006 question classification schemes presented at ICPC and ICSE in 2018 and 2019 [4, 39].
2007 These previous studies focused on mobile apps and web applications. Although not
2008 a direct motivation, our work also serves as a validation of the applicability of these
2009 two issue classification taxonomies [4, 39] in the context of IWSs (hence potential
2010 for generalisation). Additionally our work is the first—to our knowledge—to *test*
2011 the applicability of these taxonomies in a new study.

2012 The taxonomies in previous works focus on the specific aspects from the domain
2013 (e.g. API usage, specificity within the documentation etc.) and as such do not
2014 deeply consider the learning gap of an application developer. To explore the API
2015 learning implications raised by our SO analysis, we applied an additional lens of
2016 two taxonomies from the field of pedagogy. This was motivated by the need to offer
2017 an insight into the work needed to help developers learn how to use these relatively
2018 new services.

2019 The key findings of our study are:

- 2020 • The primary areas that developers raise as issues reflect a relatively primitive
2021 understanding of the underlying concepts of data-driven ML approaches used.
2022 We note this via the issues raised due to conceptual misunderstanding and
2023 confusion in interpreting errors,
- 2024 • Developers predominantly encounter a different distribution of issue types than
2025 were reported in previous studies, indicating the complexity of the technical
2026 domain has a non-trivial influence on intelligent API usage; and
- 2027 • Most of these issues can be resolved with better documentation, based on our
2028 analysis.

2029 The paper also offers a data-set as an additional contribution to the research
2030 community and to permit replication [405]. The paper structure is as follows:
2031 Section 5.2 provides motivational examples to highlight the core focus of our study;
2032 Section 5.3 provides a background on prior studies that have mined SO to gather
2033 insight into the software engineering community; Section 5.4 describes our study
2034 design in detail; Section 5.5 presents the findings from the SO extraction; Section 5.6
2035 offers an interpretation of the results in addition to potential implications that arise
2036 from our work; Section 5.7 outlines the limitations of our study; concluding remarks
2037 are given in Section 5.8.

2038 5.2 Motivation

2039 “Intelligent” services are often available as a cloud end-point and provide devel-
2040 opers a friendly approach to access recent AI/ML advances without being experts
2041 in the underlying processes. Figure 5.1 highlights how these services abstract
2042 away much of the technical know-how needed to create and operationalise these
2043 IWSs [263]. In particular, they hide information about the training algorithm and

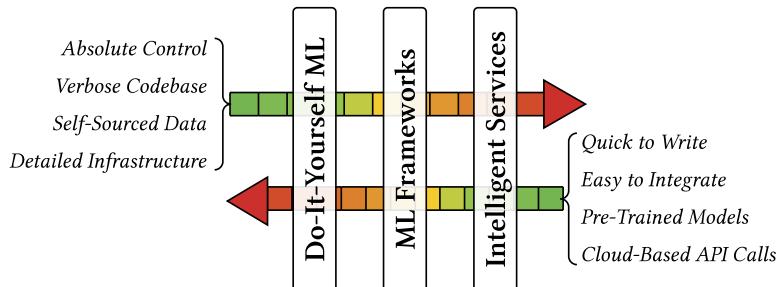


Figure 5.1: Some traits of Intelligent Services vs. ‘Do-It-Yourself’ ML. Green-to-red arrows indicate the presence of these traits. *Adapted from Ortiz [263].*

2044 data-sets used in training, the evaluation procedures, the optimisations undertaken,
2045 and—surprisingly—they often do not offer a properly versioned end-point [87, 260].
2046 That is, the cloud vendors may change the behaviour of the services without sufficient
2047 transparency.

The trade-off towards ease of use for application developers, coupled with the current state of documentation (and assumed developer background) has a cost as reflected in the increasing discussions on developer communities such as SO (see Figure 5.2). To illustrate the key concerns, we list below a few up-voted questions:

- **unsure of ML specific vocabulary:** “*Though it’s now not SO clear to me what ‘score’ actually means.*” [449]; “*I’m trying out the [IWS], and there’s a score field that returns that I’m not sure how to interpret [it].*” [463]
 - **frustrated about non-deterministic results:** “*Often the API has troubles in recognizing single digits... At other times Vision confuses digits with letters.*” [462]; “*Is there a way to help the program recognize numbers better, for example limit the results to a specific format, or to numbers only?*” [459]
 - **unaware of the limitations behind the services:** “*Is there any API available where we can recognize human other body parts (Chest, hand, legs and other parts of the body), because as per the Google vision API it’s only able to detect face of the human not other parts.*” [443]
 - **seeking further documentation:** “*Does anybody know if Google has published their full list of labels ([‘produce’, ‘meal’, ...]) and where I could find that? Are those labels structured in any way? - e.g. is it known that ‘food’ is a superset of ‘produce’, for example.*” [446]

The objective of our study is to better understand the nature of the questions that developers raise when using IWSs, in order to inform the service designers and documenters. In particular, the knowledge we identify can be used to improve the documentation, educational material and (potentially) the information contained in the services' response objects—these are the main avenues developers have to learn and reason about when using these services. There is previous work that has investigated issues raised by developers [4, 39, 340]. We build on top of this work by adapting the study methodology and apply the taxonomies offered to identify the nature of the issues and this results in the following research questions in this paper:

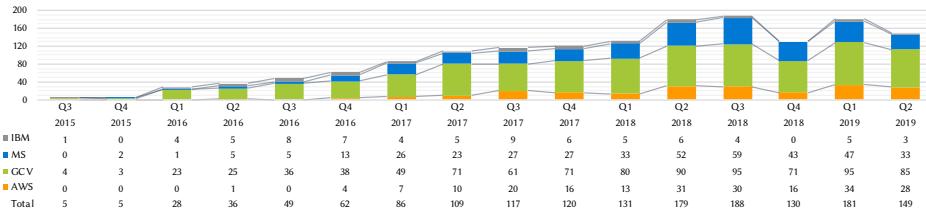


Figure 5.2: Trend of posts, where IBM = IBM Watson Visual Recognition, MS = Azure Computer Vision, AWS = AWS Rekognition and GCV = Google Cloud Vision. Three MS posts from Q4 2012, Q3 2013 and Q4 2013 have been removed for graph clarity.

- 2076 **RQ1. How do developers mis-comprehend IWSs as presented within SO**
 2077 **pain-points?** While the AI community is well aware in the nuances that
 2078 empower IWSs, such services are being released for application developers
 2079 who may not be aware of their limitations or how they work. This is
 2080 especially the case when machine intelligence is accessed via web-based
 2081 APIs where such details are not fully exposed.
- 2082 **RQ2. Are the distribution of issues similar to prior studies?** We compare
 2083 how the distributions of previous studies' of posts about conventional,
 2084 deterministic API services differ from those of IWSs. By assessing the
 2085 distribution of IWSs' issues against similar studies that focus on mobile
 2086 and web development, we identify whether a new taxonomy is needed
 2087 specific to AI-based services, and if gaps specific to AI knowledge exist
 2088 that need to be captured in these taxonomies.

2089 5.3 Background

2090 The primary goal of analysing issues is to better understand the root causes. Hence,
 2091 a good issue classification taxonomy should ideally capture the underlying causal
 2092 aspects (instead of pure functional groupings) [76]. Although this idea (of cause
 2093 related classification) is not new (Chillarege advocated for it in this TSE paper in
 2094 1992), this is not a universally followed approach when studying online discussions
 2095 and some recent works have largely classified issues into the “*what is*” and not
 2096 “*how to fix it*” [28, 38, 349]. They typically (manually) classify discussion into
 2097 either *functional areas* (e.g., Website Design/CSS, Mobile App Development, .NET
 2098 Framework, Java [28]) or *descriptive areas* (e.g., Coding Style/Practice, Problem/-
 2099 Solution, Design, QA [28, 349]). As a result, many of these studies do not give
 2100 us a prioritised means of targeted attack on how to *resolve* these issues with, for
 2101 example, improved documentation. Interestingly, recent taxonomies that studied SO
 2102 data (Aghajani et al. [4] and Beyer et al. [39]) were causal in nature and developed to
 2103 understand discussions related to mobile and web applications. However, issues that
 2104 arise when developers use IWSs have not been studied, nor do we know if existing
 2105 issue classification taxonomies are sufficient in this domain.

2106 Researchers studying APIs have also attempted to understand developer's opin-
 2107 ions towards APIs [349], categorise the questions they ask about these APIs [28,

30, 39, 300], and understand API related documentation and usage issues [4, 5, 8, 28, 162, 340]. These studies often employ automation to assist in the data analysis stages of their research. Latent Dirichlet Allocation [8, 28, 300, 349] is applied for topic modelling and other ML techniques such as Random Forests [39], Conditional Random Fields [5] or Support Vector Machines [39, 162] are also used.

However, automatic techniques are tuned to classify into *descriptive* categories, that is, they help paint a landscape of *what is*, but generally do not address the causal factors to address the issues in great detail. For example, functional areas such as ‘Website Design’ [28], ‘User Interface’ [38] or ‘Design’ [350] result from such analyses. These automatic approaches are generally non-causal, making it hard to address reasons for *why* developers are asking such questions. However, not all studies in the space use automatic techniques; other studies employ manual thematic analysis [4, 30, 340] (e.g., card sorting) or a combination of both [38, 39, 300, 347]. Our work uses a manual approach for classification, and we use taxonomies that are more causally aligned allowing our findings to be directly useful in terms of addressing the issues.

Evidence-based software engineering [193] has helped shape the last 15 years worth of research, but the reliability of such evidence has been questioned [181, 183, 320]. Replication studies, especially in empirical works, can give us the confidence that existing results are adaptable to new domains; in this context, we extend (to IWSs) and work with study methods developed in previous works.

5.4 Method

5.4.1 Data Extraction

This study initially attempted to capture SO posts on a broad range of many IWSs by identifying issues related to four popular IWS cloud providers: Google Cloud [411], AWS [386], Azure [425] and IBM Cloud [421]. We based our selection criteria on the prominence of the providers in industry (Google, Amazon, Microsoft, IBM) and their ubiquity in cloud platform services. Additionally, in 2018, these services were considered the most adopted cloud vendors for enterprise applications [294].

However, during the filtering stage (see Section 5.4.2), we decided to focus on a subset of these services, computer vision, as these are one of the more mature and stable ML/AI-based services with widespread and increasing adoption in the developer community (see Figure 5.2). We acknowledge other services beyond the four analysed provide similar capabilities [399, 400, 407, 420, 472, 473] and only English-speaking services have been selected, excluding popular services from Asia (e.g., [397, 398, 419, 438, 439])—see Section 5.7. For comprehensiveness, we explain below our initial attempts to extract *all* IWSs.

5.4.1.1 Defining a list of IWSs

As there exists no global ‘list’ of IWSs to search on, we needed to derive a *corpus of initial terms* to allow us to know *what* to search for on the Stack Exchange Data

2148 Explorer¹ (SEDE). We began by looking at different brand names of cloud services
2149 and their permutations (e.g., Google Cloud Services and GCS) as well as various
2150 ML-related products (e.g., Google Cloud ML). To do this, we performed extensive
2151 Google searches² in addition to manually reviewing six ‘overview’ pages of the
2152 relevant cloud platforms. We identified 91 initial IWSs to incorporate into our
2153 search terms³.

2154 5.4.1.2 *Manual search for relevant, related terms*

2155 We then ran a manual search² on each term to determine if these terms were relevant.
2156 We did this by querying each term within SO’s search feature, reviewing the titles
2157 and body post previews of the first three pages of results (we did not review the
2158 answers, only the questions). We also noted down the user-defined *Tags* of each post
2159 (up to five per question); by clicking into each tag, we could review similar tags (e.g.,
2160 ‘project-oxford’ for ‘azure-cognitive-services’) and check if the tag had synonyms
2161 (e.g., ‘aws-lex’ and ‘amazon-lex’). We then compiled a *corpus of tags* consisting of
2162 31 terms.

2163 5.4.1.3 *Developing a search query*

2164 We recognise that searching SEDE via *Tags* exclusively can be ineffective (see [28,
2165 340]). To mitigate this, we produced a *corpus of title and body terms*. Such terms
2166 are those that exist within the title and body of the posts to reflect the ways in
2167 which individual developers commonly use to refer to different IWSs. To derive
2168 at such a list, we performed a search^{2,3} of the 31 tags above in SEDE, filtering
2169 out posts that were not answers (i.e., questions only) as we wanted to see how
2170 developers *phrase* their questions. For each search, we extracted a random sample
2171 of 100 questions (400 total for each service) and reviewed each question. We noted
2172 many patterns in the permutations of how developers refer to these services, such
2173 as: common misspellings ('bind' vs. 'bing'); brand misunderstanding ('Microsoft
2174 computer vision' vs. 'Azure computer vision'); hyphenation ('Auto-ML' vs. 'Auto
2175 ML'); UK and US English ('Watson Analyser' vs. 'Watson Analyzer'); and, the
2176 use of apostrophes, plurals, and abbreviations ('Microsoft's Computer Vision API',
2177 'Microsoft Computer Vision Services', 'GCV' vs. 'Google Cloud Vision'). We
2178 arrived at a final list of 229 terms compromising all of the IWSs provided by
2179 Google, Amazon, Microsoft and IBM as of January 2019³.

2180 5.4.1.4 *Executing our search query*

2181 Our next step was to perform a case-insensitive search of all 229 terms within the
2182 body or title of posts. We used Google BigQuery’s public data-set of SO posts⁴ to
2183 overcome SEDE’s 50,000 row limit and to conduct a case-insensitive search. This

¹<http://data.stackexchange.com/stackoverflow>

²This search was conducted on 17 January 2019

³For reproducibility, this is available at <http://bit.ly/2ZcwNJO>.

⁴<http://bit.ly/2LrN7OA>

²¹⁸⁴ search was conducted on 10 May 2019, where we extracted 21,226 results. We then
²¹⁸⁵ performed several filtering steps to cleanse our extracted data, as explained below.

²¹⁸⁶ 5.4.2 Data Filtering

²¹⁸⁷ 5.4.2.1 Refining our inclusion/exclusion criteria

²¹⁸⁸ We performed an initial manual filtering of the 50 most recent posts (sorted by
²¹⁸⁹ descending *CreationDate* values) of the 21,226 posts above, assessing the suitability
²¹⁹⁰ of the results and to help further refine our inclusion and exclusion criteria. We
²¹⁹¹ did note that some abbreviations used in the search terms (e.g., ‘GCV’, ‘WCS’⁵),
²¹⁹² resulting in irrelevant questions in our result set. We therefore removed abbreviations
²¹⁹³ from our search query and consolidated all overlapping terms (e.g., ‘Google Vision
²¹⁹⁴ API’ was collapsed into ‘Google Vision’).

²¹⁹⁵ We also recognised that 21,226 results would be non-trivial to analyse without
²¹⁹⁶ automated techniques. As we wanted to do manual qualitative analysis, we reduced
²¹⁹⁷ our search space to 27 search terms of just the CVSs within the original corpus of
²¹⁹⁸ 229 terms. These were Google Cloud Vision [411], AWS Rekognition [386], Azure
²¹⁹⁹ Computer Vision [425], and IBM Watson Visual Recognition [421]. This resulted
²²⁰⁰ in 1,425 results that were extracted on 21 June 2019. The query used and raw results
²²⁰¹ are available online in our supplementary materials [405].

²²⁰² 5.4.2.2 Duplicates

²²⁰³ Within 1,425 results, no duplicate questions were noted, as determined by unique
²²⁰⁴ post ID, title or timestamp.

²²⁰⁵ 5.4.2.3 Automated and manual filtering

²²⁰⁶ To assess the suitability and nature of the 1,425 questions extracted, the first author
²²⁰⁷ began with a manual check on a randomised sample of 50 questions. As the questions
²²⁰⁸ were exported in a raw CSV format (with HTML tags included in the post’s body), we
²²⁰⁹ parsed the questions through an ERB templating engine script⁶ in which the ID, title,
²²¹⁰ body, tags, created date, and view, answer and comment counts were rendered for
²²¹¹ each post in an easily-readable format. Additionally, SQL matches in the extraction
²²¹² process were also highlighted in yellow (i.e., in the body of the post) and listed at
²²¹³ the top of each post. These visual cues helped to identify 3 false positive matches
²²¹⁴ where library imports or stack traces included terms within our corpus of 26 CVS
²²¹⁵ terms. For example, `aws-java-sdk-rekognition:jar` is falsely matched as a
²²¹⁶ dependency within an unrelated question. As such exact matches would be hard to
²²¹⁷ remove without the use of regular expressions, and due to the low likelihood (6%)
²²¹⁸ of their appearance, we did not perform any followup automatic filtering.

⁵Watson Cognitive Services

⁶We make this available for future use at: <http://bit.ly/2NqBB70>

2219 **5.4.2.4 Classification**

2220 Our 1,425 posts were then split into 4 additional random samples (in addition to the
2221 random sample of 50 above). 475 posts were classified by the first author and three
2222 other research assistants, software engineers with at least 2 years industry experience,
2223 assisted to classify the remaining 900. This left a total of 1,375 classifications
2224 made by four people plus an additional 450 classifications made from reliability
2225 analysis, in which the remaining 50 posts were classified nine times (as detailed in
2226 Section 5.4.3.1). Thus, a total of 1,825 classifications were made from the original
2227 1,425 posts extracted.

2228 Whilst we could have chosen to employ topic modelling, these are too descriptive
2229 in nature (as discussed in Section 5.3). Moreover, we wanted to see if prior
2230 taxonomies can be applied to IWSs (as opposed to creating a new one) and compare
2231 if their distributions are similar. Therefore, we applied the two existing taxonomies
2232 described in Section 5.3 to each post; (i) a documentation-specific taxonomy that ad-
2233 dresses issues directly resulting from documentation, and (ii) a generalised taxonomy
2234 that covers a broad range of SO issues in a well-defined software engineering area
2235 (specifically mobile app development). Aghajani et al.'s documentation-specific
2236 taxonomy (Taxonomy A) is multi-layered consisting of four dimensions and 16
2237 sub-categories [4]. Similarly, Beyer's SO generalised post classification taxonomy
2238 (Taxonomy B) consists of seven dimensions [39]. We code each dimension with
2239 a number, X , and each sub-category with a letter y : (Xy). We describe both tax-
2240 onomies in detail within Table 5.1. Where a post was included in our results but
2241 not applicable to IWSs (see Section 5.4.2.3) or not applicable to a taxonomy dimen-
2242 sion/category, then the post was flagged for removal in further analysis. Table 5.1
2243 presents *our understanding* of the respective taxonomies; our intent is not to method-
2244 logically replicate Aghajani et al. or Beyer et al.'s studies in the IWS domain, rather
2245 to acknowledge related work in the area of SO classification and reduce the need to
2246 synthesise a new taxonomy. We baseline all coding against *our interpretation only*.
2247 Our classifications are therefore independent of the previous authors' findings.

2248 **5.4.3 Data Analysis**

2249 **5.4.3.1 Reliability of Classification**

2250 To measure consistency of the categories assigned by each rater to each post, we
2251 utilised both intra- and inter-rater reliability [233]. As verbatim descriptions from
2252 dimensions and sub-categories were considered quite lengthy from their original
2253 sources, all raters met to agree on a shared interpretation of the descriptions, which
2254 were then paraphrased as discussed in the previous subsection and tabulated in
2255 Table 5.1. To perform statistical calculations of reliability, each category was as-
2256 signed a nominal value and a random sample of 50 posts were extracted. Two-phase
2257 reliability analysis followed.

2258 Firstly, intra-rater agreement by the first author was conducted twice on 28 June
2259 2019 and 9 August 2019. Secondly, inter-rater agreement was conducted with the
2260 remaining four co-authors in addition to three research assistants within our research

Table 5.1: Descriptions of dimensions (■) and sub-categories (→) from both taxonomies used.

A Documentation-specific classification (Aghajani et al. [4])	
A-1	■ Information Content (What)
A-1a	→ <i>Correctness</i>
A-1b	→ <i>Completeness</i>
A-1c	→ <i>Up-to-dateness</i>
A-2	■ Information Content (How)
A-2a	→ <i>Maintainability</i>
A-2b	→ <i>Readability</i>
A-2c	→ <i>Usability</i>
A-2d	→ <i>Usefulness</i>
A-3	■ Process-Related
A-3a	→ <i>Internationalisation</i>
A-3b	→ <i>Contribution-Related</i>
A-3c	→ <i>Configuration-Related</i>
A-3d	→ <i>Implementation-Related</i>
A-3e	→ <i>Traceability</i>
A-4	■ Tool-Related
A-4a	→ <i>Tooling Bugs</i>
A-3b	→ <i>Tooling Discrepancy</i>
A-3c	→ <i>Tooling Help Required</i>
A-3d	→ <i>Tooling Migration</i>
B Generalised classification (Beyer et al. [39])	
B-1	■ API usage
B-2	■ Discrepancy
B-3	■ Errors
B-4	■ Review
B-5	■ Conceptual
B-6	■ API change
B-7	■ Learning

2261 group in mid-August 2019. Thus, the 50 posts were classified an additional nine
2262 times, resulting in 450 classifications for reliability analysis. We include these
2263 classifications in our overall analysis.

2264 At first, we followed methods of reliability analysis similar to previous SO
2265 studies (e.g., [340]) using the percentage agreement metric that divides the number
2266 of agreed categories assigned per post by the total number of raters [233]. However,
2267 percentage agreement is generally rejected as an inadequate measure of reliability
2268 analysis [81, 148, 200] in statistical communities. As we used more than 2 coders
2269 and our reliability analysis was conducted under the same random sample of 50
2270 posts, we applied *Light's Kappa* [211] to our ratings, which indicates an overall
2271 index of agreement. This was done using the `irr` computational R package [127]
2272 as suggested in [148].

2273 **5.4.3.2 Distribution Analysis**

2274 In order to compare the distribution of categories from our study with previous studies
2275 we carried out a χ^2 test. We selected a χ^2 test as the following assumptions [321]
2276 are satisfied: (i) the data is categorical, (ii) all counts are greater than 5, and (iii)
2277 we can assume simple random sampling. The null hypothesis describes the case
2278 where each population has the same proportion of observations and the alternative
2279 hypothesis is where at least one of the null hypothesis statements is false. We chose
2280 a significance value, α , of 0.05 following a standard rule of thumb. As to the best
2281 of our knowledge this is the first statistical comparison using Taxonomy A and B on
2282 SO posts. To report the effect size we selected Cramer's Phi, ϕ_c which is well suited
2283 for use on nominal data [321].

2284 **5.5 Findings**

2285 We present our findings from classifying a total of 1,825 SO posts aimed at answering
2286 RQs 1 and 2. 450 posts were classified using Taxonomies A and B for reliability
2287 analysis as described in Section 5.4.3.1 and the remaining 1,375 posts were classified
2288 as per Section 5.4.2.4. A summary of our classification using Taxonomies A and B
2289 is shown in Figure 5.3.

2290 **5.5.1 Post classification and reliability analysis**

2291 When undertaking the classification, we found that 238 issues (13.04%) did not
2292 relate to IWSs directly. For example, library dependencies were still included in
2293 a number of results (see Section 5.4.2.3), and we found there to be many posts
2294 discussing Android's Mobile Vision API as Google (Cloud) Vision. These issues
2295 were flagged and ignored for further analysis (see Section 5.4.2.4).

2296 For our reliability analysis, we classified a total of 450 posts of which 70 posts
2297 were flagged as irrelevant. Landis and Koch [206] provide guidelines to interpret
2298 kappa reliability statistics, where $0.00 \leq \kappa \leq 0.20$ indicates *slight* agreement and
2299 $0.21 \leq \kappa \leq 0.40$ indicates *fair* agreement. Despite all raters meeting to agree
2300 on a shared interpretation of the taxonomies (see Section 5.4.3.1) our inter-rater

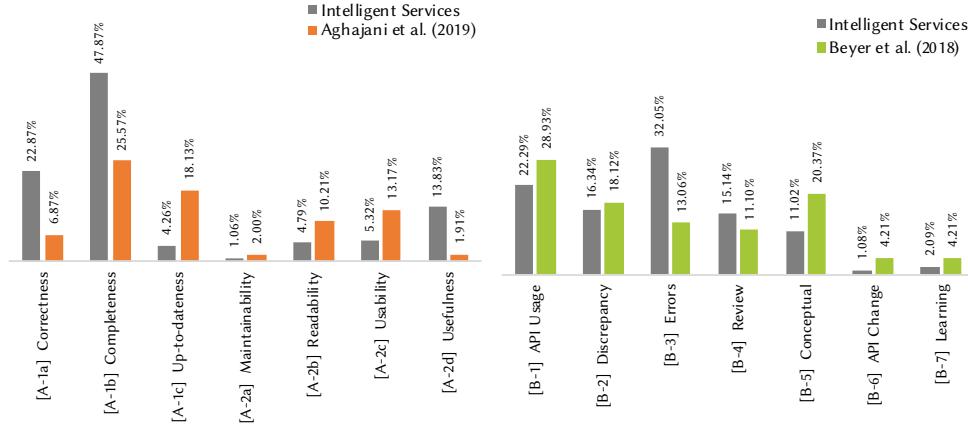


Figure 5.3: *Left:* Documentation-specific classification taxonomy results highlights a mostly similar distribution to that of Aghajani et al.’s findings [4]. *Right:* Generalised classification taxonomy results highlight differences from more mature fields (i.e., Android APIs in Beyer et al. [39]) to less mature fields (i.e., IWSs).

measures aligned *slightly* (0.148) for Taxonomy A and *fairly* (0.295) for Taxonomy B. We report further in Section 5.7.

5.5.2 Developer Frustrations

We found Beyer et al.’s high-level abstraction taxonomy (Taxonomy B) was able to classify 86.52% of posts. 10.30% posts were assigned exclusively under Aghajani et al.’s documentation-specific taxonomy (Taxonomy A). We found that developers do not generally ask questions exclusive to documentation, and typically either pair documentation-related issues to their own code or context. The following two subsections further explain results from both Taxonomy A and B’s perspective.

5.5.2.1 Results from Aghajani et al.’s taxonomy

Results for Aghajani et al.’s low-level documentation taxonomy (Taxonomy A), indicates that most discussion on SO does not directly relate to documentation about an IWS. We did not find any process-related (A-3) or tool-related (A-4) questions as, understandably, the developers who write the documentation of the IWSs would not be posting questions of such nature on SO. One can *infer* documentation-related issues from posts (i.e., parts of the documentation *lacking* that may cause the issue posted). However, there are few questions that *directly* relate to documentation of IWSs.

Few developers question or ask questions directly about the API documentation, but some (47.87%) posts ask for additional information to understand the API (**completeness (A-1b)**), for example: “*Is there a full list of potential labels that Google’s Vision API will return?*” [446]; “*There seems to be very little to no documentation for AWS iOS text recognition inside an image*” [444].

22.87% of posts question the **accuracy (A-1a)** of certain parts of the cloud docu-

2325 mentation, especially in relation to incorrect quotas and limitations: “*Are the Cloud*
2326 *Vision API limits in documentation correct?*” [457], “*According to the Google Vision*
2327 *documentation, the maximum number of image files per request is 16. Elsewhere,*
2328 *however, I’m finding that the maximum number of requests per minute is as high as*
2329 *1800.*” [442].

2330 There are also many references (23.94%) addressing the confusing nature of
2331 some documentation, indicating that the **readability, usability and usefulness of**
2332 **the documentation (A-2b, A-2c and A-2d)** could be improved. For example, “*Am*
2333 *I encoding it correctly? The docs are quite vague.*” [440], “*The aws docs for this*
2334 *are really confusing.*” [469].

2335 5.5.2.2 *Results from Beyer et al.’s taxonomy*

2336 We found that a majority (32.05%) of posts are primarily **error-related questions**
2337 **(B-3)**, including a dump of the stack trace or exception message from the service’s
2338 programming-language SDK (usually Java, Python or C#) that relates to a specific
2339 error. For example: “*I can’t fix an error that’s causing us to fall behind.*” [466]; “*I’m*
2340 *using the Java Google Vision API to run through a batch of images... I’m now getting*
2341 *a channel closed and ClosedChannelException error on the request.*” [460].

2342 **API usage questions (B-1)** were the second highest category at 22.29% of
2343 posts. Reading the questions revealed that many developers present an insufficient
2344 understanding of the behaviour, functional capability and limitation of these services
2345 and the need for further data processing. For example, while Azure provides an
2346 image captioning service, this is not universal to all CVSSs: “*In Amazon Rekognition*
2347 *for image processing how do I get the caption for an image?*” [451]. Similarly,
2348 OCR-related and label-related questions often indicate interest in cross-language
2349 translation, where a separate translation service would be required: “*Can Google*
2350 *Cloud Vision generate labels in Spanish via its API?*” [465]; “[*How can I] specify*
2351 *language for response in Google Cloud Vision API*” [452]; “*When I request a text*
2352 *detection of an image, it gives only English Alphabet characters (characters without*
2353 *accents) which is not enough for me. How can I get the UTF-32 characters?*” [447].

2354 It was commonplace to see questions that demonstrate a lack of depth in under-
2355 standing and appreciating how these services work, instead posting simple debugging
2356 questions. For instance, in the 11.02% of **conceptual-related questions (B-5)** that
2357 we categorised, we noticed causal links to a misunderstanding (or lack of aware-
2358 ness) of the vocabulary used within computer vision. For example: “*The problem*
2359 *is that I need to know not only what is on the image but also the position of that*
2360 *object. Some of those APIs have such feature but only for face detection.*” [458];
2361 “*I want to know if the new image has a face similar to the original image.... [the*
2362 *service] can identify faces, but can I use it to get similar faces to the identified face*
2363 *in other images?*” [450]. It is evident that some application developers are not aware
2364 of conceptual differences in computer vision such as object/face *detection* versus
2365 *localisation* versus *recognition*.

2366 In the 16.34% of **discrepancy-related questions (B-2)**, we see further unaware-
2367 ness from developers in how the underlying systems work. In OCR-related questions,
2368 developers do not understand the pre-processing steps required before an OCR is

2369 performed. In instances where text is separated into multiple columns, for example,
 2370 text is read top-down rather than left-to-right and segmentation would be required
 2371 to achieve the expected results. For example, “*it appears that the API is using some*
 2372 *kind of logic that makes it scan top to bottom on the left side and moving to right*
 2373 *side and doing a top to bottom scan.*” [464]; “*this method returns scanned text in*
 2374 *wrong sequence... please tell me how to get text in proper sequence.*” [470].

2375 A number of **review-related questions (B-4)** (15.14%) seem to provide some
 2376 further depth in understanding the context to which these systems work, where train-
 2377 ing data (or training stages) are needed to understand how inferences are made: “*How*
 2378 *can we find an exhaustive list (or graph) of all logos which are effectively recognized*
 2379 *using Google Vision logo detection feature?*” [468]; “*when object banana is detected*
 2380 *with accuracy greater than certain value, then next action will be dispatched... how*
 2381 *can I confidently define and validate the threshold value for each item?*” [454].

2382 **API change (B-6)** was shown in 1.08% of posts, with evolution of the services
 2383 occurring (e.g., due to new training data) but not necessarily documented “*Recently*
 2384 *something about the Google Vision API changed... Suddenly, the API started to*
 2385 *respond differently to my requests. I sent the same picture to the API today, and I*
 2386 *got a different response (from the past).*” [467].

2387 5.5.3 Statistical Distribution Analysis

2388 We obtained the following results $\chi^2 = 131.86$, $\alpha = 0.05$, $p \text{ value} = 2.2 \times 10^{-16}$ and
 2389 $\phi_c = 0.362$ from our distribution analysis with Taxonomy A to compare our study
 2390 with that of Aghajani et al. [4]. Comparing our study to Beyer et al. [39] produced the
 2391 following results $\chi^2 = 145.58$, $\alpha = 0.05$, $p \text{ value} = 2.2 \times 10^{-16}$ and $\phi_c = 0.252$.
 2392 These results show that we are able to reject the null hypothesis that the distribution
 2393 of posts using each taxonomy was the same as the comparison study. While there are
 2394 limited guidelines for interpreting ϕ_c when there is no prior information for effect
 2395 size [335], Sun et al. suggests the following: $0.07 \leq \phi_c \leq 0.20$ indicates a *small*
 2396 effect, $0.21 \leq \phi_c \leq 0.35$ indicates a *medium* effect, and $0.35 > \phi_c$ indicates a *large*
 2397 effect. Based on this criteria we obtained a *large* effect size for the documentation-
 2398 specific classification (Taxonomy A) and a *medium* effect size for the generalised
 2399 classification (Taxonomy B).

2400 5.6 Discussion

2401 5.6.1 Answers to Research Questions

2402 5.6.1.1 *How do developers mis-comprehend IWSs as presented within SO pain-*
 2403 *points? (RQ1)*

2404 Upon meeting to discuss the discrepancies between our categorisation of IWS usage
 2405 SO posts, we found that our interpretations of the *posts themselves* were largely sub-
 2406 jective. For example, many posts presented multi-faceted dimensions for Taxonomy
 2407 B; Beyer et al. [39] argue that a post can have more than one question category and

2408 therefore multi-label classification is appropriate at times. We highlight this further
2409 in the threats to validity (Section 5.7).

2410 We have to define the context of IWSs to address RQ1. We use the concept
2411 of a “technical domain” [25] to define this context. A technical domain captures
2412 the domain-specific concerns that influence the non-functional requirements of a
2413 system [25]. In the context of IWSs, the technical domain includes exploration, data
2414 engineering, distributed infrastructure, training data, and model characteristics as
2415 first class citizens [25]. We would then expect to see posts on SO related to these
2416 core concerns.

2417 In Figure 5.3, for the documentation-specific classification, the majority of posts
2418 were classified as **Completeness (A1-b)** related (47.87%). An interpretation for this
2419 is that the documentation does not adequately cover the technical domain concerns.
2420 Comments by developers such as “*I'm searching for a list of all the possible image*
2421 *labels that the Google Cloud Vision API can return?*” [445] indicates the documen-
2422 *tation does not adequately describe the training data for the API—developers do*
2423 *not know the required usage assumptions. Another quote from a developer, “Can*
2424 *Google Cloud Vision generate labels in Spanish via its API? ... [Does the API]*
2425 *allow to select which language to return the labels in?”* [465] points to a lack of
2426 details relating to the characteristics of the models used by the API. It would seem
2427 that developers are unaware of aspects of the technical domain concerns.

2428 The next most frequent category is **Correctness (A-1a)** with 22.87% of posts. In
2429 the context of the technical domain there are many limits that developers need to be
2430 aware of: range and increments of a model score [87]; required data pre-processing
2431 steps for optimal performance; and features provided by the models (as explained in
2432 Section 5.5.2.2). Considering the relation between technical concerns and software
2433 quality, developers are right to question providers on correctness; “*Are the Cloud*
2434 *Vision API limits in documentation correct?*” [457].

2435 5.6.1.2 *Are the distribution of issues similar to prior studies? (RQ2)*

2436 Visual inspection of Figure 5.3 shows that the distributions for the documentation-
2437 specific classification and the generalised classification are different (compared to
2438 prior studies). As a sanity check we conducted a χ^2 test and calculated the effect
2439 size ϕ_c . We were able to reject the null hypothesis for both classification schemes,
2440 that the distribution of issues were the same as the previous studies (see Section 5.5).
2441 We now discuss the most prominent differences between our study and the previous
2442 studies.

2443 In the context of IWS SO posts, Taxonomy B suggests that Errors (B-3) are
2444 discussed most amongst developers. These results are in contrast to similar studies
2445 made in more *mature* API domains, such as Mobile Development [26, 27, 38, 39, 300]
2446 and Web Development [347]. Here, API Usage (B-1) is much more frequently
2447 discussed, followed by Conceptual (B-5), Discrepancy (B-2) and Errors (B-3). We
2448 argue in the following section that an improved developer understanding can be
2449 achieved by educating them about the IWS lifecycle and the ‘whole’ system that
2450 wraps such services.

2451 In the Android study API usage questions (B-1) were the highest category
2452 (28.93% compared to 22.29% in our study). As stated in the analysis of the Error
2453 questions this discrepancy could be due to the maturity of the domain. However,
2454 another explanation could be the scope of the two individual studies. Beyer et al. [39]
2455 used a broad search strategy consisting of posts tagged Android. This search term
2456 fetches issues related to the entire Android platform which is significantly larger
2457 than searching for computer vision APIs using 229 search terms. As a consequence
2458 of more posts and more APIs there would be use cases resulting in additional posts
2459 related to API Usage (B-1).

2460 Applying existing SO taxonomies allowed us to better understand the distribution
2461 of the issues across different domains. In particular, the issues raised around IWSs
2462 appear to be primarily due to poor documentation, or insufficient explanation around
2463 errors and limitations. Hence, many of the concerns could be addressed by adding
2464 more details to the end-point descriptions, and by providing additional information
2465 around how these services are designed to work.

2466 5.6.2 The Developer’s Learning Approach

2467 In this subsection, we offer an explanation as to why developers are complaining
2468 about certain things when trying to use IWSs on SO (RQ1), as characterised through
2469 the use of prior SO classification frameworks (RQ2). This is described through
2470 the theoretical lenses of two learning taxonomies: Bloom’s context complexity and
2471 intellectual ability taxonomy, and the Structure of the Observed Learning Outcome
2472 (SOLO) taxonomy (i.e., the nature by which developer’s learn). We argue that the
2473 issues with using IWSs relating to the lower-levels of these learning taxonomies
2474 are easily solvable by slight fixes and improvements to the documentation of these
2475 services. However, the higher dimensions of these taxonomies demand far more
2476 rigorous mitigation strategies than documentation alone (potentially more structured
2477 education). Thus, many of the questions posted are from developers who are *learning*
2478 to understand the domain of IWSs and AI, and (hence) both SOLO and Bloom’s
2479 taxonomies are applicable for this discussion—as described below within the context
2480 of our domain—as pedagogical aides.

2481 5.6.2.1 Bloom’s Taxonomy

2482 The cognitive domain under Bloom’s taxonomy [44] consists of six objectives.
2483 Within the context of IWSs, developers are likely to ask questions due to causal links
2484 that exist in the following layers of Bloom’s taxonomy: (i) *knowledge*, where the
2485 developer does not remember or know of the basic concepts of computer vision and
2486 AI (in essence, they may think that AI is as smart as a human); (ii) *comprehension*,
2487 where the developer does not understand how to interpret basic concepts, or they
2488 are mis-understanding how they are used in context; (iii) *application*, where the
2489 developer is struggling to apply existing concepts within the context of their own
2490 situation; (iv) *analysis*, where the developer is unable to analyse the results from IWSs
2491 (i.e., understand response objects); (v) *evaluation*, where the developer is unable to
2492 evaluate issues and make use of best-practices when using IWSs; and (vi) *synthesise*,

2493 where the developer is posing creative questions to ask if new concepts are possible
2494 with CVSSs.

2495 **5.6.2.2 SOLO Taxonomy**

2496 The SOLO taxonomy [40] consists of five levels of understanding. The causal
2497 links behind the SO questions we have found relate to the following layers of the
2498 SOLO taxonomy: (i) *pre-structural*, where the developer has a question indicating
2499 incompetence or has little understanding of computer vision; (ii) *uni-structural*,
2500 where the developer is struggling with one key aspect (i.e., a simple question about
2501 computer vision); (iii) *multi-structural*, where the developer is questioning multiple
2502 concepts (independently) to understand how to build their system (e.g., system
2503 integration with the IWS); (iv) *relational*, where the developer is comparing and
2504 contrasting the best ways to achieve something with IWSs; and (v) *extended abstract*,
2505 where the developer poses a question theorising, formulating or postulating a new
2506 concept within IWSs.

Table 5.2: Example Alignments of SO posts to Bloom’s and the SOLO taxonomy.

Issue Quote	Bloom	SOLO
“I’m using Microsoft Face API for a small project and I was trying to detect a face inside a .jpg file in the local system (say, stored in a directory D:\Image\abc.jpg)... but it does not work.” [461]	Knowledge	Pre-Structural
“The problem is that the response JSON is rather big and confusing. It says a lot about the picture but doesn’t say what the whole picture is of (food or something like that).” [441]	Comprehension	Uni-Structural
“The bounding box around individual characters is sometimes accurate and sometimes not, often within the same image. Is this a normal side-effect of a probabilistic nature of the vision algorithm, a bug in the Vision API, or of course an issue with how I’m interpreting the response?” [448]	Comprehension	Multi-Structural
“I’m working on image processing. SO far Google Cloud Vision and Clarifai are the best API’s to detect objects from images and videos, but both API’s doesn’t support object detection from 360 degree images and videos. Is there any solution for this problem?” [455]	Application	Uni-Structural
“Before I train Watson, I can delete pictures that may throw things off. Should I delete pictures of: Multiple dogs, A dog with another animal, A dog with a person, A partially obscured dog, A dog wearing glasses, Also, would dogs on a white background make for better training samples? Watson also takes negative examples. Would cats and other small animals be good negative examples?” [453]	Analysis	Relational

2507 **5.6.2.3 Aligning SO taxonomies to Bloom’s and SOLO taxonomies**

2508 To understand our findings with the lenses of pedagogical aids, we aligned Tax-
2509 onomies A and B to Bloom’s and the SOLO taxonomies for a random sample of 50
2510 issues described in Section 5.4.3.1. To do this, we reviewed all 50 of these SO posted
2511 questions and applied both the Bloom and SOLO taxonomies. The primary author

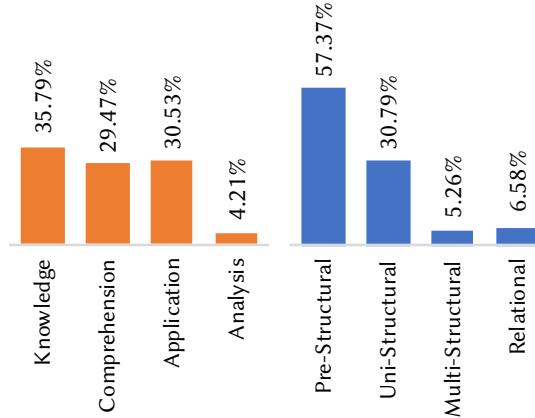


Figure 5.4: Alignment of Bloom (Orange) and SOLO (Blue) taxonomies against Taxonomy A and B dimensions against all 213 classifications made in the random sample of 50 posts.

2512 assigned each of the 50 questions a level within the Bloom and SOLO taxonomies,
 2513 removed out noise (i.e., false positive posts of no relevance to IWSs) and unassigned
 2514 dimensions from reliability agreement, and then compared the relevant dimensions
 2515 of Taxonomy A and B dimensions (not sub-categories). The comparison of align-
 2516 ments of posts to the five SOLO dimensions and six Bloom dimensions are shown
 2517 in Figure 5.4. We acknowledge that this is only an approximation of the current
 2518 state of the developer’s understanding of IWSs. This early model will require further
 2519 studies to perform a more thorough analysis, but we offer this interpretation for early
 2520 discussion.

2521 As shown in Figure 5.4, the bulk of the posts fall in the lower constructs of
 2522 Bloom’s and the SOLO taxonomy. This indicates that modification to certain doc-
 2523 umentation aspects can address many of these issues. For example, many issues
 2524 can be ratified with better descriptions of response data and error messages: “*I was*
 2525 *exploring google vision and in the specific function ‘detectCrops’, gives me the crop*
 2526 *hints. what does this means exactly?”* [456]; “*I am making a very simple API call*
 2527 *to the Google Vision API, but all the time it’s giving me error that ‘google.oauth2’*
 2528 *module not found.”* [471]

2529 However, and more importantly, the higher-construct questions ranging from
 2530 the middle of the third dimensions on are not as easily solvable through improved
 2531 documentation (i.e., apply and multi-structural) which leaves 34.74% (Bloom’s)
 2532 and 11.84% (SOLO) unaccounted for, resolvable only through improved education
 2533 practices.

2534 5.6.3 Implications

2535 5.6.3.1 For Researchers

2536 **Investigate the evolution of post classification** Analysing how the distribution of
 2537 the reported issues changes over time would be an important study. This study could

2538 answer questions such as ‘*Does the evolution of IWSs follow the same pattern as*
2539 *previous software engineering trends such as mobile app or web development?*’ As
2540 with any new emerging field, it is key to analyse how developers perceive such issues
2541 over time. For instance, early issues with web or mobile app development matured
2542 as their respective domain matured, and we would expect similar results to occur
2543 in the IWSs space. Future researchers could plan for a longitudinal study, such as
2544 a long-term survey with developers to gather their insights in this evolving domain,
2545 reviewing case studies of projects that use intelligent web services from now into
2546 the future, or re-mining SO at a later date and comparing the results to this study.
2547 This will help assess evolving trends and characteristics, and determine how and if
2548 the nature of the developer’s experience with IWSs (and AI in general) changes with
2549 time.

2550 **Investigate the impact of technical challenges on API usage** As discussed above,
2551 IWSs have characteristics that may influence API usage patterns and should be
2552 investigated as a further avenue of research. Further mining of open source software
2553 repositories that make use of IWSs could be assessed, thereby investigating if API
2554 patterns evolve with the rise of AI-based applications.

2555 **5.6.3.2 For Educators**

2556 **Education on high-level aspects of IWSs** As demonstrated in our analysis of their
2557 SO posts, many developers appear to be unaware of the higher-level concepts that
2558 exist within the AI and ML realm. This includes the need to pre- and post-process
2559 data, the data dependency and instability that exists in these services, and the specific
2560 algorithms that empower the underlying intelligence and hence their limitations and
2561 characteristics. However, most developers don’t seem to complain about these factors
2562 due to the lack of documentation (i.e., via Taxonomy A). Rather, they are unaware
2563 that such information should be documentation and instead ask generalised and open
2564 questions (i.e., via Taxonomy B). Thus, documentation improvements alone may not
2565 be enough to solve these issues. This results in uncertainty during the preparation
2566 and operation (usage) of such services. Such high-level conceptual information is
2567 currently largely missing in developer documentation for IWSs. Furthermore, many
2568 of the background ML and AI algorithm information needed to understand and use
2569 intelligent systems in context are built within data science (not software engineering)
2570 communities. A possible road-map to mitigate this issue would be the development
2571 of a software engineer’s ‘crash-course’ in ML and AI. The aim of such a course
2572 would encourage software engineers to develop an appreciation of the nuances and
2573 the inherent risks and implications that comes with using IWSs. This could be
2574 taught at an undergraduate level to prepare the next generation of developers of a
2575 ‘programming 2.0’ era. However, the key aspects and implications that are presented
2576 with AI would need to be well-understood before such a course is developed, and
2577 determining the best strategy to curate the content to developers would be best left
2578 to the software engineering education domain. Further investigation in applying
2579 educational taxonomies in the area (such as our attempts to interpret our findings

2580 using Bloom's and the SOLO taxonomies) would need to be thoroughly explored
2581 beforehand.

2582 *5.6.3.3 For Software Engineers*

2583 **Better understanding of intelligent API contextual usage** Our results show that
2584 developers are still learning to use these APIs. We applied two learning perspectives
2585 to interpret our results. In applying the two pedagogical taxonomies to our findings,
2586 we see that most issues seem to fall into the pre-structural and knowledge-based
2587 categories; little is asked of higher level concepts and a majority of issues do not
2588 offer complex analysis from developers. This suggests that developers are struggling
2589 as they are unaware of the vocabulary needed to actually use such APIs, further
2590 reinforcing the need for API providers to write overview documentation (as noted in
2591 prior work [86]) and not just simple endpoint documentation. This said, improved
2592 documentation isn't always enough—as suggested by our discussion in Section 5.6.2,
2593 software engineers should explore further education to attain a greater appreciation
2594 of the nuances of ML when attempting to use these services.

2595 *5.6.3.4 For Intelligent Service Providers*

2596 **Clarify use cases for IWSs** Inspecting SO posts revealed that there is a level of
2597 confusion around the capabilities of different IWSs. This needs to be clarified in
2598 associated API documentation. The complication with this comes with targeting
2599 the documentation such that software developers (who are untrained in the nuances
2600 of AI and ML as per Section 5.6.3.2) can to digest it and apply it in-context to
2601 application development.

2602 **Technical domain matters** More needs to be provided than a simple endpoint
2603 description as conventional APIs offer by describing the whole framework by which
2604 the endpoint sits, giving further context. This said, compared to traditional APIs,
2605 we find that developers complain less about the documentation and more about
2606 shallower issues. All expected pre-processing and post-processing needs to be
2607 clearly explained. A possible mitigation to this could be an interactive tutorial that
2608 helps developers fully understand the technical domain using a hands-on approach.
2609 For example, websites offer interactive Git tutorials⁷ to help developers understand
2610 and explore the technical domain matters under version control in their own pace.

2611 **Clarify limitations** API developers need to add clear limitations of the existing
2612 APIs. Limitations include list of objects that can be returned from an endpoint. We
2613 found that the cognitive anchors of how existing, conventional API documentation
2614 is written has become ‘ported’ to the computer vision realm, however a lot more
2615 overview documentation than what is given at present (i.e., better descriptions of
2616 errors, improved context of how these systems work in etc.) needs to be given. Such
2617 documentation could be provided using interactive tutorials.

⁷For example, <https://learngitbranching.js.org>.

2618 5.7 Threats to Validity**2619 5.7.1 Internal Validity**

2620 As detailed in Section 5.4.3.1, Taxonomies A and B present slight and fair agreement,
2621 respectively, when inter-rater reliability was applied. The nature of our disagree-
2622 ments largely fell due to the subjectivity in applying either taxonomies to posts.
2623 Despite all coders agreeing to the shared interpretation of both taxonomies, both
2624 taxonomies are subjective in their application, which was not reported by either
2625 Aghajani et al. or Beyer et al.. In many cases, multi-label classification seemed ap-
2626 propriate, however both taxonomies use single-label mapping which we find results
2627 in too much subjectivity. This subjectivity, therefore, ultimately adversely affects
2628 inter-rater reliability (IRR) analysis. Thus, a future mitigation strategy for similar
2629 work should explore multi-label classification to avoid this issue; Beyer et al., for
2630 example, plan for multi-label classification as future work. However, these studies
2631 would need to consider the statistical challenges in calculating multi-rater, multi-
2632 label IRR for thorough reliability analysis in addressing subjectivity. The selection
2633 of SO posts used for our labelling, chiefly in the subjectivity of our classifications, is
2634 of concern. We mitigate this by an extensive review process assessing the reliability
2635 of our results as per Section 5.4.3.1. The classification of our posts into the SOLO
2636 and Bloom’s taxonomies was performed by the primary author only, and therefore
2637 no inter-rater reliability statistics were performed. However, we used these peda-
2638 gogy related taxonomies as a lens to gain an additional perspective to interpret our
2639 results. Future studies should attempt a more rigorous analysis of SO posts using
2640 Bloom’s and SOLO taxonomies. We only aligned posts to one category for each
2641 taxonomy and did not align these using multi-label classification. This brings more
2642 complexity to the analysis, and our attempts to repeat prior studies’ methodologies
2643 (see Section 5.3). Multi-label classification for IWSs SO posts is an avenue for future
2644 research.

2645 5.7.2 External Validity

2646 While every effort was made to select posts from SO relevant to CVSs, there are
2647 some cases where we may have missed some posts. This is especially due to the
2648 case where some developers mis-reference certain IWSs under different names (see
2649 Section 5.4.2.1).

2650 Our SOLO and Bloom’s taxonomy analysis has only been investigated through
2651 the lenses of IWSs, and not in terms of conventional APIs (e.g., Andriod APIs).
2652 Therefore, we are not fully certain how these results found would compare to other
2653 types of APIs. Two *existing* SO classification taxonomies were used rather than
2654 developing our own. We wanted to see if previous SO taxonomies could be applied
2655 to IWSs before developing a new, specific taxonomy, and these taxonomies were
2656 applied based on our interpretation (see Section 5.4.2.4) and may not necessarily
2657 reflect the interpretation of the original authors. Moreover, automated techniques
2658 such as topic modelling were not utilised as we found these produce descriptive
2659 classifications only (see Section 5.3). Hence, manual analysis was performed by

humans to ensure categories could be aligned back to causal factors. Only English-speaking IWSs were selected; the applicability of our analysis to other, non-English speaking services may affect results. Use of computer vision in this study is an illustrative example to focus on one area of the IWSs spectrum. While our narrow scope helps us obtain more concrete findings, we suggest that wider issues exist in other IWS domains may affect the generalisability of this study, and suggest future work be explored in this space.

5.7.3 Construct Validity

Some questions extracted from SO produced false positives, as mentioned in Sections 5.4.2.1, 5.4.2.3 and 5.5. However, all non-relevant posts were marked as noise for our study, and thus did not affect our findings. Moreover, SO is known to have issues where developers simply ask basic questions without looking at the actual documentation where the answer exists. Such questions, although down-voted, were still included in our data-set analysis, but as these were SO few, it does not have a substantial impact on categorised posts.

5.8 Conclusions

CVSs offer powerful capabilities that can be added into the developer's toolkit via simple RESTful APIs. However, certain technical nuances of computer vision become abstracted away. We note that this abstraction comes at the expense of a full appreciation of the technical domain, context and proper usage of these systems. We applied two recent existing SO classification taxonomies (from 2018 and 2019) to see if existing taxonomies are able to fully categorise the types of complaints developers have. IWSs have a diverging distribution of the types of issues developers ask when compared to more mature domains (i.e., mobile app development and web development). Developers are more likely to complain about shallower, simple debugging issues without a distinct understanding of the AI algorithms that actually empower the APIs they use. Moreover, developers are more likely to complain about the completeness and correctness of existing IWS documentation, thereby suggesting that the documentation approach for these services should be reconsidered. Greater attention to education in the use of AI-powered APIs and their limitations is needed, and our discussion offered in Section 5.6.2 motivates future work in resolving these issues in the software engineering education space.

CHAPTER 6

2692

2693

2694

Ranking Computer Vision Service Issues using Emotion[†]

2695

2696 **Abstract** Software developers are increasingly using intelligent web services to implement
2697 ‘intelligent’ features. Studies show that incorporating AI into an application increases technical
2698 debt, creates data dependencies, and introduces uncertainty due to non-deterministic
2699 behaviour. However, we know very little about the emotional state of software developers
2700 who deal with such issues. In this paper, we do a landscape analysis of emotion found in
2701 1,425 Stack Overflow posts about computer vision services. We investigate the application
2702 of an existing emotion classifier EmoTxt and manually verify our results. We found that the
2703 emotion profile varies for different question categories.

2704 6.1 Introduction

2705 Recent advances in artificial intelligence have provided software engineers with
2706 new opportunities to incorporate complex machine learning capabilities, such as
2707 computer vision, through cloud-based intelligent web services (IWSs). These new
2708 set of services, typically offered as API calls are marketed as a way to reduce the
2709 complexity involved in integrating AI-components. However, recent work shows that
2710 software engineers struggle to use these IWSs [90]. Furthermore, the accompanying
2711 documentation fails to address common issues experienced by software engineers
2712 and, often, engineers resort to online communication channels, such as, JIRA and
2713 Stack Overflow (SO) to seek advice from their peers [90].

2714 While seeking advice on the issues, software engineers tend to express their emotions
2715 (such as frustration or confusion) within the questions. Recognising the value
2716 of considering emotions, other researchers have investigated emotions expressed by
2717 software developers within communication channels [264] including SO [68, 257];

[†]This chapter is originally based on M. K. Curumsing, A. Cummaudo, U. M. Graetsch, S. Barnett, and R. Vasa, “Ranking Computer Vision Service Issues using Emotion,” *arXiv preprint arXiv:2004.03120*, 2020. Terminology has been updated to fit this thesis.

2718 the broad motivation of these works is to generally understand the emotional land-
2719 scape and improve developer productivity [126, 245, 264]. However, previous works
2720 have not directly focused on the nature of emotions expressed in questions related to
2721 IWSs. We also do not know if certain types of questions express stronger emotions.

2722 The machine-learnt behaviour of these IWSs is typically non-deterministic and,
2723 given the dimensions of data used, their internal inference process is hard to reason
2724 about [87]. Compounding the issue, documentation of these cloud systems does not
2725 explain the limits, nor how they were created (esp. data sets used to train them).
2726 This lack of transparency makes it difficult for even senior developers to properly
2727 reason about these systems, so their prior experience and anchors do not offer
2728 sufficient support [90]. In addition, adding machine learned behaviour to a system
2729 incurs ongoing maintenance concerns [312]. There is a need to better understand
2730 emotions expressed by developers to inform cloud vendors and help them improve
2731 their documentation and error messages provided by their services.

2732 This work builds on top of recent work that explored *what* pain-points developers
2733 face when using IWSs through a general analysis of 1,425 SO posts (questions) [90]
2734 using an existing SO issue classification taxonomy [39]. In this work, we consider
2735 the emotional state expressed within these pain-points, using the same data set of
2736 1,425 SO posts. We identify the emotions in each SO question, and investigate if
2737 the distribution of these emotions is similar across the various types of questions.

2738 In order to classify emotions from SO posts, we use EmoTxt, a recently proposed
2739 toolkit for emotion recognition from text [67, 68, 257]. EmoTxt has been trained
2740 and built on SO posts using the emotion classification model proposed by Shaver
2741 et al. [318]. The category of issue was manually determined in our prior work.

2742 The key findings of our study are:

- 2743 • The distribution of emotions is different across the taxonomy of issues.
- 2744 • A deeper analysis of the results, obtained from the EmoTxt classifier, suggests
2745 that the classification model needs further refinement. Love and joy, the
2746 least expected emotions when discussing API issues, are visible across all
2747 categories.
- 2748 • A different emotion classification scheme is required to better reflect the
2749 emotions within the questions.

2750 In order to promote future research and permit replication, we make our data
2751 set publicly available.¹ The paper structure is as follows: Section 6.2 provides
2752 an overview on prior work surrounding the classification of emotions from text;
2753 Section 6.3 describes our research methodology; Section 6.4 presents the results
2754 from the EmoTxt classifier; Section 6.5 provides a discussion of the results obtained;
2755 Section 6.6 outlines the threats to validity; Section 6.7 presents the concluding
2756 remarks.

¹See <http://bit.ly/2RiULgW>.

2757 **6.2 Emotion Mining from Text**

2758 Several studies have investigated the role of emotions generally in software development [126, 264, 319, 375]. Work in the area of behavioural software engineering
2759 established the link between software developer's happiness and productivity [143].
2760 Wrobel [375] investigated the impact that software developers' emotion has on the
2761 development process and found that frustration and anger were amongst the emotions
2762 that posed the highest risk to developer's productivity.
2763

2764 Recent studies focused on emotion mining from text within communication chan-
2765 nels used by software engineers to communicate with their peers [126, 245, 257,
2766 264]. Murgia et al. [245] and Ortú et al. [264] investigated the emotions expressed
2767 by developers within an issue tracking system, such as JIRA, by labelling issue com-
2768 ments and sentences written by developers using Parrott's framework. Gachechiladze
2769 et al. [126] applied the Shaver framework to detect anger expressed in comments
2770 written by developers in JIRA. The Collab team [67, 257] extended the work done
2771 by Ortú et al. [264] and developed a gold standard data set collected from SO
2772 posts consisting of questions, comments and feedback. This data set was manually
2773 annotated using the Shaver's emotion model. The Shaver's model consists of a tree-
2774 structured, three level, hierarchical classification of emotions. The top level consists
2775 of six basic emotions namely, love, joy, anger, sadness, fear and surprise [318]. The
2776 subsequent levels further refines the granularity of the previous level. One of their
2777 recent work [257] involved 12 raters to manually annotate 4,800 posts (where each
2778 post included the question, answer and comments) from SO. The same question
2779 was assigned to three raters to reduce bias and subjectivity. Each coder was re-
2780 quested to indicate the presence/absence of each of the six basic emotions from the
2781 Shaver framework. As part of their work they developed an emotion mining toolkit,
2782 EmoTxt [67]. The work conducted by the Collab team is most relevant to our study
2783 since their focus is on identifying emotion from SO posts and their toolkit is trained
2784 on a large data set of SO posts.

2785 **6.3 Methodology**

2786 As mentioned in our introduction, this paper uses the data set reported in Cummaudo
2787 et al.'s ICSE 2020 paper [90]. As this paper is in press, we reproduce a summary
2788 of the methodology used in constructing this data set methodology below. For full
2789 details, we refer to the original paper. Supplementary materials used for this work
2790 are provided for replication.¹

2791 Our research methodology consisted of the following steps: (i) data extraction
2792 from SO resulting in 1,425 questions about intelligent computer vision services
2793 (CVSs); (ii) question classification using the taxonomy presented by Beyer et al. [39];
2794 (iii) automatic emotion classification using EmoTxt based on Shaver et al.'s emotion
2795 taxonomy [318]; and (iv) manual classification of 25 posts to better understand
2796 developers emotion. We calculated the inter-rater reliability between EmoTxt and
2797 our manually classified questions in two ways: (i) to see the overall agreement
2798 between the three raters in applying the Shaver et al. emotions taxonomy, and (ii) to

²⁷⁹⁹ see the overall agreement with EmoTxt’s classifications. Further details are provided
²⁸⁰⁰ below.

²⁸⁰¹ 6.3.1 Data Set Extraction from Stack Overflow

²⁸⁰² 6.3.1.1 Intelligent Service Selection

²⁸⁰³ We contextualise this work within popular CVS providers: Google Cloud [411],
²⁸⁰⁴ AWS [386], Azure [425] and IBM Cloud [421]. We chose these four providers given
²⁸⁰⁵ their prominence and ubiquity as cloud service vendors, especially in enterprise
²⁸⁰⁶ applications [294]. We acknowledge other services beyond the four analysed which
²⁸⁰⁷ provide similar capabilities [399, 400, 407, 420, 472, 473]. Additionally, only
²⁸⁰⁸ English-speaking services have been selected, excluding popular CVSs from Asia
²⁸⁰⁹ (e.g., [397, 398, 419, 438, 439]).

²⁸¹⁰ 6.3.1.2 Developing a search query

²⁸¹¹ To understand the various ways developers refer to these services, we needed to find
²⁸¹² search terms that are commonplace in question titles and bodies that discuss the
²⁸¹³ service names. One approach is to use the *Tags* feature in SO. To discover which
²⁸¹⁴ tags may be relevant, we ran a search² within SO against the various brand names of
²⁸¹⁵ these CVSs, reviewed the first three result pages, and recorded each tag assigned per
²⁸¹⁶ question.³ However, searching using tags alone on SO is ineffective (see [28, 340]).
²⁸¹⁷ To overcome this limitation, we ran a second query within the Stack Exchange Data
²⁸¹⁸ Explorer⁴ (SEDE) using these tags, we sampled 100 questions (per service), and
²⁸¹⁹ noted the permutations in how developers refer to each service⁵. We noted 229
²⁸²⁰ permutations.

²⁸²¹ 6.3.1.3 Executing our search query

²⁸²² Next, we needed to extract questions that make reference to any of these 229 per-
²⁸²³ mutations. SEDE has a 50,000 row limit and does not support case-insensitivity,
²⁸²⁴ however Google’s BigQuery does not. Therefore, we queried Google’s SO dataset
²⁸²⁵ on each of the 229 terms that may occur within the title or body of question posts,⁶
²⁸²⁶ which resulted in 21,226 questions.

²⁸²⁷ 6.3.1.4 Refining our inclusion/exclusion criteria

²⁸²⁸ To assess the suitability of these questions, we filtered the 50 most recent posts
²⁸²⁹ as sorted by their *CreationDate* values. This helped further refine the inclusion
²⁸³⁰ and exclusion criteria: for example, certain abbreviations in our search terms (e.g.,

²The query was run on January 2019.

³Up to five tags can be assigned per question.

⁴<http://data.stackexchange.com/stackoverflow>

⁵E.g., misspellings, misunderstanding of brand names, hyphenation, UK vs. US English, and varied uses of apostrophes, plurals, and abbreviations.

⁶See <http://bit.ly/2LrN70A>.

Table 6.1: Descriptions of dimensions from our interpretation of Beyer et al.’s SO question type taxonomy.

Dimension	Our Interpretation
API usage	Issue on how to implement something using a specific component provided by the API
Discrepancy	The questioner’s <i>expected behaviour</i> of the API does not reflect the API’s <i>actual behaviour</i>
Errors.....	Issue regarding an error when using the API, and provides an exception and/or stack trace to help understand why it is occurring
Review	The questioner is seeking insight from the developer community on what the best practices are using a specific API or decisions they should make given their specific situation
Conceptual.....	The questioner is trying to ascertain limitations of the API and its behaviour and rectify issues in their conceptual understanding on the background of the API’s functionality
API change.....	Issue regarding changes in the API from a previous version
Learning	The questioner is seeking for learning resources to self-learn further functionality in the API, and unlike discrepancy, there is no specific problem they are seeking a solution for

²⁸³¹ ‘GCV’, ‘WCS’⁷) allowed for false positive questions to be included, which were
²⁸³² removed. Furthermore, we consolidated all overlapping terms (e.g., ‘Google Vision
²⁸³³ API’ was collapsed into ‘Google Vision’) to enhance the query. Additionally, we
²⁸³⁴ reduced our 221 search terms to just 27 search terms by focusing on CVSs *only*⁸
²⁸³⁵ which resulted in 1,425 questions. No duplicates were recorded as determined by
²⁸³⁶ the unique ID, title and timestamp of each question.

²⁸³⁷ 6.3.1.5 *Manual filtering*

²⁸³⁸ The next step was to assess the suitability and nature of the 1,425 questions extracted.
²⁸³⁹ The second author ran a manual check on a random sample of 50 posts, which were
²⁸⁴⁰ parsed through a templating engine script⁹ in which the ID, title, body, tags, created
²⁸⁴¹ date, and view, answer and comment counts were rendered for each post. Any match
²⁸⁴² against the 27 search terms in the title or body of the post were highlighted, in which
²⁸⁴³ three false positives were identified as either library imports or stack traces, such
²⁸⁴⁴ as `aws-java-sdk-rekognition.jar`. In addition, we noted that there were false
²⁸⁴⁵ positive hits related to non-CVSs. We flagged posts of such nature as ‘noise’ and
²⁸⁴⁶ removed them from further classification.

⁷Watson Cognitive Services

⁸Our original data set aimed at extracting posts relevant to *all* IWSs, and not just CVSs. However, 21,226 questions were too many to assess without automated analysis, which was beyond the scope of our work.

⁹We make this available for future use at: <http://bit.ly/2NqBB70>.

2847 6.3.2 Question Type & Emotion Classification

2848 6.3.2.1 Manual classification of question category

2849 We classify our 1,425 posts using Beyer et al.'s taxonomy [39] as it was comprehensive and validated [90]. We split the posts into 4 additional random samples, in
 2850 addition to the random sample of 50 above. 475 posts were classified by the second
 2851 author and three other research assistants¹⁰ classified the remaining 900 (i.e., a total
 2852 of 1,375 classifications). An additional 450 classifications were assigned due to
 2853 reliability analysis, in which the remaining 50 posts were classified nine times by
 2854 various researchers in our group.¹¹

2855 Due to the nature of reliability analysis, multiple classifications (450) existed
 2856 for these 50 posts. Therefore, we applied a 'majority rule' technique to each post
 2857 allowing for a single classification assignment and therefore analysis within our re-
 2858 sults. When there was a majority then we used the majority classification; when
 2859 there was a tie, then we used the classification that was assigned the most out of the
 2860 entire 450 classifications. As an example, 3 raters classified a post as *API Usage*,
 2861 1 rater classified the same post as a *Review* question and 5 raters classified the post
 2862 as *Conceptual*, resulting in the post being classified as a *Conceptual* question. For
 2863 another post, three raters assigned *API Usage*, *Discrepancy* and *Learning* (respec-
 2864 tively), while 3 raters assigned *Review* and 3 raters assigned *Conceptual*. In this
 2865 case, *Review* and *Conceptual* were tied, but was resolved down to *Conceptual* as this
 2866 classification received 147 more votes than *Review* across all classifications made in
 2867 the sample of 50 posts.

2868 However, where a post was extracted from our original 1,425 posts but was either
 2869 a false positive, not applicable to IWSs (see Section 6.3.1.5), or not applicable to
 2870 a taxonomy dimension/category, then the post was flagged for removal in further
 2871 analysis. This was done 180 times, leaving a total of 1,245 posts.

2872 Our interpretation Beyer et al.'s taxonomy is provided in Table 6.1, which
 2873 presents a transcription of *our understanding* of the respective taxonomy. We
 2874 baselined all coding against *our interpretation only*, and thus our classifications
 2875 are therefore independent of Beyer et al.'s findings, since we baseline results via
 2876 Table 6.1's interpretation.

2878 6.3.2.2 Emotion classification using AI techniques

2879 After extracting and classifying all posts, we then piped in the body of each question
 2880 into a script developed to remove all HTML tags, code snippets, blockquotes and
 2881 hyperlinks, as suggested by Novielli et al. [257]. We replicated and extended the
 2882 study conducted by Novielli et al. [257] on our data set derived from 1,425 SO posts,
 2883 consisting of questions only. Our study consisted of three main steps, namely, (1)
 2884 automatic emotion classification using EmoTxt, (2) manual annotation process and,
 2885 (3) comparison of the automatic classification result with the manually annotated
 2886 data set.

¹⁰Software engineers in our research group with at least 2 years industry experience

¹¹Due to space limitations, reliability analysis is omitted and is reported in [90].

2887 6.3.2.3 *Emotion classification using EmoTxt*

2888 We started with a file containing 1,245 non-noise SO questions, each with an as-
2889 sociated question type as classified using the strategy discussed in Section 6.3.2.1.
2890 We pre-processed this file by extracting the question ID and body text to meet the
2891 format requirements of the EmoTxt classifier [67]. This classifier was used as it
2892 was trained on SO posts as discussed in Section 6.2. We ran the classifier for each
2893 emotion as this was required by EmoTxt model. This resulted in 6 output prediction
2894 files (one file for each emotion: *Love*, *Joy*, *Surprise*, *Sadness*, *Fear*, *Anger*). Each
2895 question within these files referenced the question ID and a predicted classification
2896 (YES or NO) of the emotion. We then merged the emotion prediction files into an
2897 aggregate file with question text and Beyer et al.’s taxonomy classifications. This
2898 resulted in 796 emotion classifications. We further analysed the classifications and
2899 generated an additional classification of *No Emotion* for the 622 questions where
2900 EmoTxt predicted NO for all the emotion classification runs.

2901 Of the 796 questions with emotion detected, 143 questions had 2 or more
2902 emotions predicted: 1 question¹² had up to 4 emotions detected (*Surprise*, *Sadness*,
2903 *Joy* and *Fear*), 28 questions had up to 3 emotions detected, and the remaining 114
2904 had up to two emotions detected.

2905 6.3.2.4 *Manual Annotation Process*

2906 In order to evaluate and also better understand the process used by EmoTxt to
2907 classify emotions, we manually annotated a small sample of 25 SO posts, randomly
2908 selected from our data set. Each of these 25 posts were assigned to three raters who
2909 carried out the following three steps: (i) identify the presence of an emotion; (ii)
2910 if an emotion(s) exists, classify the emotion(s) under one of the six basic emotions
2911 proposed by the Shaver framework [318]; (iii) if no emotion is identified, annotate as
2912 neutral. We then collated all rater’s results and calculated Light’s Kappa (L_K) [211]
2913 to measure the overall agreement *between* raters to measure the similarity in which
2914 independent raters classify emotions to SO posts. As L_K does not support multi-class
2915 classification (i.e., multiple emotions) per subjects (i.e., per SO post), we binarised
2916 the results each emotion and rater as TRUE or FALSE to indicate presence, calculated
2917 the L_K per emotion against the three raters, and averaged the result across all emotions
2918 to get an overall strength of agreement.

2919 6.3.2.5 *Comparing EmoTxt results with the results from Manual Classification*

2920 The next step involved comparing the ratings of the 25 SO posts that were manually
2921 annotated by the three raters with the results obtained for the same set of 25 SO
2922 posts from the EmoTxt classifier. Similar to Section 6.3.2.4, we used Cohen’s Kappa
2923 (C_K) [81] to measure the consistency of classifications of EmoTxt’s classifications
2924 versus the manual classifications of each rater. We separated the classifications
2925 per emotion and calculated C_K for each rater against EmoTxt and averaged these
values for all emotions. After noticing poor results, the three raters involved in

¹²See <http://stackoverflow.com/q/55464541>.

²⁹²⁷ Section 6.3.2.4 were asked to compare and discuss the ratings from the EmoTxt
²⁹²⁸ classifier against the manual ratings.

²⁹²⁹ The findings from this process are presented and discussed in the next two
²⁹³⁰ sections.

²⁹³¹ 6.4 Findings

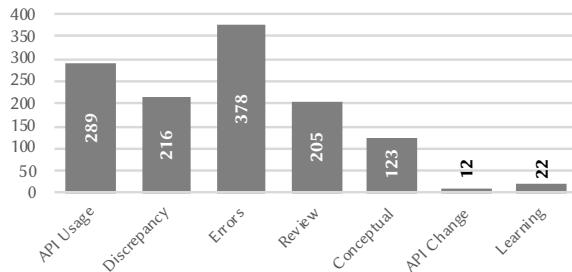


Figure 6.1: Distribution of SO question types.

²⁹³² Figure 6.1 displays the overall distribution of question types from the 1,245
²⁹³³ posts classified in [90], when adjusted for majority ruling as per Section 6.3.2.1. It
²⁹³⁴ is evident that developers ask issues predominantly related to API errors when using
²⁹³⁵ CVSs and, additionally, how they can use the API to implement specific functionality.
²⁹³⁶ There are few questions related to version issues or self-learning.

Table 6.2: Frequency of emotions per question type.

Question Type	Fear	Joy	Love	Sadness	Surprise	Anger	No Emotion	Total
API Usage	50	22	34	18	59	13	135	331
Discrepancy	38	12	18	7	48	20	108	251
Errors	69	34	22	21	48	23	206	423
Review	34	16	15	16	42	14	98	235
Conceptual	26	10	10	7	21	5	59	138
API Change	4	2	2	1	1	1	5	16
Learning	3	4	2	0	4	0	11	24
Total	224	100	103	70	223	76	622	1418

²⁹³⁷ Table 6.2 displays the frequency of questions that were classified by EmoTxt
²⁹³⁸ when compared to our assignment of question types, while Figure 6.2 presents the
²⁹³⁹ emotion data proportionally across each type of question. *No Emotion* was the
²⁹⁴⁰ most prevalent across all question types, which is consistent with the findings of the
²⁹⁴¹ Collab group during the training of the EmoTxt classifier. Interestingly, *API Change*
²⁹⁴² questions had a distinct distribution of emotions, where 31.25% of questions had *No*
²⁹⁴³ *Emotion* compared to the average of 42.01%. This is likely due to the low sample
²⁹⁴⁴ size of *API Change* questions, with only 12 assignments, however the next highest
²⁹⁴⁵ set of emotive questions are found in the second largest sample (*API Usage*, at
²⁹⁴⁶ 59.21%) and so greater emotion detected is not necessarily proportional to sample

size. Unsurprisingly, *Discrepancy* questions had the highest proportion of the *Anger* emotion, at 7.97%, compared to the mean of 4.74%, which is indicative of the frustrations developers face when the API does something unexpected. *Love*, an emotion which we expected least by software developers when encountering issues, was present across the different question types. The two highest emotions, by average, were *Fear* (16.67%) and *Surprise* (14.90%), while the two lowest emotions were *Sadness* (4.47%) and *Anger* (4.74%). *Joy* and *Love* were roughly the same and fell in between the two proportion ends, with means of 8.96% and 8.16%, respectively.

Results from our reliability analysis showed largely poor results. Guidelines of indicative strengths of agreement are provided by Landis and Koch [206], where $\kappa \leq 0.000$ is *poor agreement*, $0.000 < \kappa \leq 0.200$ is *slight agreement* and $0.200 < \kappa \leq 0.400$ is *fair agreement*. Our readings were indicative of poor agreement between raters ($C_\kappa = -0.003$) and slight agreement with EmoTxt ($L_\kappa = 0.155$). The strongest agreements found were for *No Emotion* both between each of our three raters ($L_\kappa = 0.292$) and each rater and EmoTxt ($C_\kappa = 0.086$), with fair and slight agreement respectively.

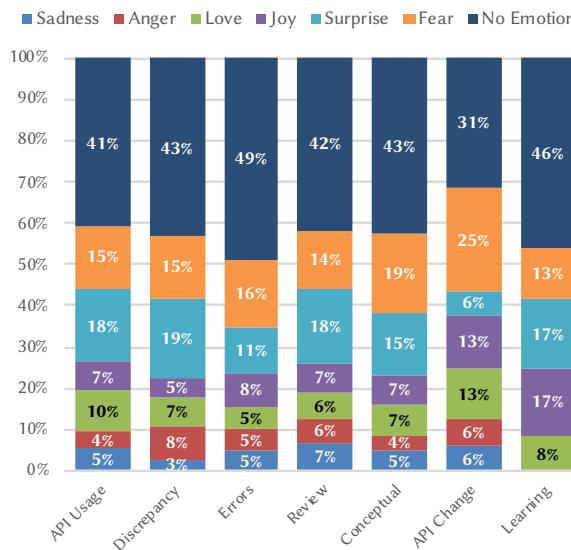


Figure 6.2: Proportion of emotions per question type.

6.5 Discussion

Our findings from the comparison between the manually annotated SO posts and the automatic classification revealed substantial discrepancies. Table 6.3 provide some sample questions from our data set and the emotion identified by EmoTxt within the text. A subset of questions analysed by our three raters do not indicate the automatic (EmoTxt) emotion, and upon manual inspection of the text after poor

Table 6.3: Sample questions comparing question type to emotion. Questions located at [https://stackoverflow.com/q/\[ID\]](https://stackoverflow.com/q/[ID]).

ID	Quote	Classification	Emotion
53249139	<i>"I'm trying to integrate my project with Google Vision API... I'm wondering if there is a way to set the credentials explicitly in code as that is more convenient than setting environment variables in each and every environment we are running our project on... I know for a former client version 1.22 that was possible... but for the new client API I was not able to find the way and documentation doesn't say anything in that regards."</i>	API Usage	Fear
40013910	<i>"I want to say something more about Google Vision API Text Detection, maybe any Google Expert here and can read this. As Google announced, their TEXT_DETECTION was fantastic... But for some of my pics, what happened was really funny... There must be something wrong with the text detection algorithm."</i>	Discrepancy	Anger
50500341	<i>"I just started using PYTHON and now i want to run a google vision cloud app on the server but I'm not sure how to start. Any help would be greatly appreciated."</i>	API Usage	Sadness
49466041	<i>"I am getting the following error when trying to access my s3 bucket... my hunch is it has something to do with the region...I have given almost all the permissions to the user I can think of.... Also the region for the s3 bucket appears to be in a place that can work with rekognition. What can I do?"</i>	Errors	Surprise
55113529	<i>"Following a tutorial, doing everything exactly as in the video... Hoping to figure this out as it is a very interesting concept...Thanks for the help... I'm getting this error:..."</i>	Errors	Joy
39797164	<i>"Seems that the Google Vision API has moved on and the open Sourced version has not....In my experiments this 'finds' barcodes much faster than using the processor that the examples show. Am I missing something somewhere?"</i>	API Change	Love

2970 results from our reliability analysis, an introspection of the data set sheds some light
2971 to the discrepancy. For example, question 55113529 shows no indication of *Joy*,
2972 rather the developer is expressing a state of confusion. The phrase “*Thanks for your*
2973 *help*” could be the reason why the miss-classification occurred if words like “thanks”
2974 were associated with joy. However, in this case, it seems unlikely that the developer
2975 is expressing joy as the developer has followed a tutorial but is still encountering
2976 an error. Similarly, question 39797164, classified as *Love* and question 50500341,
2977 classified as *Sadness* express a state of confusion and the urge to know more about the
2978 product; upon inspecting the entire question in context, it is difficult to consistently
2979 agree with the emotions as determined by EmoTxt, and further exploration into the
2980 behaviour and limitations of the model is necessary.

2981 Our results indicate further work is needed to refine the machine learning (ML)
2982 classifiers that mine emotions in the SO context. The question that arises is whether
2983 the classification model is truly reflective of real-world emotions expressed by soft-
2984 ware developers. As highlighted by Curumsing [92], the divergence of opinions with
2985 regards to the emotion classification model proposed by theorists raises doubts to the
2986 foundations of basic emotions. Most of the studies conducted in the area of emotion
2987 mining from text is based on an existing general purpose emotion framework from
2988 psychology [63, 257, 264]—none of which are tuned for software engineering do-
2989 main. In our study, we note the emotions expressed by software developers within
2990 SO posts are quite narrow and specific. In particular, emotions such as frustration
2991 and confusion would be more appropriate over love and joy.

2992 **6.6 Threats to Validity**

2993 **6.6.1 Internal Validity**

2994 The *API Change* and *Learning* question types were few in sample size (only 12 and
2995 22 questions, respectively). The emotion proportion distribution of these question
2996 types are quite different to the others. Given the low number of questions, the sample
2997 is too small to make confident assessments. Furthermore, our assignment of Beyer
2998 et al.’s question type taxonomy was single-label; a multi-labelled approach may work
2999 better, however analysis of results would become more complex. A multi-labelled
3000 approach would be indicative for future work.

3001 **6.6.2 External Validity**

3002 EmoTxt was trained on questions, answers and comments, however our data set
3003 contained questions only. It is likely that our results may differ if we included other
3004 discussion items, however we wished to understand the emotion within developers’
3005 *questions* and classify the question based on the question classification framework
3006 by Beyer et al. [39]. Moreover, this study has only assessed frustrations within the
3007 context of a concrete domain of CVSs. The generalisability of this study to other
3008 IWSs, such as natural language processing services, or conventional web services,
3009 may be different. Furthermore, we only assessed four popular CVSs; expanding the

3010 data set to include more services, including non-English ones, would be insightful.
3011 We leave this to future work.

3012 **6.6.3 Construct Validity**

3013 Some posts extracted from SO were false positives. Whilst flagged for removal
3014 (Section 6.3.1.5), we cannot guarantee that all false positives were removed. Fur-
3015 thermore, SO is known to have questions that are either poorly worded or poorly
3016 detailed, and developers sometimes ask questions without doing any preliminary
3017 investigation. This often results in down-voted questions. We did not remove such
3018 questions from our data set, which may influence the measurement of our results.

3019 **6.7 Conclusions**

3020 In this paper we analysed SO posts for emotions using an automated tool and cross-
3021 checked it manually. We found that the distribution of emotion differs across the
3022 taxonomy of issues, and that the current emotion model typically used in recent
3023 works is not appropriate for emotions expressed within SO questions. Consistent
3024 with prior work [213], our results demonstrate that machine learning classifiers for
3025 emotion are insufficient; human assessment is required.

3026

3027

3028

3029

Lessons learnt using a pre-trained AI model[†]

3030 **Abstract** Pre-trained AI models are increasingly available as APIs and tool-kits to soft-
3031 ware engineers, making complex AI-enabled functionality available via standard and well-
3032 understood methods. However, reusing such models comes with risks relating to the lack of
3033 transparency of the model and training data bias, making it difficult to confidently employ
3034 the toolkit in a new situation. Vendors are responding and proposing artefacts such as
3035 model cards and datasheets to make models and their training more transparent. But is this
3036 enough? As part of an investigation into determining if a cloud-based intelligent web service
3037 was ready for production use, we processed developer questions on Stack Overflow using
3038 a published pre-trained classifier that was specifically tuned for the software engineering
3039 domain. In this paper, we present lessons learnt in this automation effort. We find the results
3040 were unexpected and led us to delve into model and training data—an option available to
3041 us because the information was available for research. We found that had a model card and
3042 datasheet been prepared, we could have identified risks to our study earlier on. However,
3043 model cards and datasheets specifications are not yet mature enough and additional tools
3044 and processes are still required to confirm a decision whether a model can be reused with
3045 confidence.

3046

7.1 Introduction

3047 Pre-trained AI models are increasingly available to software engineers either directly
3048 or wrapped into web-based components and toolkits.¹ The grand promise is the
3049 rapid creation of AI-infused functionality into end-applications as developers can
3050 simply reuse models instead of training them from scratch, as training is laborious
3051 and resource-intensive [287]. Vendors do provide usage guidelines, component

[†]This chapter is originally based on U. M. Graetsch, A. Cummaudo, R. Vasa, and M. K. Curumsing, “Lessons learnt using a pre-trained AI model,” 2020, Unpublished. Terminology has been updated to fit this thesis.

¹For example, Google’s Cloud AI or Microsoft Azure’s Cognitive Services.

3052 documentation, code examples and a compelling marketing narrative, although the
3053 limitations and risks are not as well-presented in official documentation [87, 90]. In
3054 practice, developers and technical architects study issue trackers and online forums
3055 such as Stack Overflow (SO) to assess and inform their decisions. This is also
3056 complemented by multiple studies that highlight the value and insights to be gained
3057 from these online forums [2, 330].

3058 This paper is the result of an investigation into determining if cloud-based
3059 intelligent web services (IWSs) are ready for a specific industry use case. Inspired
3060 by the possibility of finding insight from content in the online forums, we wanted to
3061 analyse the questions posed and issues raised—in particular on SO—that relate to
3062 that relate to IWSs that provide computer vision (i.e., computer vision services or
3063 CVSS). Although a manual analysis is possible, we wanted to automate this process
3064 using natural language processing techniques, which was motivated by (i) the gain
3065 from automation—specifically having a repeatable process that can be tuned to focus
3066 on different aspects—and, more importantly, (ii) to learn about potential issues with
3067 these pre-trained models as we use one of these services to turn on themselves.

3068 In our analysis, beyond the direct summative aspects, we focused on emotions
3069 within the content posed on the online forums. This was motivated by work done
3070 by Wrobel [375], who suggested that frustration and anger were amongst the emotions
3071 that posed the highest risk to developer productivity. Our goal was to determine
3072 if negative emotions such as anger or frustration are the predominant theme within
3073 questions on these forums: the natural expectation is that developers would not pose
3074 questions unless they needed support and help. Similarly, we would expect the tone
3075 of responses to be neutral and hopefully supportive. Our focus, however, remains
3076 on the questions posed.

3077 Our findings, elaborated further in Section 7.2.3, were surprising. While the
3078 pre-trained model we selected was trained specifically on SO and tuned for emotions
3079 [67, 257], our results show that 14% issues can be considered in the category
3080 of *Love* or *Joy*, and a surprising small amount (5%) are in the category of *Anger* (or
3081 frustration). A closer examination using multiple human reviewers showed an even
3082 more interesting insight: the reviewers did not agree with the automated machine
3083 classification, and worse, the reviewers did not agree with each other, suggesting that
3084 training machines with a consistent set of labels is a non-trivial exercise. Finally,
3085 we reflected whether the pre-trained classifier could be better documented. We
3086 found vendors are recognising these challenges and are offering solutions to better
3087 document their models [132, 243]. However, when we looked into the information
3088 captured by these solutions, we found their specification to be very broad and addi-
3089 tional guidance for completion is required to help evaluate risks faced in an industry
3090 context (discussed in Section 7.3.3).

3091 7.2 Case Study

3092 In this section, we discuss the case study which inspired the initial objective of
3093 investigating if cloud-based IWSs were ready for use in an industrial context. To
3094 permit replication, the raw results produced from this case study are made available

3095 online at <https://bit.ly/36DIARI>.

3096 7.2.1 Scope

3097 To align with our use case, we narrowed our focus to cloud-based computer vision
3098 services (CVSs). Recent research has identified growth in questions on SO relating
3099 to such services, giving us confidence that we would have a rich data set [90]. We
3100 decided to explore emotions expressed by developers through the questions they
3101 pose on SO to identify whether developers are surprised, angry, frustrated, or overall
3102 positive? Previous works into developer questions show that despite the technical
3103 nature, their communications on SO do exhibit emotions [67, 256]. Although we
3104 could have read these posts manually, for consistency, repeatability, and efficiency,
3105 we chose to automate this process by utilising an emotion aware text classification
3106 system trained specifically on SO [257]. Our expectation was that we would gain
3107 some insight into the questions through the emotions, and we hypothesised that we
3108 would see a high proportion of surprise (i.e., the API does not work as expected)
3109 and anger (frustration due to mismatched expectations).

3110 7.2.2 Method

3111 We selected a published peer-reviewed emotion model as the text classifier. This
3112 classifier is included in the EMTk toolkit and has been specifically trained for emotional
3113 text classification in the software engineering domain [67]. The EMTk toolkit is
3114 available with a fully labelled training dataset [257], permitting reuse and analysis of
3115 internals. The classifier is based on Shaver et al.'s emotional hierarchy model [318]
3116 and performs binary classifications against text data provided in input files and an
3117 input parameter designating the emotion to be classified—one of *Love, Joy, Surprise,*
3118 *Fear, Sadness or Anger*. As input for the classifier, we used a dataset of the 1,425
3119 SO questions restricted to intelligent CVSs available in [90] and we ran the classifier
3120 with the same input dataset for each of the six emotions. To cross-check classified
3121 output, we manually annotated a random sample of 25 questions. Each of these 25
3122 posts were assigned to three raters who carried out the following three steps: (i)
3123 identify the presence of emotion(s); (ii) if emotion(s) exists, classify the emotion(s)
3124 under one of the six basic emotions as per the Shaver framework. After collating
3125 each rater's results, we calculated a Fleiss' Kappa [118] as a measure of inter-rater
3126 agreement per emotion for each of the three human raters (manual rating). We
3127 then used the results from the classifier as a 'fourth' *automated* rater, comparing
3128 the results with the manual rating by calculating the agreement for each emotion
3129 and calculated the observed percentage of agreement and Fleiss' Kappa for further
3130 inter-rater agreement analysis.

3131 7.2.3 Results

3132 Of the 1,425 SO questions, the classifier did not classify any emotion to 622 posts
3133 (labelled *No Emotion*). The remaining posts were classified: 224 posts as *Fear*, 223
3134 as *Surprise*, 70 as *Sadness*, 103 as *Love*, 100 as *Joy*, and 76 as *Anger*. Some posts

Table 7.1: Results from Inter-Rater Agreements.

Emotion	Three Raters	Three Raters + Classifier
Anger	0.256	0.145
Fear	-0.014	-0.075
Joy	0.306	0.132
Love	1.000	-0.031
Sadness	-0.071	-0.053
Surprise	-0.056	-0.091
No Emotion	0.265	0.139

3135 classified against two or more emotions, and as a result, the total proportions do
3136 not add up to exactly 100%. Results from our inter-rater analysis are reported in
3137 Table 7.1.

3138 Guidelines of indicative strengths of agreement are provided by Landis and
3139 Koch [206], where: $\kappa \leq 0$ indicates *poor* agreement; $0 < \kappa \leq 0.2$ indicates *slight*
3140 agreement; $0.2 < \kappa \leq 0.4$ indicates *fair* agreement; $0.4 < \kappa \leq 0.6$ indicates
3141 *moderate* agreement; $0.6 < \kappa \leq 0.8$ indicates *substantial* agreement. When using
3142 the classifier's output as a fourth 'rater', there was slight agreement on *Anger*, *Joy*,
3143 and *No Emotion*. Between the three human raters, those same emotions were rated
3144 with fair agreement. Agreement for *Love* was unanimous amongst the three human
3145 raters, finding zero instances of *Love* across the sample (thus resulting in a kappa
3146 value of 1.00). Inter-rater agreement was poor for the *Fear*, *Sadness*, and *Surprise*.

3147 7.3 Findings and Discussion

3148 In this section, we reflect on our results with respect to limitations in the classifier
3149 and investigation of the peer-reviewed and published dataset used to train the clas-
3150 sifier. Given the weak results, we also investigate whether model cards [243] and/or
3151 datasheets [132] could have provided a more effective approach to informing the
3152 viability and limits of the pre-trained model.

3153 7.3.1 Limitations of the Text Classifier

3154 The classifier did not assign any emotion to 43% posts. This result corroborates the
3155 findings by Murgia et al., who identified via a manual process *No Emotion* as the
3156 most prevalent classification [245]. For illustration, we provide a set of examples
3157 in ???. (The numbers in the raters column indicate the count of raters who assigned
3158 this label.) In the first example, each human rater assigned a different emotion to
3159 the same question, despite the classifier classifying it with *No Emotion*. The second
3160 example also highlights discrepancy between raters (two raters agree on *Joy*) but the
3161 classifier classified it as *Love*. Lastly, the third example highlights one example of
3162 consistency between all three raters and the classifier.

Table 7.2: Sample questions comparing EmoTxt to manual classification. Questions located at: [https://stackoverflow.com/q/\[ID\]](https://stackoverflow.com/q/[ID]).

Question ID	Quote	EmoTxt	Raters
54521080	<p><i>"I am aware that it is better to use AWS Rekognition for this. However, it does not seem to work well when I tried it out with the images I have (which are sort of like small containers with labels on them). The text comes out misspelled and fragmented. I am new to ML and Sagemaker ... Is it possible to do it with Sagemaker? I would appreciate it if someone pointed me in the right direction."</i></p>	No Emotion	Fear (1) Sadness (1) Surprise (1)
52446033	<p><i>"I am using the Google Cloud Vision API to search similar images (web detection) and it works pretty well. Google detects full matching images and partial matching images (cropped versions). I am looking for a way to detect more different versions. For example, when I look for a logo, I would like to detect large, small, square, rectangular versions of this logo. For now, I detect images that match exactly the one I upload and cropped versions. Do you know if this is possible and how can I do that?"</i></p>	Love	Joy (2) No Emotion (1)
54677464	<p><i>"I have implemented a QR scanner(QRScanner class) using Google Vision API. Once a value is detected it is passed to another activity(Info class) using Intents. The problem is that once a QR code is scanned the Info class gets opened several times.I want to limit the QRScanner class to get only one QR value and Info classed to be opened only once. Currently once a QR is detected the Info class gets called several times. I want the QRScanner to get only one value and Info class to get called only once."</i></p>	No Emotion	No Emotion (3)

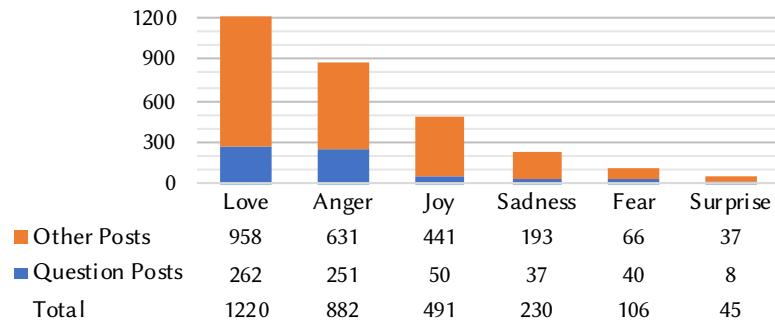


Figure 7.1: The emotion classifier training dataset distribution is largely skewed toward *Love*, resulting in data imbalance. (*No Emotion* labels were removed from this graph.)

7.3.2 Peering into the Training Dataset

We investigated the training dataset and related research documentation to see if that would give us further insights. We found two areas warranting further exploration—training data balance and training data annotation.

7.3.2.1 Data imbalance

We found that the purpose of the training dataset was actually to train two classifiers—a sentiment classifier and an emotional classifier. Each post in the training dataset was labelled with zero, one or more emotions. In addition, emotions were grouped, i.e., the positive emotions of *Joy* and *Love* were grouped into positive sentiment while *Sadness*, *Anger* and *Fear* were grouped into negative sentiment. *Surprise* was assigned either positive or negative sentiment, depending on context [68, 257]. Figure 7.1 shows the distribution of emotion labels across 4,800 posts in the training dataset; *No Emotion* is removed to emphasise emotion-only results. (Total count: 1959.)

Class imbalance and its impact on classifier models is a known problem in machine learning [220, 365], where one class (known as the majority, positive class) significantly outnumbers the other class (known as the minority, negative class). The impact of class imbalance on classification models results in minority classes with lower precision and lower recall than the majority class, since the classifier does not generate rules for the minority class. One set of relevant techniques for addressing class imbalance is data sampling; including undersampling or subsampling, oversampling and hybrid approaches [220]. Whilst the training dataset seems balanced for the purpose of sentiment analysis, there is a lack of balance across individual emotions. The predominant emotion in the training dataset was *No Emotion* at 40.8% of total posts. The most dominant emotions were *Love* and *Anger* at 25% and 18% respectively. Less than 1% of posts were labelled with *Surprise*. This means that the number of posts falling into some of the categories, for example, *Surprise* and *Fear* (i.e., 45 and 106 posts, respectively) is very low for training purposes.

Further, the training dataset was spread across different types of SO posts (i.e., questions posts, answer posts, question comments, answer comments) to capture the

3193 different emotional language, however our study was interested in classifying SO
3194 question posts only. Of the training dataset's 4,800 posts, only 1,044 were question
3195 posts and within that subset of posts, the distribution of emotions was more polarised
3196 than in the overall 4,800 posts. *Love* and *Anger* are the most predominant emotions
3197 in the training dataset, however *Anger* has a higher proportion (24%) in question
3198 posts, as opposed to only 18.4% in the overall dataset.

3199 In summary, the training dataset was not balanced within each emotion category
3200 and some emotions had very low sample numbers, as emphasised in the skew in
3201 Figure 7.1. Proportions of training data examples per question per emotion was very
3202 low for *Joy*, *Surprise*, *Sadness* and *Fear*. To address this imbalance and achieve
3203 better performance, training data could be enhanced to include additional samples
3204 or to use an oversampling approach. A recent study into class-balancing approaches
3205 in the context of defect prediction models found that class rebalancing does lead to
3206 a shift in the learned concepts [348].

3207 7.3.2.2 *Emotion labeling bias*

3208 In software engineering, hierarchical categorical emotional frameworks including
3209 those featured in Parrott [269], Ekman et al. [110] and Shaver et al. [318] have
3210 been assessed by researchers and pragmatically selected as the basis for training
3211 emotional classifiers. The chosen emotion framework is then used as the taxonomy
3212 truth labels for classifier training datasets. Data for labeling is sourced from systems
3213 such as SO and JIRA [126, 245, 257, 264]. In the software engineering domain,
3214 truth labeling of emotions has to date been done manually [126, 245, 257]. Emotion
3215 annotation involves at least a pair of annotators [11, 136]. For the EMTk training
3216 dataset, annotation was performed manually by a team of 12 coders, divided into four
3217 groups of three with a computer science background [67, 257]. Manual annotation
3218 challenges when coding emotions can be encountered due to different levels of
3219 semantic ambiguity within emotions and how humans express emotions in text [150].
3220 In the absence of an objective emotional truth, researchers' consistency is taken as
3221 a measure of correctness—i.e., multiple annotators that agree [245]. A measure of
3222 inter-rater agreement is Cohen's Kappa [81] (for two raters) or Fleiss' Kappa [118]
3223 for more raters. For the training data set, inter-rater agreement ranged from $\kappa = 0.30$
3224 (fair) for *Joy* to $\kappa = 0.66$ (substantial) for *Love*. The researchers specifically trained
3225 dataset coders for consistency. The challenge of this approach with a subject such
3226 as emotions is the opportunity for bias. In contrast, in other studies, researchers
3227 specifically attempted to reduce the opportunity for biases by including raters with
3228 different nationalities, skills, cultural backgrounds, by increasing the number of
3229 raters [264] and opting against consistency training [9]. As such, the approach
3230 taken to achieve consistency and makeup of label coders is important information
3231 for downstream consumers of an AI model.

3232 7.3.2.3 *Emotion labeling and classification granularity*

3233 Training data annotation was performed on SO posts—which included questions,
3234 answers, and comments to questions and answers. Emotion annotation can be

3235 performed at different levels of granularity—word level [331], spans of words in a
3236 sentence [11], sentence level or larger. While a word level or keyword approach is
3237 considered too granular (as it does not capture the emotional context sufficiently),
3238 there is a risk of emotion progression during narratives and also within sentences [11,
3239 245]. Our CVS dataset consisted of questions only as we were seeking to assess
3240 developer emotion expressed at the time of raising the question. Question posts are
3241 typically longer than comments and may contain multiple emotions expressed at
3242 different levels of intensity that are interpreted differently by different readers. For
3243 example, in the first question in ?? the first sentence does not carry any emotion
3244 while in the second part the reader expressed *Surprise* that the API does not work
3245 in all cases (*Surprise/Sadness*) and their lack of experience (*Fear*), and appreciation
3246 (possibly *Love*).

3247 7.3.3 New tools: Model Cards and Datasheets

3248 Model cards are emerging tools proposed by Google to communicate performance
3249 characteristics of pre-trained models [243]. Google have recently published sam-
3250 ple model cards relating to their Cloud Vision API.² Microsoft have focused on
3251 a standardised process of dataset documentation through datasheets to encourage
3252 transparency and accountability by documenting the motivation, composition, col-
3253 lection process and intended uses of data [132]. IBM have proposed a FactSheet
3254 concept combining model and data information [14]. These tools are being adopted
3255 by organisations and researchers; for example, Open AI have published a basic model
3256 card of their generative language model and Google provided a sample model card
3257 for their Toxicity analyser [243]. Model cards are also being considered for high
3258 stakes environments such as clinical decision making [316], where they facilitate
3259 overarching governance regimes on how and when models can be used. For our case
3260 study, a combination of Model Card for the classifier and datasheet for training data
3261 could have provided a valuable, easy to digest first step to support an evaluation of the
3262 classifier for our context. However, the current specification of datasheet contents is
3263 very broad and lacks detailed directions for those completing the information. Had
3264 all the required information been provided to sufficient detail, including a highlight
3265 of the importance of consistency training, we could have better assessed whether an
3266 emotion assessment was appropriate. However, we would still have had to complete
3267 the study—but with rater consistency training and a better appreciation of the limits
3268 of the classifier.

3269 7.3.4 Threats to validity

3270 We sampled only 25 posts for inter-rater reliability and it remains a limitation of our
3271 analysis. Although, only 25 posts were sampled for inter-rater reliability, the first
3272 author reviewed an additional 500 posts and the inconsistency observed from the 25
3273 posts maps to the broader analysis.

²<https://modelcards.withgoogle.com/model-reports> last accessed 25 May 2020.

3274 **7.4 Conclusion**

3275 We started the journey presented in this work with an idea to use existing AI tech-
3276 niques to *automatically* investigate what other developers think of cloud intelligent
3277 services. This translated into our attempt to use a pre-trained model that learnt from
3278 posts provided by software engineers on SO.

3279 Developers learn, improve and deepen their skills from documentation, formal or
3280 self-paced education, experience, and sharing their knowledge. Good documentation
3281 often forms the foundation that enables learning and also to create educational
3282 aids. In this work, we presented an observation case study that highlights a set of
3283 gaps in how a peer-reviewed model, published in the field of software engineering,
3284 lacks information about the limitations both within the documentation, as well as
3285 the articles published. To resolve these gaps, we investigated if new solutions
3286 that are being proposed such as model cards would have helped us. Model cards
3287 and datasheets will be a necessary and helpful first step, but as such we found
3288 their specification to be insufficient and additional guidance is required for those
3289 completing the cards and datasheets. Although we study only one pre-trained model
3290 in depth, our analysis shows that there are gaps in proposed solutions that can be
3291 addressed and our future work will focus on investigating other models and IWSs to
3292 develop a more detailed documentation approach, specifically those that are being
3293 aimed for software engineering.

CHAPTER 8

3294

3295

3296

Better Documenting Computer Vision Services[†]

3297

3298 **Abstract** Using cloud-based computer vision services (CVSs) is gaining traction with
3299 developers for many applications for many reasons: developers can simply access these
3300 AI-components through familiar RESTful APIs, and need not orchestrate large training and
3301 inference infrastructures or curate and label large training datasets. However, while their
3302 APIs *seem* familiar to use, their non-deterministic run-time behaviour and evolution profile
3303 are not adequately communicated to developers, and this results in developers struggling
3304 to use such APIs in-practice. Therefore, improving these services' API documentation is
3305 paramount, as a more complete document facilities the development process of intelligent
3306 software. This study presents an analysis of what facets a 'complete' API document should
3307 have, as synthesised into a taxonomy from 21 academic studies via a systematic mapping
3308 study. We triangulate these findings from literature against 83 developers to assess the
3309 efficacy and utility in-practice of such knowledge. We produce two weighted 'scores'
3310 for each dimension in our taxonomy based on (i) the number of papers producing these
3311 outcomes and their citation count and (ii) the extent to which developers *agree* with the
3312 recommendations arising from these studies (based on our survey). Furthermore, we apply
3313 the taxonomy to three popular CVSs and assess their compliance, producing a third 'score'
3314 using the taxonomy to identify 12 suggested improvements to the API documentation of
3315 these intelligent web services.

3316 8.1 Introduction

3317 Improving API documentation quality is a valuable task for any API—an extensive
3318 API document facilitates productivity, and therefore improved quality is better
3319 engineered into a system [235]. Where application developers integrate new services
3320 (such as computer vision services (CVSs) [87]) into their systems via APIs, their

[†]This chapter is originally based on A. Cummaudo, R. Vasa, and J. Grundy, "Assessing API documentation knowledge for computer vision services," 2020, Unpublished. Terminology has been updated to fit this thesis.

productivity is affected either by inadequate skills (“*I’ve never used an API like this, so must learn from scratch*”) or, where their skills are adequate, an imbalanced cognitive load that causes excessive context switching (“*I have the skills for this, but am confused or misunderstand*”). This is commonly seen in the emerging computer vision web services space, where the documentation does not yet completely or correctly describe the APIs in full [90].

What causes a developer to be confused and how to mitigate it via an improved API document has been largely explored for conventional APIs. Various studies have provided a myriad of recommendations based on both qualitative and quantitative analysis of developer opinion. Such recommendations propose ways by which developers, managers and solution architects can construct systems better with improved documentation. However, while previous works have covered certain aspects of API usage, many have lacked a systematic review of literature and do not offer a taxonomy to consolidate these guidelines together. For example, some studies have considered the technical implementation improving API usability or tools to generate (or validate) API documentation from its source code (e.g., [223, 258, 362]); still lacks a consolidated effort to capture recommendations on how to *manually write* complete, correct, and effective API documentation. The works that *do* produce these recommendations from literature are largely scattered across multiple sources, and systematically capturing the information into a readily accessible, consolidated framework (designed to assist writing API documentation) must be validated in real-world circumstances to assess its efficacy with practitioners and existing documentation [86].

As a real-world use case, consider an intelligent web service (IWS)—such as CVSs—in which an AI-based component produces a non-deterministic result based on a machine-learnt data-driven algorithm, rather than a predictable, rule-driven one [87]. These services use machine intelligence to make predictions on images such as object labelling or facial recognition [386, 397, 398, 399, 400, 407, 411, 419, 420, 421, 425, 438, 439, 472, 473]. The impacts of poor and incomplete documentation results in developer complaints on online discussion forums such as Stack Overflow [90]. Many comments show that developers do not think in the non-deterministic mental model of the designers who created the CVSs. They ask many varied questions from their peers to try and clarify their understanding.

This paper significantly extends our previous work [86] by evaluating our API documentation taxonomy in two additional contexts. In our previous work, we developed a weighted metric for each dimension and category based on how many literary sources agree that the aspects of our taxonomy should be implemented. We refer to this as an ‘in-literature’ agreement score. We build upon this facet but *in-practice* by assessing the efficacy of our taxonomy against developers using a survey built upon an interpretation of the System Usability Scale (SUS) [61]. We produce a second weighting for the dimensions and categories of the taxonomy, referred to as a ‘in-practice’ agreement score. We then compare both the in-literature and in-practice scores directly, thereby contrasting the statistical agreement the two have. Lastly, we assess the taxonomy against three popular CVSs, namely Google Cloud Vision [411], Amazon Rekognition [386] and Azure Computer Vision [425].

3366 For each category in our taxonomy, we assess whether the respective service's
3367 documentation contains, partially-contains or does not contain the recommendation.
3368 From this, we triangulate each category's in-literature and in-practice score against
3369 the service's level of inclusion of the recommendation, thereby making a judgement
3370 as to where the services can improve their documentation to make them more
3371 complete.

3372 The primary contributions in this work are:

- 3373 • a systematic mapping study (SMS) consisting of 21 studies that capture what
3374 knowledge or artefacts should be contained within API documentation;
- 3375 • a five dimensional taxonomy consisting of 34 recommendations based on those
3376 consolidated from the 21 studies;
- 3377 • a score metric for each recommendation based on the number of papers that
3378 agree with the recommendation;
- 3379 • a score metric assessing the efficacy of the 34 recommendations that empirically
3380 reflects what is important to document from a *practitioner* point of view;
3381 and,
- 3382 • a heuristic validation of each recommendation against CVSs, assessing where
3383 existing CVS API documentation needs improvement.

3384 After performing our SMS on what API knowledge should be captured in documentation
3385 to assist API designers, we propose our taxonomy consisting of the
3386 following dimensions: (1) Usage Description; (2) Design Rationale; (3) Domain
3387 Concepts; (4) Support Artefacts; and (5) Documentation Presentation. Following
3388 this, we adopted the SUS surveying technique to assess the overall utility of each
3389 of these recommendations, producing a survey consisting of 43 questions. This
3390 survey was then tested three times within our research group: firstly against three
3391 researchers for feedback on the survey's design, secondly against three software
3392 engineers in our research group with varying levels of experience for developers
3393 for test-retest reliability [192], thirdly against 22 software engineers in our research
3394 group for wider feedback on the survey. Given these feedback improvements, we
3395 surveyed 83 external developers between May 2019 to October 2019, and then analysed
3396 the relevance of each recommendation from the practitioner's viewpoint. We
3397 also assessed the three CVSs for inclusion of each recommendation, and once our
3398 surveys were complete, determined a weighted 'score' of each service to see where
3399 improvements to their documentation was made.

3400 This paper is structured as thus: Section 8.2 presents related work in the areas
3401 of API usability, intelligent CVSs, and the SUS; Section 8.3 is divided into two
3402 subsections, the first describing how primary sources were selected in a SMS with the
3403 second describing the development of our taxonomy from these sources; Section 8.4
3404 presents the taxonomy; Section 8.5 describes how we developed a survey instrument
3405 of 43 questions to validate the taxonomy against developers, and assess its efficacy
3406 against the three popular CVSs selected to make 12 suggested improvements to
3407 the existing service API documentation; Section 8.6 presents the findings from our
3408 validation analysis and the weightings for the taxonomy; Section 8.7 describes the

³⁴⁰⁹ threats to validity of this work and Section 8.8 provides concluding remarks and the
³⁴¹⁰ future directions of this study. Additional materials are provided in Appendix C.

³⁴¹¹ 8.2 Related Work

³⁴¹² 8.2.1 API Usability and Documentation Knowledge

³⁴¹³ Use of the SMS approach has explored developer experience and API usability.
³⁴¹⁴ A 2018 study reviewed 36 API documentation generation tools and approaches, and
³⁴¹⁵ analysed the tools developed and their inputs and documentation outputs [258]. The
³⁴¹⁶ findings from this study emphasise that the largest effort in API documentation tool-
³⁴¹⁷ ling is to assist developers to generate either example code snippets and/or templates
³⁴¹⁸ or natural language descriptions of the API directly from the program’s source code.
³⁴¹⁹ These snippets or descriptions can then be placed in the API documentation, thereby
³⁴²⁰ increasing the efficiency at which API documentation can be written. Additionally,
³⁴²¹ tools from 12 studies target the maintainability of existing APIs of existing APIs,
³⁴²² with tools from 11 studies target the correctness and accuracy of the documentation
³⁴²³ by validating that what is written in the documentation is accurate to the technical
³⁴²⁴ structure of the API. From the end-developer’s perspective, some tools (17 studies)
³⁴²⁵ help target improvements to the developer’s understandability and learnability of
³⁴²⁶ new APIs by linking in examples directly with questions such as on Stack Overflow.

³⁴²⁷ However, the results from this study regards the *tooling* used to either assist in
³⁴²⁸ producing, validating or learning from API documentation. While this is a systematic
³⁴²⁹ study with key insights into the types of tooling produced, there is still a gap for a
³⁴³⁰ SMS in what *guidelines* have been produced by the literature in developing natural-
³⁴³¹ language documentation itself and how well developers *agree* to those guidelines,
³⁴³² which our work has addressed.

³⁴³³ Watson [362] performed a heuristic assessment from 35 popular APIs against 11
³⁴³⁴ high-level universal design elements of API documentation. This study highlighted
³⁴³⁵ how many APIs, even popular ones, fail to grasp these basic design elements.
³⁴³⁶ For example, 25% of the documentation sets did not provide any basic overview
³⁴³⁷ documentation to the API. The heuristics used within Watson’s study is based on
³⁴³⁸ only three seminal works and only contains 11 design elements—our study extends
³⁴³⁹ these heuristics and structures them into a consolidated, hierarchical taxonomy which
³⁴⁴⁰ we then validate against practitioners.

³⁴⁴¹ A taxonomy of distinct knowledge patterns within reference documentation
³⁴⁴² by Maalej and Robillard [223] classified 12 distinct knowledge types. The tax-
³⁴⁴³ onomy was then evaluated against the JDK 6 and .NET 4.0 frameworks, and showed
³⁴⁴⁴ that the functionality and structure of these APIs are well-communicated, although
³⁴⁴⁵ core concepts and rationale about the API are quite rarer to see. The authors also
³⁴⁴⁶ identified low-value ‘non-information’—described as documentation that provides
³⁴⁴⁷ uninformative boilerplate text with no insight into the API at all—which was sub-
³⁴⁴⁸ stantially present in the documentation of methods and fields in the two frameworks.
³⁴⁴⁹ They recommend that developers factor their 12 distinct knowledge types into the
³⁴⁵⁰ process of code documentation, thereby preventing low-value non-information. The

3451 development of their taxonomy consisted of questions to model knowledge and information,
3452 thereby capturing the reason about disparate information units independent
3453 to context; a key difference to this paper is the systematic taxonomy approach utilised.

3454 8.2.2 Adapting the System Usability Scale

3455 The SUS was first introduced by Brooke as early as 1986 as a “quick and dirty”
3456 survey scale to easily assess the overall usability of a product or service in a timely
3457 manner. Its popularity in the usability community demonstrated the need for a
3458 tool that can collect a quantifiable rating of usability from a participant’s subjective
3459 opinion, and was later published in [61]. Since, its adoption as an industry standard is
3460 widely demonstrated [23, 62] and studies have adopted its ease of use for generalised
3461 purposes.

3462 While translation of the SUS into other languages [48, 228, 307] is generally
3463 the most adapted form of Brooke’s original survey, some studies have proposed
3464 alternative measurement models to the SUS, such as separating the usability and
3465 learnability components of the survey into a two-dimensional structure [48]. Other
3466 adaptations of the SUS include a 2014 study that proposed a usability scale based
3467 on the SUS for Handheld Augmented Reality applications [306] conceptualised
3468 against comprehensibility and manipulability. However, few studies have designed
3469 questionnaires patterned from the SUS in other contexts, and to our knowledge, this
3470 study presents an initial attempt at doing so in the API documentation knowledge
3471 domain.

3472 8.2.3 Computer Vision Services

3473 Recent studies into cloud-based CVSSs have demonstrated that poor reliability and
3474 robustness in computer vision can ‘leak’ into end-applications if such aspects are
3475 not sufficiently appreciated by developers. A study by Hosseini et al. [161] showed
3476 that Google Cloud Vision’s labelling fails when as little as 10% noise is added to the
3477 image. Facial recognition classifiers are easily confused by modifying pixels of a face
3478 and using transfer learning to adapt one person’s face into another [357]. Our own
3479 prior work found that the non-deterministic evolution of these types of services is not
3480 adequately communicated to developers [87], resulting in lost developer productivity
3481 whereby developers ask fundamental questions about the concepts behind these
3482 services, how they work, and where better documentation can be found [90]. This
3483 paper continues this line of research by providing a means for service providers to
3484 better document their services using a taxonomy and suggested improvements.

3485 8.3 Taxonomy Development

3486 We developed our taxonomy under two primary phases. First, we conducted a SMS
3487 identifying API documentation studies, following guidelines by Kitchenham and
3488 Charters [191] and Petersen et al. [276] (Section 8.3.1). A high level overview of
3489 this first phase is given in Figure 8.2. Second, we followed a software engineering
3490 taxonomy development method by Usman et al. [351] (Section 8.3.2) based on the

³⁴⁹¹ findings of our SMS, which involved an extensive validation involving real-world
³⁴⁹² developers and contextualised with computer vision APIs (Section 8.5).

³⁴⁹³ 8.3.1 Systematic Mapping Study

³⁴⁹⁴ 8.3.1.1 Research Questions (RQs)

³⁴⁹⁵ The first step in producing our SMS was to pose two RQs:

- ³⁴⁹⁶ • **RQ1:** What knowledge do API documentation studies contribute?
- ³⁴⁹⁷ • **RQ2:** How is API documentation studied?

³⁴⁹⁸ Our intent behind RQ1 was to collect as many studies provided by literature on how
³⁴⁹⁹ API documentation should be written using natural language (i.e., not using assistive
³⁵⁰⁰ tooling). This helped us shape and form the taxonomy provided in Section 8.4.
³⁵⁰¹ Secondly, RQ2's intent was to understand how the studies derive at their conclusions,
³⁵⁰² thereby helping us identify gaps in literature where future studies can potentially
³⁵⁰³ focus.

³⁵⁰⁴ 8.3.1.2 Automatic Filtering

³⁵⁰⁵ As done in similar software engineering studies [130, 138, 351], we explored au-
³⁵⁰⁶ tomatic filtering of online databases. We defined which SWEBOK knowledge
³⁵⁰⁷ areas [166] were relevant to devise a search query. Our search query was built using
³⁵⁰⁸ related knowledge areas, relevant synonyms, and the term 'software engineering'
³⁵⁰⁹ (for comprehensiveness) all joined with the OR operator. Due to the lack of a
³⁵¹⁰ standard definition of an API, we include the terms: 'API' and its expanded term;
³⁵¹¹ software library, component and framework; and lastly software development kit
³⁵¹² (SDK). These too were joined with the OR operator, appended with an AND. Lastly,
³⁵¹³ the term 'documentation' was appended with an AND. Our final search string was:

```
( "software design" OR "software architecture" OR "software construction" OR "software development"
OR "software maintenance" OR "software engineering process" OR "software process" OR "software
lifecycle" OR "software methods" OR "software quality" OR "software engineering professional practice"
OR "software engineering" ) AND ( API OR "application programming interface" OR "software library"
OR "software component" OR "software framework" OR sdk OR "software development kit" ) AND (
documentation )
```

³⁵¹⁴ We executed the query on all available metadata (title, abstract and keywords) in
³⁵¹⁵ May 2019 against Web of Science¹ (WoS), Compendex/Inspec² (C/I) and Scopus³.
³⁵¹⁶ We selected three particular primary sources given their relevance in software en-
³⁵¹⁷ gineering literature (containing the IEEE, ACM, Springer and Elsevier databases)
³⁵¹⁸ and their ability to support advanced queries [60, 191]. A total 4,501 results⁴ were
³⁵¹⁹ found, with 549 being duplicates. Table 8.1 displays our results in further detail (du-
³⁵²⁰ plicates not omitted); Figure 8.1 shows an exponential trend of API documentation

¹<http://apps.webofknowledge.com> last accessed 23 May 2019.

²<http://www.engineeringvillage.com> last accessed 23 May 2019.

³<http://www.scopus.com> last accessed 23 May 2019.

⁴Raw results can be located at <http://bit.ly/2KxBLs4>.

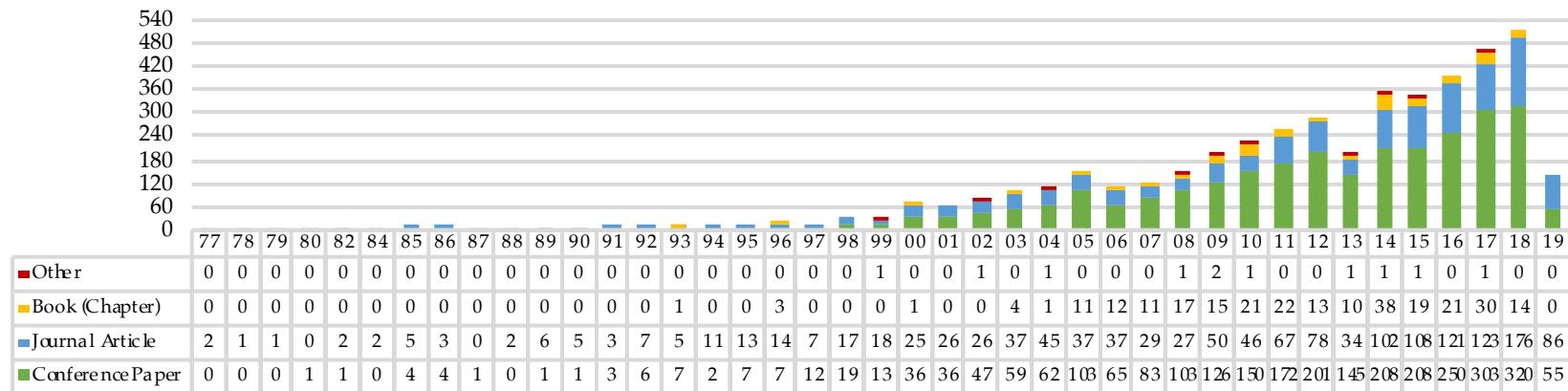


Figure 8.1: Search results by year and venue type.

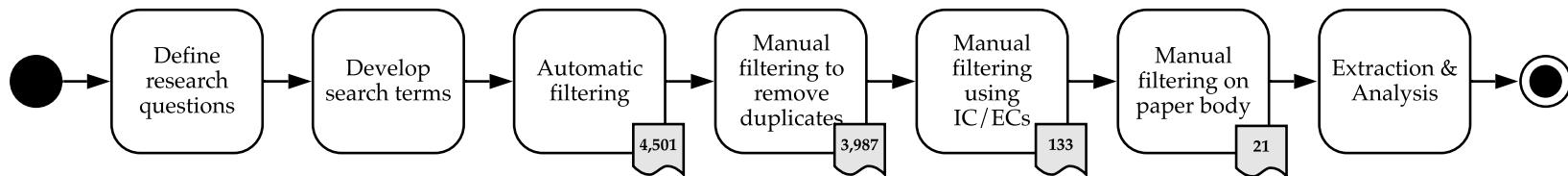


Figure 8.2: A high level overview of the filtering steps from defining and executing our search query to the data extraction of our primary studies. Number of accepted papers resulting from each filtering step is shown.

Table 8.1: Search results and publication types

Publication type	WoS	C/I	Scopus	Total
Conference Paper	27	442	2353	2822
Journal Article	41	127	1236	1404
Book	23	17	224	264
Other	0	5	6	11
Total	91	591	3819	4501

3521 publications produced within the last two decades. (As this search was conducted
3522 in May 2019, results taper in 2019.)

3523 8.3.1.3 *Manual Filtering*

3524 A follow-up manual filtering stage followed the 4,501 results obtained by automatic
3525 filtering. As described below, we applied the following inclusion criteria (IC) and
3526 exclusion criteria (EC) to each result:

- 3527 **IC1** Studies must be relevant to API documentation: specifically, we exclude
3528 studies that deal with improving the technical API usability (e.g., improved
3529 usage patterns);
- 3530 **IC2** Studies must propose new knowledge or recommendations to document
3531 APIs;
- 3532 **IC3** Studies must be relevant to software engineering as defined in SWEBOk;
- 3533 **EC1** Studies where full-text is not accessible through standard institutional databases;
- 3534 **EC2** Studies that do not propose or extend how to improve the official, natural
3535 language documentation of an API;
- 3536 **EC3** Studies proposing a third-party tool to enhance existing documentation or
3537 generate new documentation using data mining (i.e., not proposing strategies
3538 to improve official documentation);
- 3539 **EC4** Studies not written in English;
- 3540 **EC5** Studies not peer-reviewed.

3541 Each of these ICs and ECs were applied to every paper after exporting all
3542 metadata of our results to a spreadsheet. The first author then curated the publications
3543 using the following revision process.

3544 Firstly, we read the publication source—to rapidly omit non-software engineering
3545 papers—as well as the author keywords, title, and abstract of all 4,501 studies.
3546 As some studies were duplicated between our three primary sources, we needed to
3547 remove any repetitions. We sorted and reviewed any duplicate DOIs and fuzzy-
3548 matched all very similar titles (i.e., changes due to punctuation between primary
3549 sources), thereby retaining only one copy of the paper from a single database. Sim-
3550ilarly, as there was no limit to our date ranges, some studies were republished in
3551 various venues (i.e., same title but different DOIs). These were also removed using

3552 fuzzy-matching on the title, and the first instance of the paper's publication was
3553 retained. This second phase resulted in 3,987 papers.

3554 Secondly, we applied our inclusion and exclusion criteria to each of the 3,987
3555 papers by reading the abstract. Where there was any doubt in applying the criteria
3556 to the abstract alone, we automatically shortlisted the study. We rejected 427 studies
3557 that were unrelated to software engineering, 3,235 were not directly related to docu-
3558 menting APIs (e.g., to enhance coding techniques that improve the overall developer
3559 usability of the API), 182 proposed new tools to enhance API documentation or
3560 used machine learning to mine developer's discussion of APIs, and 10 were not in
3561 English. This resulted in 133 studies being shortlisted to the final phase.

3562 Thirdly, we re-evaluated each shortlisted paper by re-reading the abstract, the
3563 introduction and conclusion. We removed a further 64 studies that were on API
3564 usability or non API-related documentation (i.e., code commenting). At this stage,
3565 we decided to refine our exclusion criteria to better match the research goals of this
3566 study by including the word 'natural language' documentation in EC2. This removed
3567 studies where the focus was to improve technical documentation of APIs such as
3568 data types and communication schemas. Additionally, we removed 26 studies as
3569 they were related to introducing new tools (EC3), 3 were focused on tools to mine
3570 API documentation, 7 studies where no recommendations were provided, 2 further
3571 duplicate studies, and a further 10 studies where the full text was not available,
3572 not peer reviewed or in English. Books are commonly not peer-reviewed (EC5),
3573 however no books were shortlisted within these results. This final stage resulted in
3574 21 primary studies for further analysis, and the mapping of primary study identifiers
3575 to references S1–21 can be found in Appendix C.3.

3576 As a final phase, we conducted reliability analysis of our shortlisting method.
3577 We conducted intra-rater reliability of our 133 shortlisted papers using the test-
3578 retest approach suggested by Kitchenham and Charters [191]. We re-evaluated a
3579 random sample of 10% of the 133 shortlisted papers a week after initial studies were
3580 shortlisted. This resulted in *substantial agreement* [206], measured using Cohen's
3581 kappa ($\kappa = 0.7547$).

3582 8.3.1.4 Data Extraction & Systematic Mapping

3583 Of the 21 primary studies, we conducted abstract key-wording adhering to Petersen
3584 et al.'s guidelines [276] to develop a classification scheme. An initial set of keywords
3585 were applied for each paper in terms of their methodologies and research approaches
3586 (RQ2), based on an existing classification schema used in the requirements engineer-
3587 ing field by Wieringa et al. [369]. These are: *evaluation papers*, which evaluates
3588 existing techniques in-practice; *validation papers*, which investigates proposed tech-
3589 niques not yet implemented in-practice; *experience papers*, which do investigate or
3590 evaluate either proposed or existing techniques, but presents insightful experiences
3591 of authors that warrant communication to other practitioners; and *philosophical pa-
3592 pers*, which presents new conceptual frameworks that describes a language by which
3593 we can describes our observations of existing or new techniques, thereby implying
3594 a new viewpoint for understanding phenomena.

Table 8.2: Data extraction form

Data item(s)	Description
Citation metadata	Title, author(s), years, publication venue, publication type
Key recommendation(s)	As per IC2, the study must propose at least one recommendation on what should be captured in API documentation
Evaluation method	Did the authors evaluate their recommendations? If so, how?
Primary technique	The primary technique used to devise the recommendation(s)
Secondary technique	As above, if a second study was conducted
Tertiary technique	As above, if a third study was conducted
Research type	The research type employed in the study as defined by Wieringa et al.'s taxonomy

3595 After all primary studies had been assigned keywords, we noticed that all papers
 3596 used field study techniques, and thus we consolidated these keywords using Singer
 3597 et al.'s framework of software engineering field study techniques [323]. Singer et al.
 3598 captures both study techniques *and* methods to collect data within the one framework,
 3599 namely: *direct techniques*, including brainstorming and focus groups, interviews and
 3600 questionnaires, conceptual modelling, work diaries, think-aloud sessions, shadowing
 3601 and observation, participant observation; *indirect techniques*, including instrumenting
 3602 systems, fly-on-the-wall; and *independent techniques*, including analysis of work
 3603 databases, tool use logs, documentation analysis, and static and dynamic analysis.

3604 Table 8.2 describes our data extraction form, which was used to collect relevant
 3605 data from each paper. Figure 8.3 presents our systematic mapping, where each study
 3606 is mapped to one (or more, if applicable) of methodologies plotted against Wieringa
 3607 et al.'s research approaches. We find that a majority of these studies survey develop-
 3608 ers using direct techniques (i.e., interviews and questionnaires) and some performing
 3609 structured documentation analysis. Few studies report recent experiences, with the
 3610 majority of API documentation knowledge being evaluation research, and some val-
 3611 idation studies. There are few experience papers describing anecdotal evidence of
 3612 API documentation knowledge, and almost no philosophical papers that describe new
 3613 conceptual ways at approaching API documentation as a large majority of existing
 3614 work either evaluates existing (in-practice) strategies or validates the effectiveness
 3615 of new strategies.

3616 **8.3.2 Development of the Taxonomy**

3617 A majority of taxonomies produced in software engineering studies are often made
 3618 extemporaneously [351]. For this reason, we decided to proceed with a systematic
 3619 approach to develop our taxonomy using the guidelines provided by Usman et al.
 3620 [351], which are extended from lessons learned in more mature domains. In this
 3621 subsection, we outline the 4 phases and 13 steps taken to develop our taxonomy

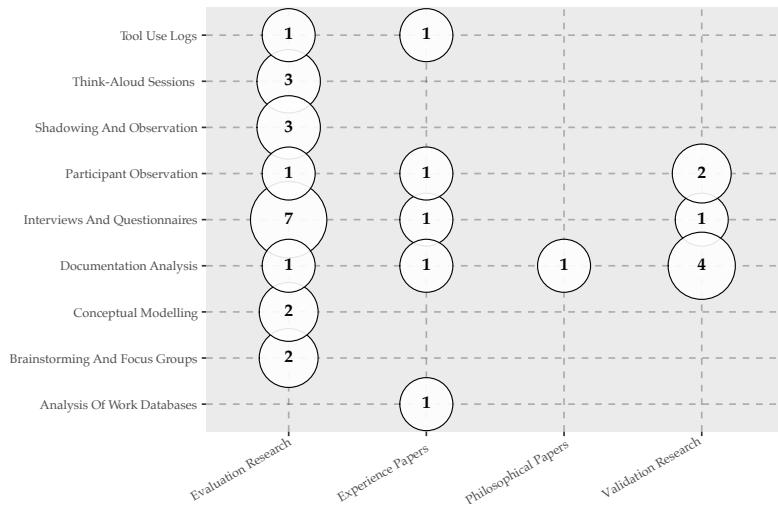


Figure 8.3: Systematic map: field study technique vs research type

3622 based on Usman et al.'s technique. Usman et al.'s final *validation* phase is largely
3623 detailed within Section 8.5 after we present our taxonomy in Section 8.4.

3624 8.3.2.1 *Planning phase*

3625 The preliminary phase involves answering the following:

- 3626 **(1) define the software engineering knowledge area:** The software engineering
3627 knowledge area, as defined by the SWEBOK, is software construction;
- 3628 **(2) define the objective:** The main objective of the proposed taxonomy is to define
3629 a set of categories that enables to classify different facets of natural-language
3630 API documentation knowledge (not API *usability* knowledge) as reported in
3631 existing literature;
- 3632 **(3) define the subject matter:** The subject matter of our proposed taxonomy is
3633 documentation artefacts of APIs;
- 3634 **(4) define the classification structure:** The classification structure of our proposed
3635 taxonomy is *hierarchical*;
- 3636 **(5) define the classification procedure:** The procedure used to classify the docu-
3637 mentation artefacts is qualitative;
- 3638 **(6) define the data sources:** The basis of the taxonomy is derived from field study
3639 techniques (see Section 8.3.1.4).

3640 8.3.2.2 *Identification and extraction phase*

3641 The second phase of the taxonomy development involves **(7) extracting all terms** and
3642 *concepts* from relevant literature, which we have achieved from our SMS. These
3643 terms are then consolidated by **(8) performing terminology control**, as some terms
3644 may refer to different concepts and vice-versa.

3645 8.3.2.3 *Design phase*

3646 The design phase identified the core dimensions and categories within the extracted
3647 data items. The first step is to (9) *identify and define taxonomy dimensions*; for this
3648 study we utilised a bottom-up approach to identify each dimension, i.e., extracting the
3649 categories first and then nominating which dimensions these categories fit into using
3650 an iterative approach. As we used a bottom-up approach, step (9) also encompassed
3651 the second stage of the design phase, which is to (10) *identify and describe the*
3652 *categories of each dimension*. Thirdly, we (11) *identify and describe relationships*
3653 between dimensions and categories, which can be skipped if the relationships are
3654 too close together, as is the case of our grouping technique which allows for new
3655 dimensions and categories to be added. The last step in this phase is to (12) *define*
3656 *guidelines for using and updating the taxonomy*. The taxonomy is as simple as a
3657 checklist that can be heuristically applied to an API document, and each dimension
3658 is malleable and covers a broad spectrum of artefacts; while we do not anticipate
3659 any further dimensions to be added, new categories can easily be fitted into one of
3660 the dimensions (see Section 8.8). We provide guidelines for use in our application
3661 of the taxonomy against CVSSs within Sections 8.4 and 8.6.

3662 8.3.2.4 *Validation phase*

3663 In the final phase of taxonomy development, taxonomy designers must (13) *validate*
3664 *the taxonomy* to assess its usefulness. Usman et al. [351] describe three approaches to
3665 validate taxonomies: (i) orthogonal demonstration, in which the taxonomy's orthog-
3666 onality is demonstrated against the dimensions and categories, (ii) benchmarking
3667 the taxonomy against similar classification schemes, or (iii) utility demonstration by
3668 applying the taxonomy heuristically against subject-matter examples. In our study,
3669 we adopt utility demonstration by use of a survey and heuristic application of the
3670 taxonomy against real-world case-studies (i.e., within the domain of CVSSs). This is
3671 discussed in greater detail within Section 8.5.

3672 **8.4 API Documentation Knowledge Taxonomy**

3673 Our taxonomy consists of five dimensions (labelled A–E). We expand these five di-
3674 mensions into 34 categories (sub-dimensions). Each dimension respectively covers:

- 3675 • **[A] Usage Description** on *how* to use the API for the developer's intended
3676 use case;
- 3677 • **[B] Design Rationale** on *when* the developer should choose this API for a
3678 particular use case;
- 3679 • **[C] Domain Concepts** of the domain behind the API to understand *why* this
3680 API should be chosen for this domain;
- 3681 • **[D] Support Artefacts** that describe *what* additional documentation the API
3682 provides; and
- 3683 • **[E] Documentation Presentation** to help organise the *visualisation* of the
3684 above information.

[A] Usage Description

- [A1] Quick-start guides #3
- [A2] Low-level reference manual #3 SH
- [A3] Explanation of high level architecture
- [A4] Introspection source code comments SH
- [A5] Code snippets of basic component function #2 #1 VH
- [A6] Step-by-step tutorials with multiple components #2 SH
- [A7] Downloadable production-ready source code
- [A8] Best-practices of implementation
- [A9] An exhaustive list of all components
- [A10] Minimum system requirements to use the API
- [A11] Instructions to install/update the API and its release cycle #4
- [A12] Error definitions describing how to address problems #5

[B] Design Rationale

- [B1] Entry-point purpose of the API #4
- [B2] What the API can develop
- [B3] Who should use the API
- [B4] Who will use the applications built using the API
- [B5] Success stories on the API
- [B6] Documentation comparing similar APIs to this API
- [B7] Limitations on what the API can/cannot provide #1

[C] Domain Concepts

- [C1] Relationship between API components and domain concepts
- [C2] Definitions of domain terminology
- [C3] Documentation for nontechnical audiences

[D] Support Artefacts

- [D1] FAQs
- [D2] Troubleshooting hints
- [D3] API diagrams
- [D4] Contact for technical support NH
- [D5] Printed guide
- [D6] Licensing information

[E] Documentation Presentation

- [E1] Searchable knowledge base
- [E2] Context-specific discussion forums
- [E3] Quick-links to other relevant components
- [E4] Structured navigation style
- [E5] Visualised map of navigational paths
- [E6] Consistent look and feel #5

Figure 8.4: Our proposed taxonomy on what artefacts should be documented in a complete API document.

3685 Further descriptions of the categories encompassing each dimension are given within
3686 Figure 8.4 and Appendix C.1, coded as $[Xi]$, where i is the category identifier within
3687 a dimension, $X \in \{A, B, C, D, E\}$.

3688 Appendix C.1 shows which of the primary sources (S1–21) provide the rec-
3689 ommendation described as well as an ‘in-literature score’ (ILS). This score is a
3690 weighting calculated as a percentage of the number of primary studies that make the
3691 recommendation divided by the total of primary studies, and indicates the overall
3692 level of agreement that academic sources suggest these documentation artefacts.
3693 This score is contrasted to the ‘in-practice score’ (IPS) which indicates the over-
3694 all level of agreement that *practitioners* think such documentation artefacts are
3695 needed. Further details about the ILS and IPS values, how they were calculated and
3696 analysed for each category, and a rigorous contrast between the two are provided
3697 Sections 8.5.1.2 and 8.6.1 to 8.6.3. For comparative purposes, we illustrate a colour
3698 scale (from red to green) to indicate the relevancy weight between ILS and IPS
3699 values in Appendix C.1: for example, while quick-start guides [A1] are few refer-
3700 enced in academic sources at 14%, they are generally well-desired by practitioners
3701 88% agreement. We then provide three columns that assesses the presence of these
3702 documentation artefacts against three popular CVSSs: Google Cloud Vision, AWS’s
3703 Rekognition, and Azure Cloud Vision (abbreviated to GCV, AWS and ACV). A
3704 fully shaded circle (●) indicates that the documentation artefact was clearly found
3705 in the service, while a half-shaded circle (◐) indicates that the artefact was only
3706 partially present. An outlined circle (○) indicates that the service lacks the indicated
3707 documentation artefact within our taxonomy. This empirical assessment is further
3708 detailed in Section 8.6.5, which outlines concrete areas in the respective services’
3709 documentation where improvements could be made, as well as hyperlinks to the
3710 documentation where relevant.

3711 Figure 8.4 illustrates these findings, with underlines indicating key artefacts and
3712 various iconography to indicate specific results. The computer icon (💻) includes a
3713 ranking from 1–5 of the top five most recommended artefacts according to devel-
3714 opers, as calculated from their relevant IPS scores. Conversely, the book icon (📖)
3715 indicates the rankings of the top five most recommended artefacts according to liter-
3716 ature. For example, while literature suggests the most useful documentation artefact
3717 are API usage description code snippets [A5], in-practice, we find that developers
3718 prefer design rationale on what the limitations of API are [B7] with code snippets
3719 coming in second place. Where there is strong agreement between developers and
3720 literature (within a standard deviation of 0.15) we use the handshake icon (🤝) and
3721 list whether both agree if the category is Very Helpful (VH), Slightly Helpful (SH) or
3722 Not Helpful (NH). Further details on this explanation are provided in Section 8.6.3.
3723 Lastly, we provide iconography for the presence (✓) or non-presence (✗) of these
3724 artefacts in *all three* CVSSs assessed, per Section 8.6.2.

3725 8.5 Validating our Taxonomy

3726 8.5.1 Survey Study

3727 8.5.1.1 Designing the Survey

3728 We followed the guidelines by Kitchenham and Pfleeger [192] on conducting per-
3729 sonal opinion surveys in software engineering to validate our survey. In developing
3730 our survey instrument, we shaped questions around each of our 5 dimensions and
3731 34 categories. To achieve this, we used Brooke's SUS [61] as inspiration and re-
3732 shaped the 34 categories around a question. Each dimension was marked a numeric
3733 question (3–7), and alphabetic sub-questions were marked for each sub-dimension
3734 or category.

3735 We used closed questioning where respondents could choose an answer on a
3736 5-point Likert-scale (1=*strongly disagree*, 2=*somewhat disagree*, 3=*neither agree*
3737 nor *disagree*, 4=*slightly agree* and 5=*strongly agree*). Like Brooke's study, each
3738 question alternated in positive and negative sentiment. Half of our questions were
3739 written where a likely common response would be in strong agreement and vice-
3740 versa for the other half, such that participants would have to “read each statement
3741 and make an effort to think whether they would agree or disagree with it” [61]. For
3742 example, the question regarding [B7] on API limitations was framed as: “*I believe it*
3743 *is important to know about what the limitations are on what the API can and cannot*
3744 *provide*” (Q4g), whereas the question regarding [C1] on domain concepts of the API
3745 was framed as: “*I wouldn't read through theory about the API's domain that relates*
3746 *theoretical concepts to API components and how both work together*” (Q5a).

3747 In addition, the remaining eight questions asked demographical information.
3748 An extra open question asked for further comments. The full survey is provided in
3749 Appendix C.4.

3750 8.5.1.2 Evaluating the Survey

3751 After the first pass at designing questions was completed, we evaluated our survey
3752 on three researchers within our research group for general feedback. This resulted
3753 in minor changes, such as slight re-wording of questions, clarifying the difference
3754 between web services and web APIs, and providing specific questions with examples
3755 (some with images). For example, the question regarding [A9] on an exhaustive list
3756 of all major components in the API was framed as “*I believe an exhaustive list*
3757 *of all major components in the API without excessive detail would be useful when*
3758 *learning an API*” (Q3i) with the example “e.g., a computer vision web API might
3759 *list object detection, object localisation, facial recognition, and facial comparison*
3760 *as its 4 components*”.

3761 After this, we conducted reliability analysis using a test-retest approach on three
3762 developers within our group seven weeks apart. This was calculated using the `irr`
3763 computational R package [127] (as suggested in [148]) and resulted in an average
3764 intra-class correlation of 0.63 which indicates a good overall index of agreement [78].

3765 8.5.1.3 Recruiting Participants

3766 Our target population for the study was application software developers with varying
3767 degrees of experience (including those who and who have not used CVSs or related
3768 tools before) and varying understanding of fundamental machine learning concepts.
3769 We began by recruiting software developers within our research group using a
3770 group-wide message sent on our internal messaging system. Of the 44 developers in
3771 our group's engineering cohort, 22 responses were returned, indicating an internal
3772 response rate of 50%.

3773 For external participant recruiting, we shared the survey on social media plat-
3774 forms and online-discussion forums relevant to software development. We adopted
3775 a non-probabilistic snowballing sampling where the participants, at the end of the
3776 survey, were encouraged to share the survey link to others using *AddThis*⁵. This
3777 resulted in 43 additional visits to the survey. Additionally, snowballing sampling was
3778 encouraged within members of our research group who shared the survey with an
3779 additional 21 participants. However, while there were a total of 86 respondents, only
3780 51 finished the survey, leaving 35 participants with partially completed responses.
3781 Our final response rate was therefore 59%, which is very close to median response
3782 rates of 60% [29] in information systems and 5% in software engineering [323].

3783 8.5.1.4 Analysing Response Data

3784 To analyse our response data, we used an adapted version of the SUS method to
3785 produce a score for each question's 5-point response. As per Brooke's methodology,
3786 we mapped the responses from their ordinal scale of 1–5 to 0–4, and subtracted that
3787 value by 1 for positive questions and subtracted the value from 5 for the negative
3788 questions [61]. Unlike Brooke's method, we averaged each response for every
3789 question and divided by four (i.e., now a 4-point scale) to obtain scores for each
3790 category. This is presented in Appendix C.1 under the 'in-practice score' (IPS) for
3791 each category.

3792 Demographics for our survey were consistent in terms of the experience levels of
3793 developers who responded. Most were professional programmers with 75% report-
3794 ing between 1–10 years of work experience. A majority of our respondents (33%)
3795 reported to be in mid-tier roles. Most worked in either consulting or information
3796 technology services, reported at 17% for both.

**3797 8.5.2 Empirical application of the taxonomy against Computer Vision
3798 Services**

3799 Once our taxonomy had been developed, we performed an empirical application
3800 against three CVSs: Google Cloud Vision [411], Amazon Rekognition [386] and
3801 Azure Computer Vision [425]. Our selection criteria in choosing these particular
3802 services to analyse is based on the prominence of the service providers in industry
3803 and the ubiquity of their cloud platforms (Google Cloud, Amazon Web Services,
3804 and Microsoft Azure) in addition to being the top three adopted vendors used for

⁵<https://www.addthis.com> last accessed 7 January 2020

3805 cloud-based enterprise applications [294]. In addition, we had conducted extensive
3806 investigation into the services' non-deterministic runtime behaviour and evolution
3807 profile in prior work [87] and have also identified developers' complaints about their
3808 incomplete documentation in a prior mining study on Stack Overflow [90].

3809 We began with an exploratory analysis of the presence of each dimension and
3810 its categories. Appendix C.2 displays all sources of documentation used; although
3811 we initially started on the respective services homepages [386, 411, 425], this search
3812 was expanded to other webpages hyperlinked. For each category, we listed the
3813 documentation's presence as either fully present, partially present or not present
3814 at all. This is shown in Appendix C.1 with the indication of (half-)filled circles or
3815 circle outlines for Google Cloud Vision (abbreviated to GCV), Amazon Rekognition
3816 (abbreviated to AWS), and Azure Computer Vision (abbreviated to ACV). Notes were
3817 taken for each webpage justifying the presence, and exact sources of documentation
3818 were listed when (partially) present. PDFs of each webpage were downloaded
3819 between 14–18 March 2019 for analysis.

3820 Once our analysis was completed and results from the survey finalised, we then
3821 calculated *weighted* ILS and IPS values for each dimension's category. This was done
3822 by multiplying the ILS and IPS values for each category (listed in Appendix C.1) by
3823 either 0, 0.5 or 1 for categories not present, partially present, or present (respectively)
3824 in each service. The 'maximum' ILS and IPS values indicate the highest possible
3825 score a service can be ranked as though *all* categories are present. Tables 8.3 and 8.4
3826 show the sum of weights for each category in its respective dimension, in addition to
3827 the maximum possible score. Again, we use the same abbreviations for each service
3828 as per Appendix C.1. The scores are normalised into percentages for comparative
3829 purposes as a ratio of the score over all dimensions for a particular service to
3830 the maximum possible score. For comparative purposes, these are illustrated in
3831 Figure 8.6.

3832 **8.6 Taxonomy Analysis**

3833 In this section, we analyse investigating the taxonomy from two perspectives. Firstly,
3834 we describe the ILS values, being an interpretation of the number of papers that
3835 conclude the recommendations in each category and dimension, and the weighted ILS
3836 scores, being an application of the taxonomy specifically to CVSs. Secondly, we look
3837 at the results from our survey and their respective IPS values, being an interpretation
3838 of how well developers agree with these recommendations, and the weighted IPS
3839 scores, being the application of how application developers would agree with the
3840 documentation of the CVSs. We then contrast the difference between what literature
3841 recommends and how well developers agree with these recommendations.

3842 **8.6.1 In-Literature Scores for Taxonomy Categories**

3843 ILS values indicate the proportion of papers that recommend categories within our
3844 taxonomy of all 21 studies. The most highly recommended categories from our
3845 SMS fall under the Usage Description dimension. The majority (0.71) of studies

Table 8.3: Weighted ILS Scoring.

Dimension	GCV	AWS	ACV	Max
[A] Usage Description	2.64 (60%)	3.10 (71%)	3.02 (69%)	4.38
[B] Design Rationale	0.79 (55%)	0.95 (67%)	0.95 (67%)	1.43
[C] Domain Concepts	0.33 (54%)	0.14 (23%)	0.43 (69%)	0.62
[D] Support Artefacts	0.24 (31%)	0.52 (69%)	0.50 (66%)	0.76
[E] Documentation Presentation	1.05 (79%)	1.05 (79%)	0.98 (73%)	1.33
Total	5.05 (59%)	5.76 (68%)	5.88 (69%)	8.52

Table 8.4: Weighted IPS Scoring.

Dimension	GCV	AWS	ACV	Max
[A] Usage Description	4.84 (57%)	5.26 (62%)	5.62 (66%)	8.48
[B] Design Rationale	1.78 (43%)	2.51 (61%)	2.51 (61%)	4.13
[C] Domain Concepts	0.92 (51%)	0.55 (31%)	1.43 (80%)	1.80
[D] Support Artefacts	0.96 (28%)	1.80 (53%)	1.85 (55%)	3.36
[E] Documentation Presentation	2.66 (70%)	2.66 (70%)	2.38 (63%)	3.79
Total	11.17 (52%)	12.79 (59%)	13.79 (64%)	21.56

³⁸⁴⁶ advocate for code snippets as a necessary piece in the API documentation puzzle ³⁸⁴⁷ [A5]. While code snippets generally only reflect small portions of API functionality ³⁸⁴⁸ (limited to 15–30 LoC), this is complimented by step-by-step tutorials (0.57) that tie ³⁸⁴⁹ in multiple (disparate) components of API functionality, generally with some form ³⁸⁵⁰ of screenshots, demonstrating the development of a non-trivial application using the ³⁸⁵¹ API step-by-step [A6]. The third highest category scored was also under the Usage ³⁸⁵² Description dimension, being low-level reference documentation at 0.52 [A2]. These ³⁸⁵³ three categories were the only categories to be scored as majority categories (i.e., ³⁸⁵⁴ their scores were above 0.50). The fourth and fifth highest scores are an entry- ³⁸⁵⁵ level purpose/overview of the API (0.48) that gives a brief motivation as to why a ³⁸⁵⁶ developer should choose a particular API over another [B1] and consistency in the ³⁸⁵⁷ look and feel of the documentation throughout all of the API’s official documentation ³⁸⁵⁸ (0.43) [E6].

³⁸⁵⁹ 8.6.2 In-Practice Scores for Taxonomy Categories

³⁸⁶⁰ IPS values indicate the extent to which developers ‘agree’ with the statements made ³⁸⁶¹ in our survey, as calculated using the SUS technique [61]. These values are generally ³⁸⁶² greater than the ILS values, since they are ranked by all survey participants and are not ³⁸⁶³ a ratio of the 21 primary studies. Unlike ILS scores, 28 categories scored above 0.50. ³⁸⁶⁴ The highest dimension corroborates that of the ILS scores; within the top five ranked ³⁸⁶⁵ ILS scores, Usage Description categories feature four times. However, developers ³⁸⁶⁶ generally find limitations on what the APIs can and cannot provide the most useful, ³⁸⁶⁷ at 0.94, which falls under the Design Rationale dimension [B7]. Following this, the

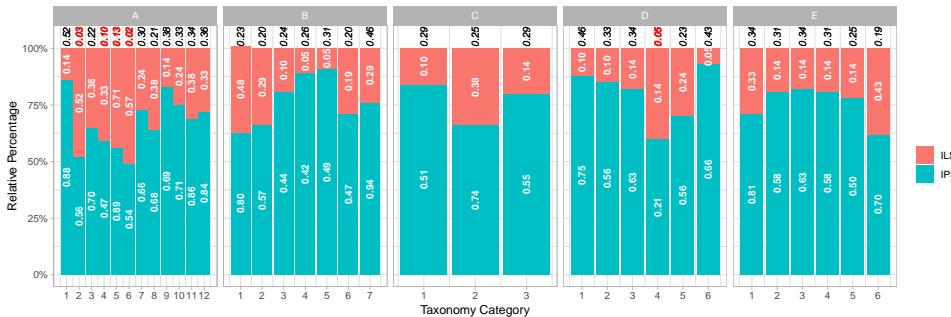


Figure 8.5: Comparison of ILS and IPS values for each category (grouped by dimensions) presented as a relative percentage.

Table 8.5: Labels assigned to ILS and IPS values.

Description	Lower Score Bound	Upper Score Bound
<i>Not Helpful</i>	0.00	0.24
<i>Slightly Unhelpful</i>	0.25	0.49
<i>Slightly Helpful</i>	0.50	0.74
<i>Very Helpful</i>	0.75	1.00

3868 code-snippets [A5] is highly ranked (as per the ILS values) with developers agreeing
 3869 that code-snippets should be included in most API documentation. Quick-start
 3870 guides [A1] are the next most-useful category that developers advocate for, reported
 3871 at 0.88. Following this, the instructions on how to install the API or begin using the
 3872 API, its release cycle, and frequently it is updated [A11] is also important, ranking
 3873 fourth at 0.86. Lastly, error definitions describing how developers can address
 3874 problems [A12] were scored at 0.84.

3875 8.6.3 Contrasting In-Literature to In-Practice Scores

3876 Figure 8.5 highlights the relative percentage of each ILS and IPS value for all
 3877 subcategories, thereby indicating the relative agreement between the two. In this
 3878 graph, an ILS and/or IPS core approaching a relative percentage of 50% indicates
 3879 equal agreement whereby both developer's and literary references share a similar
 3880 distribution of recommendation agreement. Italicised labels above each column
 3881 indicates the standard deviation between the ILS and IPS values, where red labels
 3882 indicated a standard deviation less than 0.15 (i.e., developers and literature agree to
 3883 the values to a similar extent).

3884 Where the standard deviation between ILS and IPS values is less than 0.015
 3885 (as indicated by red labels above each column in Figure 8.5), then there is strong
 3886 alignment between both scores. However, of all 34 categories, only five cases of this
 3887 occur. Developers agree to the academic works that make the recommendations *to*
 3888 *the same relative proportion* as per the labels assigned in Table 8.5:

- 3889 • Having email addresses or phone numbers listed within an API is generally

3890 not helpful at all [D4],

- 3891 • Introspecting the source code comments of an API is only somewhat helpful
[A4],
- 3892 • Low-level reference documentation with all objects and methods (etc.) docu-
3893 mented is slightly helpful [A2],
- 3894 • Following step-by-step tutorials are also slightly helpful [A6],
- 3895 • Code snippets are the most helpful [A5].

3896 The remaining categories in the dimension do not share strong association be-
3897 between both developer opinions and the number of papers producing recomme-
3898 dations. Due to the disparity between these ILS and IPS values, we do not report on
3899 their utility.

3900 8.6.4 Triangulating ILS and IPS with Computer Vision

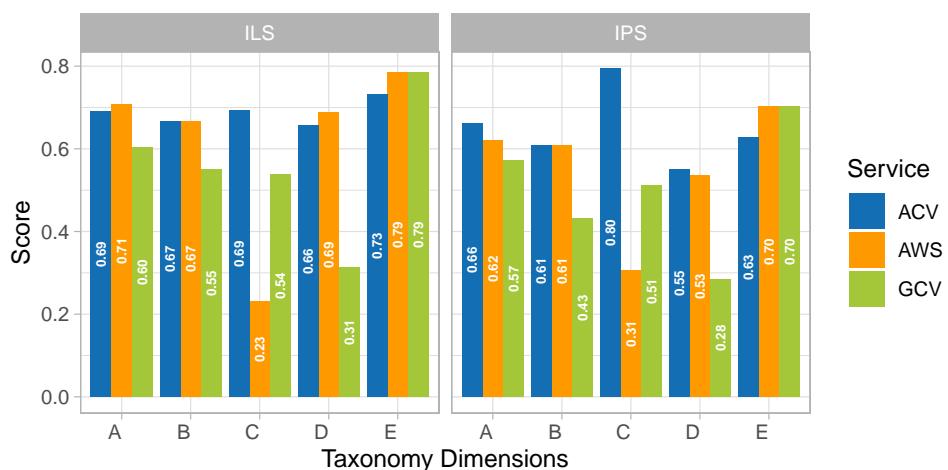


Figure 8.6: Comparison of the weighted ILS and IPS values for the three CVSs assessed.

3902 When applied in the context of CVSs, we see that Azure Computer Vision
3903 (ACV) and Amazon Rekognition (AWS) are better documented than Google Cloud
3904 Vision (GCV), particularly in Design Rationale and Usage Description. Figure 8.6
3905 highlights that Azure Computer Vision is especially well documented in Domain
3906 Concepts when measured using the weighted ILS and has the highest score of all
3907 services and dimensions. It is evident that Google Cloud Vision needs improved
3908 Design Rationale documentation and further Support Artefacts would be helpful.
3909 Generally speaking, Google Cloud Vision is less ‘complete’ than other services,
3910 except in Domain Concepts documentation and its Documentation Presentation.

3911 In the context of CVSs, IPS values share a similar distribution to ILS val-
3912 ues. Notably, in-practice, it seems developers prefer the documentation of Amazon
3913 Rekognition compared to the in-literature weighted scoring of Azure Computer Vi-
3914 sion (Figure 8.6). Except in the case of documenting Domain Concepts, Amazon

3915 Rekognition scores slightly higher than Azure Computer Vision except for the De-
3916 sign Rationale documentation where it is equal. Similar to the ILS scoring, Google
3917 Computer Vision has low compliance to the recommendations proposed, except in
3918 its Documentation Presentation.

3919 **8.6.5 Areas of Improvement for CVS Documentation**

3920 Triangulating the taxonomy developed from literary sources, the developer survey
3921 on this taxonomy to understand its efficacy in-practice, and applying the taxonomy to
3922 the CVS domain, we are able to assess the key areas of improvement in this domain.

3923 For this assessment, we select the ILS or IPS values for categories that are
3924 considered either somewhat or very helpful (i.e., a score greater than 0.50). We then
3925 match these against categories that are found to be partially or not present within
3926 each service. In total, we found 12 categories where improvements can be made
3927 across all dimensions except Documentation Presentation, detailed below .

3928 *8.6.5.1 Issues regarding Usage Description*

3929 **Quick-start guides [A1]:** Quick-start guides should provide a short tutorial that
3930 allows programmers to pick up the basics of an API in a programming language of
3931 their choice. For the services assessed, each offer various client SDKs (e.g., as Java
3932 or Python client libraries). Google Cloud Vision and Azure Computer Vision offer
3933 quick-start guides [414, 432] in which sets of articles target various SDKs or are
3934 client-agnostic with code snippets that can be changed to the client language/SDK
3935 of the developer's choice. Amazon Rekognition offers exercises in setting up the
3936 AWS SDK and using the command-line interface to interact with image analysis
3937 components [392], however this is client-agnostic nor does it provide details in how
3938 to get started with using the client SDKs.

 **Suggested improvement:** Ensure tutorials detail all client-libraries and how developers can produce a minimum working example using the service on their own computer using that client library. For each SDK offered, there should be details on how to install, authenticate and use a component using local data. For example, this may be as simple as using the service to determine if an image of a dog contains the label 'dog'.

3939 **Step-by-step tutorials [A6]:** Google Cloud Vision offers tutorials limited to one
3940 component. These do not sufficiently demonstrate how to combine *multiple compo-*
3941 *nents* of the API together and how developers should integrate it with a differ-
3942 ent platform, which a good step-by-step tutorial should detail. The official AWS
3943 Machine Learning blog [389] provides extensive tutorials (in some cases, with a
3944 suggested tutorial completion time of over an hour) that integrate multiple Amazon
3945 Rekognition components with other AWS components. Microsoft provide tutori-
3946 als [430, 435, 436] integrating multiple components within their service to mobile
3947 applications and the Azure platform.

☞ **Suggested improvement:** Ensure tutorials combine multiple components of the service together, are extensive, and require developers to spend a non-trivial amount of time to produce a basic application. For example, the tutorial may detail how to integrate the API into a smartphone application to achieve the following: (i) take a photo with the camera, (ii) detect if a person is within the image, (iii) analyse the visual features of the person.

3948 **Downloadable production-ready applications [A7]:** Microsoft provide a down-
3949 loadable application [434] that explores many components of the Azure Computer
3950 Vision API. The application is thoroughly documented with and also provides guid-
3951 ance on how to structure the architecture design of the program. While Rekognition
3952 and Google Cloud Vision also provide downloadable source code, they are largely
3953 under-documented, do not combine multiple components of the API together, and
3954 only use god-classes to handle all requests to the API [393, 416].

☞ **Suggested improvement:** Downloadable source code should be thoroughly docu-
mented, and should avoid the use of god-classes that demonstrate a single piece of the
service's functionality. Ideally, the architecture of a production-ready application should
be demonstrated to developers.

3955 **Understanding best-practices [A8]:** Google Cloud provides best-practices for its
3956 platform in both general and enterprise contexts [408, 417], but there is little advice
3957 provided to guide developers on how best to use Google Cloud Vision. Microsoft
3958 provides guidance on improving results of custom vision classifiers [431], but no
3959 further details on non-custom vision classifiers are found. We found the most detailed
3960 best-practices to be provided by Amazon Rekognition [391], which outlines more
3961 detailed strategies such as reducing data transfer by storing and referencing images
3962 on S3 Buckets or the attributes images should have in various scenarios (e.g., the
3963 angles of a person's face in facial recognition).

☞ **Suggested improvement:** Document best-practices for all major components of the
CVS. Guide developers on the types of input data that produce the best results, advisable
minimum image sizes and recommended file types, and suggest ways to overcome
limitations that improve usage and cost efficiency. Provide guidance in more than one
use case; give a range of scenarios that demonstrate different best practices for different
domains.

3964 **Exhaustive lists of all major API components [A9]:** Amazon provides a two-fold
3965 feature list that describes both the key features of Rekognition at a high-level [390]
3966 as well as a detailed, technical breakdown of each API operation provided within the
3967 service [388]. Microsoft also provide a list of high-level features that Azure Com-
3968 puter Vision can analyse [437] which provides hyperlinks to detailed descriptions of
3969 each feature. Google's Cloud Vision API provides a partial breakdown of the types
3970 of services provided, however this list is not fully complete, nor are there hyperlinks
3971 to more detailed descriptions of each of the features [418].

 **Suggested improvement:** Document key features that the computer vision classifier can perform at a high level. This should be easy to find from the service's landing page. Each feature should be described with reference to more detailed descriptions of the feature's exact API endpoint and required inputs, outputs and possible errors.

3972 **Minimum system requirements and dependencies [A10]:** Although there is no
3973 dedicated webpage for this on any of the services investigated, there are listed
3974 dependencies for the client libraries in Google's and Azure's quick-start guides [414,
3975 428]. These may be embedded within the quick-start guide as developers are likely
3976 to encounter dependency issues when they first start using the API. We found it a
3977 challenge to discover similar documentation this in Amazon's documentation.

 **Suggested improvement:** Any system requirements and dependency issues should be well-highlighted within the documentation's quick-start guide; developers are likely to encounter these issues within the early stages of using an API, and it is highly relevant to provide solutions to these issues within the quick-starts.

3978 **Installation and release cycle notes [A11]:** It is imperative that developers know
3979 what has changed between releases and how frequently the releases are exported.
3980 We found release notes for Amazon Computer Vision, although they are only major
3981 releases and have not been updated since 2017 [387] which does not account for
3982 evolution in the service's responses [87]. Google's and Microsoft's release notes are
3983 generally more frequently updated, therefore developers can get a sense of its release
3984 frequency [415, 433]. However, there are evolution issues that are not addressed.
3985 Installation instructions are detailed within Rekognition's developer guide, outlining
3986 how to sign up for an account, and install the AWS command-line interface [395].

 **Suggested improvement:** Ensure release notes detail label evolution, including any new additional labels that may have been introduced within the service. Transparency around the changes made to the service should go beyond new features: document potential changes that may influence maintenance of a system using the CVS so that developers are aware of potential side-effects of upgrading to a newer release.

3987 8.6.5.2 Issues regarding Design Rationale

3988 **Limitations of the API [B7]:** The most detailed limitations documented were
3989 found on Rekognition's dedicated limitations page [394] that outlines functional
3990 limitations such as the maximum number of faces or words that can be detected
3991 in an image, the size requirements of images, and file type information. For the
3992 other services, functional limitations are generally found within each endpoint's API
3993 documentation, instead of within a dedicated page.

 **Suggested improvement:** Document all functional limitations in a dedicated page that outline the maximum and minimum input requirements the classifier can handle. Documentation of the types of labels the service can provide is also desired.

3994 8.6.5.3 *Issues regarding Domain Concepts*

3995 **Conceptual understanding of the API [C1]:** Azure Computer Vision provides
3996 ‘concept’ pages describing the high-level concepts behind computer vision and where
3997 these functions are implemented within the APIs (e.g., [429]). We were unable to
3998 find similar conceptual documentation for the other services assessed.

⌚ Suggested improvement: Document the concepts behind computer vision; differentiate between foundational concepts such as object localisation, object recognition, facial localisation and facial analysis such that developers are able to make the distinction between them. Relate these concepts back to the API and provide references to where the APIs implement these concepts.

3999 **Definitions of domain-specific terminology [C2]:** Terminologies relevant to ma-
4000 chine learning concepts powering these CVSs are well detailed within Google’s
4001 machine learning glossary [412], however few examples matching computer vision
4002 are immediately relevant. While this page is linked from the original Google Cloud
4003 Vision documentation, it may be too technical for application developers to grasp. A
4004 slightly better example of this is [437], where developers can understand computer
4005 vision terms in lay terms.

⌚ Suggested improvement: Current CVSs use a myriad of terminologies to refer to the same conceptual feature; for example, while Microsoft refers to object recognition as ‘image tagging’, Google refers to this as ‘label detection’. If a consolidation of terms is not possible, then CVSs should provide a glossary that provides synonyms for these terminologies so that developers can easily move between service providers without needing to relink terms back to concepts.

4006 8.6.5.4 *Issues regarding Support Artefacts*

4007 **Troubleshooting suggestions [D2]:** The only troubleshooting tips found in our
4008 analysis were in Rekognition’s video service [396]. Further detailed instances of
4009 these troubleshooting tips could be expanded to non-video issues. For instance,
4010 if developers upload ‘noisy’ images, how can they inform the system of a specific
4011 ontology to use or to focus on parts of the foreground or background of the image?
4012 These are suggestions which we have proposed in prior work [87] that do not seem
4013 to be documented.

⌚ Suggested improvement: Ensure troubleshooting tips provide advice for testing against different types of valid input images.

4014 **Diagrammatic overview of the API [D3]:** None of the CVSs provide any overview
4015 of the API in terms of the features and processing steps on how they should be used.
4016 For instance, pre-processing and post-processing of input and response data should
4017 be considered and an understanding of how this fits into the ‘flow’ of an application
4018 highlighted. Moreover, no UML diagrams could be found.

☞ **Suggested improvement:** Provide diagrams illustrating the service within context of use, such as how it can be integrated with other service features or how a specific API endpoint may be used within a client application. Consider integrating interactive UML diagrams so that developers can easily explore various aspects of the documentation in a visual perspective.

4019 8.7 Threats to Validity

4020 8.7.1 Internal Validity

4021 Threats to *internal validity* represent internal factors of our study which affect
4022 concluded results. Kitchenham and Charters' guidelines on producing systematic
4023 reviews [191] suggest that researchers conducting reviews should discuss the review
4024 protocol, inclusion decisions, data extraction with a third party. Within this study,
4025 we discussed our protocols with other researchers within our research group and
4026 utilised test-retest reliability. Further assessments into reliability would involve an
4027 assessment of the review and extraction processes, which can be investigated using
4028 inter-rater reliability measures. Guidelines suggested by Garousi and Felderer [129]
4029 describe methods for independent analysis and conflict resolution could help resolve
4030 this.

4031 As stated in Section 8.3.2, we utilised a systematic software engineering tax-
4032 onomy development method by Usman et al. [351]. Two additional taxonomy
4033 validation approaches proposed by Usman et al. were not considered in our work:
4034 benchmarking and orthogonality demonstration. To our knowledge, there are no
4035 other studies that classify existing API knowledge studies into a structured taxon-
4036 omy, and therefore we are unable to benchmark our taxonomy against others. We
4037 would encourage the research community to conduct a replication of our work and
4038 investigate whether our taxonomy classification approaches are replicable to ensure
4039 that categories are reliable and the dimensions fit the objectives of the taxonomy.
4040 Moreover, we did not investigate orthogonality demonstration as our primary goals
4041 for this work were to investigate the efficacy of the taxonomy by practitioners and
4042 in-practice, with reference to our wider research area of intelligent CVSs. Therefore,
4043 we solely adopted the utility demonstration approach in two detailed experiments
4044 (Sections 8.5 and 8.6) to analyse the efficacy of our taxonomy and identify potential
4045 improvements for these services' API documentation.

4046 8.7.2 External Validity

4047 Threats to *external validity* concern the generalisation of our observations. Our
4048 systematic mapping study has used a broad range of sources however not all papers
4049 contributing to API documentation may have been found or captured within the
4050 taxonomy. While we attempted to include as many papers as we could find in our
4051 study, some papers may have been filtered out due to our exclusion criteria. For
4052 example, there are studies we found that were excluded as they were not written in
4053 English, and these excluding factors may alter our conclusions, introducing conflict-

4054 ing recommendations. However, given the consistency of these trends within the
4055 studies that were sourced, we consider this a low likelihood.

4056 Documentation of web APIs are non-static, and may evolve using contributions
4057 from both official sources and the developer community (e.g., via GitHub). We
4058 downloaded the three service’s API documentation in March of 2019—it is highly
4059 likely that new documentation may have been added since or modified since publi-
4060 cation. A recommendation to mitigate this would be to re-evaluate this study once
4061 intelligent CVSs have matured and become even more mainstream in developer
4062 communities.

4063 We also adopt research conducted in the field of questionnaire design, such as
4064 ensuring all scales are worded with labels [202] and have used a summatting rating
4065 scale [328] to address a specific topic of interest if people are to make mistakes in
4066 their response or answer in different ways at different times. This approach was
4067 also extended using the SUS methodology, in which positive and negative items
4068 were used—as multiple studies have shown [62, 307], this approach helps reduce
4069 poor-quality responses by minimising extreme responses and acquiescence biases.

4070 8.7.3 Construct Validity

4071 Threats to *construct validity* relates to the degree by which the data extrapolated
4072 in this study sufficiently measures its intended goals. Automatic searching was
4073 conducted in the SMS by choice of three popular databases (see Section 8.3.1).
4074 As a consequence of selecting multiple databases, duplicates were returned. This
4075 was mitigated by manually curating out all duplicate results from the set of studies
4076 returned. Additionally, we acknowledge that the lack manual searching of papers
4077 within particular venues may be an additional threat due to the misalignment of
4078 search query keywords to intended papers of inclusion. Thus, our conclusions are
4079 only applicable to the information we were able to extract and summarise, given the
4080 primary sources selected.

4081 While we have investigated the application of this taxonomy using a user study
4082 (Section 8.5.1), we would like to explore an observational study of developers
4083 to assess how improved and non-improved API documentation impacts developer
4084 productivity. The outcome of this work can help design a follow-up experiment,
4085 consisting of a comparative controlled study [313] that capture firsthand behaviours
4086 and interactions toward how software engineers approach using a CVS with and
4087 without our taxonomy applied. This can be achieved by providing ‘mock’ improved
4088 documentation with the suggested improvements included in this work. Such an ex-
4089 periment could recruit a sample of developers of varying experience (from beginner
4090 programmer to principal engineer) to complete a certain number of tasks under an
4091 observational, comparative controlled study, half of which will (a) develop using
4092 the improved ‘mock’ documentation, and the other half will (b) develop with the
4093 *as-is/existing* documentation. From this, we can compare if the framework makes
4094 improvements by capturing metrics and recording the observational sessions for
4095 qualitative analysis. Visual modelling can be adopted to analyse the qualitative data
4096 using matrices [98], maps and networks [310] as these help illustrate any causal, tem-

4097 poral or contextual relationships that may exist to map out the developer’s mindset
4098 and difference in approaching the two sets of designs of the same tasks.

4099 8.8 Conclusions & Future Work

4100 A good API document should facilitate a developer’s productivity, and is therefore
4101 associated to the quality of software produced; improving the quality of the docu-
4102 mentation of third-party APIs improves the quality of dependent software. However,
4103 there does not yet exist a consolidated taxonomy of key recommendations proposed
4104 by literature, and—more importantly—it is useful to know if what developers need
4105 *in-practice* differs to what documentation artefacts are anticipated by literature.
4106 Moreover, there has been little work on mapping the research produced in this space
4107 against the techniques used to arrive at the recommendations.

4108 This study priorities which aspects of API documentation knowledge is both (i)
4109 suggested by literature, and (ii) is demanded *most* by developers. We conduct a
4110 SMS from a pool of 4,501 studies and identify 21 seminal studies. From this, we
4111 synthesise a taxonomy of the various documentation aspects that should improve
4112 API documentation quality. Furthermore, we also capture the most commonly used
4113 analysis techniques used in the academic literature. We then validate our taxonomy
4114 against developers to assess its efficacy with practitioners, and conduct a heuristic
4115 evaluation against to three popular CVSs. We offer 12 detailed suggested improve-
4116 ments where these services currently have weaknesses, and where specifically they
4117 may be able to improve their documentation.

4118 Future extensions of our work may involve a restricted systematic literature
4119 review in API documentation artefacts, and many suggestions are further detailed
4120 in Section 8.7. Further, a review into the techniques of these primary studies may
4121 extend the mapping we conducted in this work, by evaluating the effectiveness of
4122 the various approaches used in each study and assessing these against the proposed
4123 conclusions of each study.

4124 The findings of our work provides a solid baseline for improving the documen-
4125 tation of non-deterministic software, such as CVSs. While our aim is to eventually
4126 improve the quality of API documentation, the ultimate goal is to improve the soft-
4127 ware engineer’s experience of non-deterministic IWSs. We hope the guidelines from
4128 this extensive study help both software developers and API providers alike by using
4129 our taxonomy as a go-to checklist for what should be considered in documenting any
4130 API.

CHAPTER 9

4131

4132

4133 Using a Facade Pattern to combine Computer Vision Services[†]

4134

4135 **Abstract** Intelligent computer vision services, such as Google Cloud Vision or Amazon
4136 Rekognition, are becoming evermore pervasive and easily accessible to developers to build
4137 applications. Because of the stochastic nature that ML entails and disparate datasets used in
4138 their training, the outputs from different computer vision services varies with time, resulting
4139 in low reliability—for some cases—when compared against each other. Merging multiple
4140 unreliable API responses from multiple vendors may increase the reliability of the overall
4141 response, and thus the reliability of the intelligent end-product. We introduce a novel
4142 methodology—inspired by the proportional representation used in electoral systems—to
4143 merge outputs of different intelligent computer vision API provided by multiple vendors.
4144 Experiments show that our method outperforms both naive merge methods and traditional
4145 proportional representation methods by 0.015 F-measure.

4146 9.1 Introduction

4147 With the introduction of intelligent web services (IWSs) that make machine learning
4148 (ML) more accessible to developers [291, 358], we have seen a large growth of
4149 intelligent applications dependent on such services [65, 134]. For example, consider
4150 the advances made in computer vision, where objects are localised within an image
4151 and labelled with associated categories. Cloud-based computer vision services
4152 (CVSs)—e.g., [386, 399, 407, 411, 420, 421, 425, 473]—are a subset of IWSs.
4153 They utilise ML techniques to achieve image recognition via a remote black-box
4154 approach, thereby reducing the overhead for application developers to understand
4155 how to implement intelligent systems from scratch. Furthermore, as the processing

[†]This chapter is originally based on T. Ohtake, A. Cummaudo, M. Abdelrazek, R. Vasa, and J. Grundy, “Merging intelligent API responses using a proportional representation approach,” in *Proceedings of the 19th International Conference on Web Engineering*. Daejeon, Republic of Korea: Springer, June 2019. DOI 10.1007/978-3-030-19274-7_28. ISBN 978-3-03-019273-0. ISSN 1611-3349 pp. 391–406. Terminology has been updated to fit this thesis.

4156 and training of the machine-learnt algorithms is offloaded to the cloud, developers
4157 simply send RESTful API requests to do the recognition. There are, however, inherit
4158 differences and drawbacks between traditional web services and IWSs, which we
4159 describe with the motivating scenario below.

4160 9.1.1 Motivating Scenario: Intelligent vs Traditional Web Services

4161 An application developer, Tom, wishes to develop a social media Android and iOS
4162 app that catalogues photos of him and his friends, common objects in the photo,
4163 and generates brief descriptions in the photo (e.g., all photos with his husky dog,
4164 all photos on a sunny day etc.). Tom comes from a typical software engineering
4165 background with little knowledge of computer vision and its underlying concepts.
4166 He knows that intelligent computer vision web APIs are far more accessible than
4167 building a computer vision engine from scratch, and opts for building his app using
4168 these cloud services instead.

4169 Based on his experiences using similar cloud services, Tom would expect consistency
4170 of the results from the same API and different APIs that provide the same (or
4171 similar) functionality. As an analogy, when Tom writes the Java substring method
4172 "doggy".substring(0, 2), he expects it to be the same result as the Swift equivalent
4173 "doggy".prefix(3). Each and every time he interacts with the substring
4174 method using either API, he gets "dog" as the response. This is because Tom is
4175 used to deterministic, rule-driven APIs that drive the implementation behind the
4176 substring method.

4177 Tom's deterministic mindset results in three key differentials between a traditional
4178 web services and an IWS:

4179 **(1) Given similar input, results differ between similar IWSs.** When Tom
4180 interacts with the API of an IWS, he is not aware that each API provider trains
4181 their own, unique ML model, both with disparate methods and datasets. These
4182 IWSs are, therefore, nondeterministic and data-driven; input images—even
4183 if they contain the same conceptual objects—often output different results.
4184 Contrast this to the substring method of traditional APIs; regardless of what
4185 programming language or string library is used, the same response is expected
4186 by developers.

4187 **(2) Intelligent responses are not certain.** When Tom interprets the response
4188 object of an IWS, he finds that there is a ‘confidence’ value or ‘score’. This
4189 is because the ML models that power IWSs are inherently probabilistic and
4190 stochastic; any insight they produce is purely statistical and associational [274].
4191 Unlike the substring example, where the rule-driven implementation provides
4192 certainty to the results, this is not guaranteed for IWSs. For example, a picture
4193 of a husky breed of dog is misclassified as a wolf. This could be due to
4194 adversarial examples [337] that ‘trick’ the model into misclassifying images
4195 when they are fully decipherable to humans. It is well-studied that such
4196 adversarial examples exist in the real world unintentionally [113, 203, 277].

4197 **(3) Intelligent APIs evolve over time.** Tom may find that responses to processing
4198 an image may change over time; the labels he processes in testing may evolve

4199 and therefore differ to when in production. In traditional web services, evo-
4200 lution in responses is slower, generally well-communicated, and usually rare
4201 (Tom would always expect "dog" to be returned in the substring example).
4202 This has many implications on software systems that depend on these APIs,
4203 such as confidence in the output and portability of the solution. Currently, if
4204 Tom switches from one API provider to another, or if he doesn't regularly test
4205 his app in production, he may begin to see a very different set of labels and
4206 confidence levels.

4207 9.1.2 Research Motivation

4208 These drawbacks bring difficulties to the intended API users like Tom. We identify a
4209 gap in the software engineering literature regarding such drawbacks, including: lack
4210 of best practices in using IWSs; assessing and improving the reliability of APIs for
4211 their use in end-products; evaluating which API is suitable for different developer
4212 and application needs; and how to mitigate risk associated with these APIs. We
4213 focus on improving reliability of CVSs for use in end-products. The key research
4214 questions in this paper are:

4215 **RQ1:** Is it possible to improve reliability by merging multiple CVS results?

4216 **RQ2:** Are there better algorithms for merging these results than currently in
4217 use?

4218 Previous attempts at overcoming low reliability include triple-modular redun-
4219 dancy [222]. This method uses three modules and decides output using majority
4220 rule. However, in CVSs, it is difficult to apply majority rule: these APIs respond with
4221 a list of labels and corresponding scores. Moreover, disparate APIs ordinarily output
4222 different results. These differences make it hard to apply majority rule because the
4223 type of outputs are complex and disparate APIs output different results for the same
4224 input. Merging search results is another technique to improve reliability [322]. It
4225 normalises scores of different databases using a centralised sample database. Nor-
4226 malising scores makes it possible to merge search results into a single ranked list.
4227 However, search responses are disjoint, whereas they are not in the context of most
4228 CVSs.

4229 In this paper, we introduce a novel method to merge responses of CVSs, using
4230 image recognition APIs endpoints as our motivating example. Section 9.2 describes
4231 naive merging methods and requirements. Section 9.3 gives insights into the struc-
4232 ture of labels. Section 9.4 introduces our method of merging computer vision labels.
4233 Section 9.5 compares precision and recall for each method. Section 9.6 presents
4234 conclusions and future work.

4235 9.2 Merging API Responses

4236 Image recognition APIs have similar interfaces: they receive a single input (image)
4237 and respond with a list of labels and associated confidence scores. Similarly, other
4238 supervised-AI-based APIs do the same (e.g., detecting emotions from text and
4239 natural language processing [422, 474]). It is difficult to apply majority rule on such

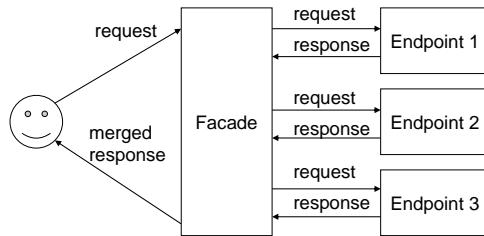


Figure 9.1: The user sends a request to the facade; this request is propagated to the relevant APIs. Responses are merged by the facade and returned back to the user.

4240 disparate, complex outputs. While the outputs by *multiple* AI-based API endpoints
 4241 is different and complex, the general format of the output is the same: a list of labels
 4242 and associated scores.

4243 9.2.1 API Facade Pattern

4244 To merge responses from multiple APIs, we introduce the notion of an API facade.
 4245 It is similar to a metasearch engine, but differs in their external endpoints. The
 4246 facade accepts the input from one API endpoint (the facade endpoint), propagates
 4247 that input to all user-registered concrete (external) API endpoints simultaneously,
 4248 then ‘merges’ outputs from these concrete endpoints before sending this merged
 4249 response to the API user. We demonstrate this process in Figure 9.1.

4250 Although the model introduces more time and cost overhead, both can be mitigated
 4251 by caching results. On the other hand, the facade pattern provides the following
 4252 benefits:

- 4253 • **Easy to modify:** It requires only small modifications to applications, e.g.,
 4254 changing each concrete endpoint URL.
- 4255 • **Easy to customise:** It merges results from disparate and concrete APIs ac-
 4256 cording to the user’s preference.
- 4257 • **Improves reliability:** It enhances reliability of the overall returned result by
 4258 merging results from different endpoints.

4259 9.2.2 Merge Operations

4260 The API facade is applicable to many use cases. However, this paper focuses on
 4261 APIs that output a list of labels and scores, as is the case for CVSs. Merge operations
 4262 involve the mapping of multiple lists and associated scores, produced by multiple
 4263 APIs, to just one list. For instance, a CVS receives a bowl of fruit as the input image
 4264 and outputs the following:

4265 `[[‘apple’, 0.9], [‘banana’, 0.8]]`

4266 where the first item is the label and the second item is the score. Similarly, another
 4267 computer vision API outputs the following for the same image:

4268 `[[‘apple’, 0.7], [‘cherry’, 0.8]].`

4269 Merge operations can, therefore, merge these two responses into just one response.
4270 Naive ways of merging results could make use of *max*, *min*, and *average* operations
4271 on the confidence scores. For example, *max* merges results to:

4272 $\text{[['apple', 0.9], ['banana', 0.8], ['cherry', 0.8]]};$

4273 *min* merges results to:

4274 $\text{[['apple', 0.7]]};$

4275 and *average* merges results to:

4276 $\text{[['apple', 0.8], ['banana', 0.4], ['cherry', 0.4]]}.$

4277 However, as the object’s labels in each result are natural language, the operations
4278 do not exploit the label’s semantics when conducting label merging. To improve
4279 the quality of the merged results, we consider the ontologies of these labels, as we
4280 describe below.

4281 9.2.3 Merging Operators for Labels

4282 Merge operations on labels are *n*-ary operations that map R^n to R , where $R_i =$
4283 $\{(l_{ij}, s_{ij})\}$ is a response from endpoint i and contains pairs of labels (l_{ij}) and scores
4284 (s_{ij}). Merge operations on labels have the following properties:

- 4285 • *identity* defines that merging a single response should output same response
4286 (i.e., $R = \text{merge}(R)$ is always true);
- 4287 • *commutativity* defines that the order of operands should not change the result
4288 (i.e., $\text{merge}(R_1, R_2) = \text{merge}(R_2, R_1)$ is always true);
- 4289 • *reflexivity* defines that merging multiple same responses should output same
4290 response (i.e., $R = \text{merge}(R, R)$ is always true); and,
- 4291 • *additivity* defines that, for a specific label, the merged response should have
4292 higher or equal score for the label if a concrete endpoint has a higher score.
4293 Let $R = \text{merge}(R_1, R_2)$ and $R' = \text{merge}(R'_1, R_2)$ be merged responses. R_1 and
4294 R'_1 are same, except R'_1 has a higher score for label l_x than R_1 . The additive
4295 score property requires that R' score for l_x should be greater than or equal to
4296 R score for l_x .

4297 The *max*, *min*, and *average* operations in Section 9.2.2 follow each of these rules
4298 as all operations calculate the score by applying these operations on each score.

4299 9.3 Graph of Labels

4300 CSVs typically return lists of labels and their associated scores. In most cases, the
4301 label can be a singular word (e.g., ‘husky’) or multiple words (e.g., ‘dog breed’).
4302 Lexical databases, such as WordNet [242], can therefore be used to describe the
4303 ontology behind these labels’ meanings. Figure 9.2 is an example of a graph of

Table 9.1: Statistics for the number of labels, on average, per service identified.

Endpoint	Average number of labels	Has synset	No synset
Amazon Rekognition	11.42 ± 7.52	10.74 ± 7.10 (94.0%)	0.66 ± 0.87
Google Cloud Vision	8.77 ± 2.15	6.36 ± 2.22 (72.5%)	2.41 ± 1.93
Azure Computer Vision	5.39 ± 3.29	5.26 ± 3.32 (97.6%)	0.14 ± 0.37

4304 labels and synsets. A synset is a grouped set of synonyms for a word. In this image,
4305 we consider two fictional endpoints, endpoints 1–2. We label red nodes as labels
4306 from endpoint 1, yellow nodes as labels from endpoint 2, and blue nodes as synsets
4307 for the associated labels from both endpoints. As actual graphs are usually more
4308 complex, Figure 9.2 is a simplified graph to illustrate the usage of associating labels
4309 from two concrete sources to synsets.

4310 9.3.1 Labels and synsets

4311 The number of labels depends on input images and concrete API endpoints used.
4312 Table 9.1 and Figure 9.3 show how many labels are returned, on average per image,
4313 from Google Cloud Vision [411], Amazon Rekognition [386] and Azure Computer
4314 Vision [425] image recognition APIs. These statistics were calculated using 1,000
4315 images from Open Images Dataset V4 [413] Image-Level Labels set.

4316 Labels from Amazon and Microsoft tend to have corresponding synsets, and
4317 therefore these endpoints return common words that are found in WordNet. On the
4318 other hand, Google’s labels have less corresponding synsets: for example, labels
4319 without corresponding synsets are car models and dog breeds.¹

4320 9.3.2 Connected Components

4321 A connected component (CC) is a subgraph in which there are paths between any
4322 two nodes. In graphs of labels and synsets, CCs are clusters of labels and synsets
4323 with similar semantic meaning. For instance, there are two CCs in Figure 9.2. CC 1
4324 in Figure 9.2 has ‘beverage’, ‘dessert’, ‘chocolate’, ‘hot chocolate’,
4325 ‘drink’, and ‘food’ labels from the red first endpoint and ‘coffee’, ‘hot
4326 chocolate’, ‘drink’, ‘caffeine’, and ‘tea’ labels from the yellow second
4327 endpoint. Therefore, these labels are related to ‘drink’. On the other hand, CC 2
4328 in Figure 9.2 has ‘cup’ and ‘coffee cup’ labels from the first red endpoint and
4329 ‘cup’, ‘coffee cup’, and ‘tableware’ labels from the yellow second endpoint.
4330 These labels are, therefore, related to ‘cup’.

4331 Figure 9.4 shows a distribution of number of CCs for the 1,000-image label
4332 detections on Amazon Rekognition, Google Cloud Vision, and Azure Computer
4333 Vision APIs. The average number of CCs is 9.36 ± 3.49 . The smaller number of
4334 CCs means that most of labels have similar meanings, while a larger value means
4335 that the labels are more disparate.

¹We noticed from our upload of 1,000 images that Google tries to identify objects in greater detail.

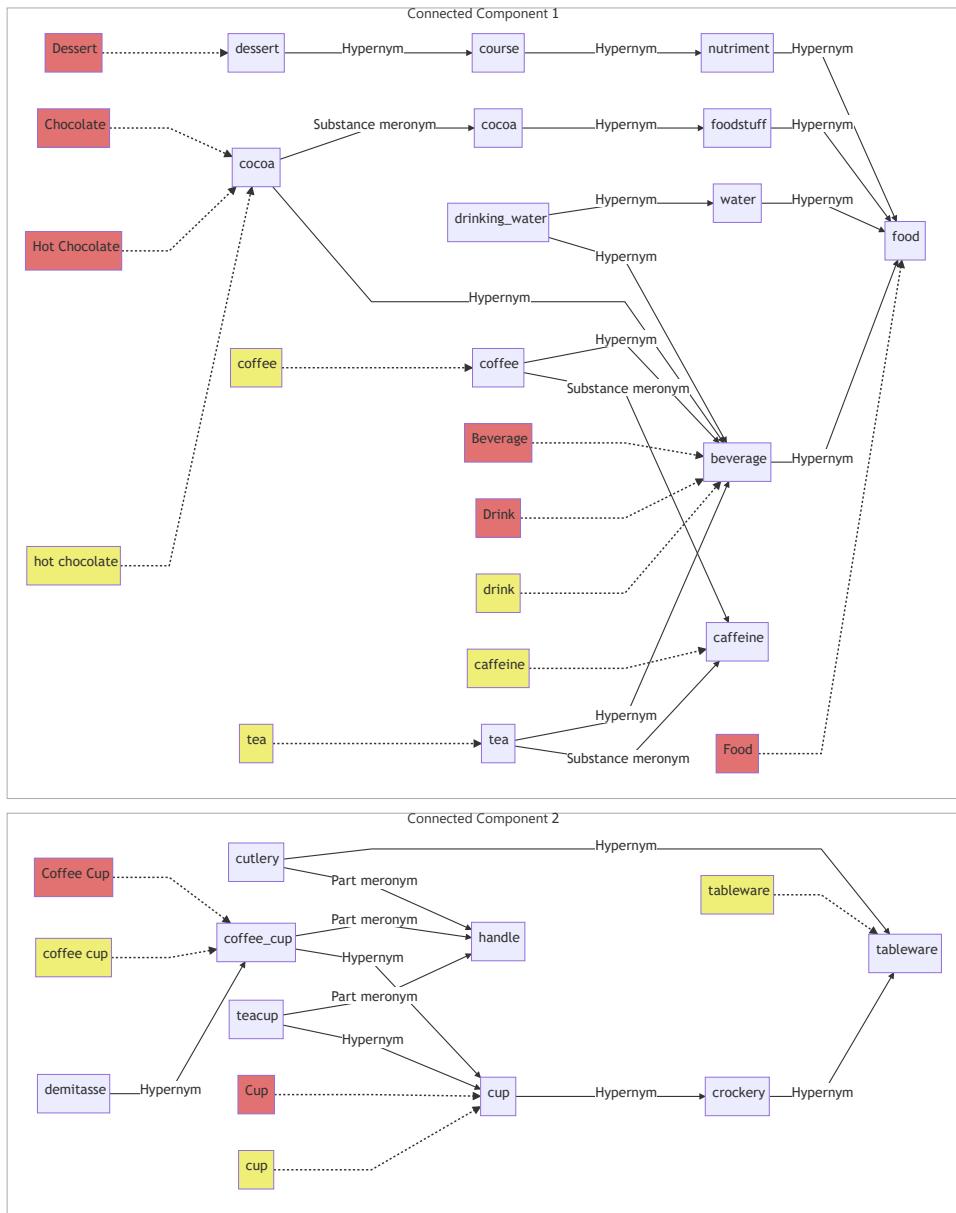


Figure 9.2: Graph of labels from two concrete endpoints (red and yellow) and their associated synsets related to both words (blue).

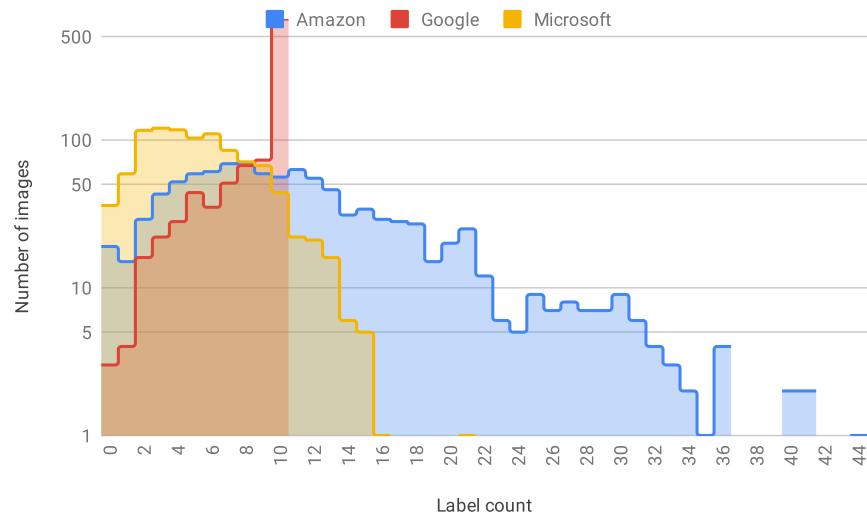


Figure 9.3: Number of labels responded from our input dataset to three concrete APIs assessed.

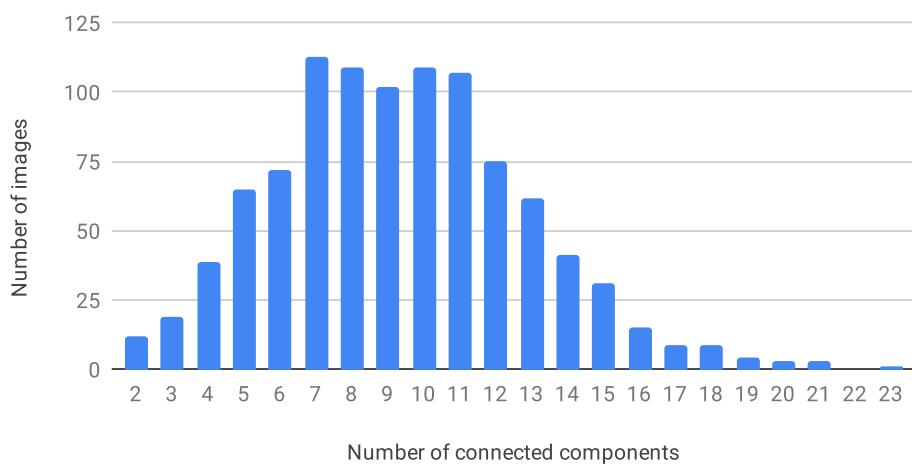


Figure 9.4: Number of connected components compared to the number of images.

4336 9.4 API Results Merging Algorithm

4337 Our proposed algorithm to merge labels consists of four parts: (1) mapping labels to
 4338 synsets, (2) deciding the total number of labels, (3) allocating the number of labels
 4339 to CCs, and (4) selecting labels from CCs.

4340 9.4.1 Mapping Labels to Synsets

4341 Labels returned in CVS responses are words (in natural language) that do not always
 4342 identify their intended meanings. For instance, a label *orange* may represent the
 4343 fruit, the colour, or the name of the longest river in South Africa. To identify the
 4344 actual meanings behind a label, our facade enumerates all synsets corresponding to
 4345 labels. It then finds the most likely synsets for labels by traversing WordNet links.
 4346 For instance, if an API endpoint outputs the ‘*orange*’ and ‘*lemon*’ labels, the
 4347 facade regards ‘*orange*’ as a related synset word of ‘*fruit*’. If an API endpoint
 4348 outputs ‘*orange*’ and ‘*water*’ labels, the facade regards ‘*orange*’ as a ‘*river*’.

4349 9.4.2 Deciding Total Number of Labels

4350 The number of labels in responses from endpoints vary as described in Section 9.3.1.
 4351 The facade decides the number of merged labels using the numbers of labels from
 4352 each endpoint. We formulate the following equation to calculate the number of
 4353 labels:

$$\min_i(|R_i|) \leq \frac{\sum_i |R_i|}{n} \leq \max_i(|R_i|) \leq \sum_i |R_i|$$

4354 where $|R|$ is number of labels and scores in response, and n is number of endpoints.
 4355 In case of naive operations in Section 9.2.2, the following is true:

$$\begin{aligned} |\text{merge}_{\max}(R_1, \dots, R_n)| &\leq \min_i(|R_i|) \\ \max_i(|R_i|) &\leq |\text{merge}_{\min}(R_1, \dots, R_n)| \leq \sum_i |R_i| \\ \max_i(|R_i|) &\leq |\text{merge}_{\text{average}}(R_1, \dots, R_n)| \leq \sum_i |R_i|. \end{aligned}$$

4356 The proposal uses $\lfloor \sum_i |R_i| / n \rfloor$ to conform to the necessary condition described in
 4357 Section 9.4.3.

4358 9.4.3 Allocating Number of Labels to Connected Components

4359 The graph of labels and synsets is then divided into several CCs. The facade decides
 4360 how many labels are allocated for each CC. For example, in Figure 9.5, there are
 4361 three CCs, where square-shaped nodes are labels in responses from endpoints. Text
 4362 within these label nodes describe which endpoint outputs the label and score, for
 4363 instance, “L-1a, 0.9” is label *a* from endpoint *L* with a score 0.9. Circle-shaped nodes
 4364 represent synsets, where the edges between the label and synset nodes indicate the
 4365 relationships between them. Edges between synsets are links in WordNet.

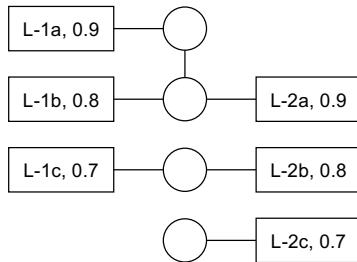


Figure 9.5: Allocation to connected components.

4366 Allegorically, allocating the number of labels to CCs is similar to proportional
 4367 representation in a political voting system, where CCs are the political parties and
 4368 labels are the votes to a party. Several allocation algorithms are introduced in
 4369 proportional representation, for instance, the D'Hondt and Hare-Niemeyer methods
 4370 [254]. However, there are differences from proportional representation in the politi-
 4371 cal context. For label merging, labels have scores and origin endpoints and such
 4372 information may improve the allocation algorithm. For instance, CCs supported
 4373 with more endpoints should have a higher allocation than CCs with fewer endpoints,
 4374 and CCs with higher scores should have a higher allocation than CCs with lower
 4375 scores. We introduce an algorithm to allocate the number of labels to CCs. This
 4376 allocates more to a CC with more supporting endpoints and higher scores. The steps
 4377 of the algorithm are:

- 4378 **Step I.** Sort scores separately for each endpoint.
- 4379 **Step II.** If all CCs have an empty score array or more, remove one, and go to Step
 4380 II.
- 4381 **Step III.** Select the highest score for each endpoint and calculate product of highest
 4382 scores.
- 4383 **Step IV.** A CC with the highest product score receives an allocation. This CC
 4384 removes every first element from the score array.
- 4385 **Step V.** If the requested number of allocations is complete, then stop allocation.
 4386 Otherwise, go to Step II.

4387 Tables 9.2 to 9.5 are examples of allocation iterations. In Table 9.2, the facade
 4388 sorts scores separately for each endpoint. For instance, the first CC in Figure 9.5
 4389 has scores of 0.9 and 0.8 from endpoint 1 and 0.9 from endpoint 2. All CCs have a
 4390 non-empty score array or more, so the facade skips Step II. The facade then picks
 4391 the highest scores for each endpoint and CC. CC 1 has the largest product of highest
 4392 scores and receives an allocation. In Table 9.3, the first CC removes every first score
 4393 in its array as it received an allocation in Table 9.2. In this iteration, the second CC
 4394 has largest product of scores and receives an allocation. In Table 9.4, the second CC
 4395 removes every first score in its array. At Step II, all the three CCs have an empty
 4396 array. The facade removes one empty array from each CC. In Table 9.5, the first CC
 4397 receives an allocation. The algorithm is applicable if total number of allocation is

Table 9.2: Allocation iteration 1.

Scores	Highest	Product	Allocated
[0.9, 0.8], [0.9]	[0.9, 0.9]	0.81	0+1
[0.7], [0.8]	[0.7, 0.8]	0.56	0
[], [0.7]	[N/A, 0.7]	N/A	0

Table 9.4: Allocation iteration 3.

Scores	Highest	Product	Allocated
[0.8], []	—	—	1
[], []	—	—	1
[], [0.7]	—	—	0

Table 9.3: Allocation iteration 2.

Scores	Highest	Product	Allocated
[0.8], []	[0.8, N/A]	N/A	1
[0.7], [0.8]	[0.7, 0.8]	0.56	0+1
[], [0.7]	[N/A, 0.7]	N/A	0

Table 9.5: Allocation iteration 4.

Scores	Highest	Product	Allocated
[0.8]	[0.8]	0.8	1+1
[]	[N/A]	N/A	1
[0.7]	[0.7]	0.7	0

4398 less than or equal to $\max_i(|R_i|)$ as scores are removed in Step II. The condition is a
 4399 necessary condition.

4400 **9.4.4 Selecting Labels from Connected Components**

4401 For each CC, the facade applies the *average* operator from Section 9.2.2 and takes
 4402 labels with n -highest scores up to allocation, as per Section 9.4.3.

4403 **9.4.5 Conformance to properties**

4404 Section 9.2.3 defines four properties: identity, commutativity, reflexivity, and addi-
 4405 tivity. Our proposed method conforms to these properties:

- 4406 • *identity*: the method outputs same result if there is one response;
- 4407 • *commutativity*: the method does not care about ordering of operands;
- 4408 • *reflexivity*: the allocations to CCs are same to number of labels in CCs; and
- 4409 • *additivity*: increases in score increases or does not change the allocation to
 4410 the corresponding CC.

4411 **9.5 Evaluation**

4412 **9.5.1 Evaluation Method**

4413 To evaluate the merge methods, we merged CVS results from three representative
 4414 image analysis API endpoints and compared these merged results against human-
 4415 verified labels. Images and human-verified labels are sourced from 1,000 randomly-
 4416 sampled images from the Open Images Dataset V4 [413] Image-Level Labels test
 4417 set.

4418 The first three rows in Table 9.7 are the evaluation of original responses from
 4419 each API endpoint. Precision, recall, and F-measure in Table 9.7 do not reflect
 4420 actual values: for instance, it appears that Google performs best at first glance, but
 4421 this is mainly because Google's labels are similar to that of the Open Images label
 4422 set.

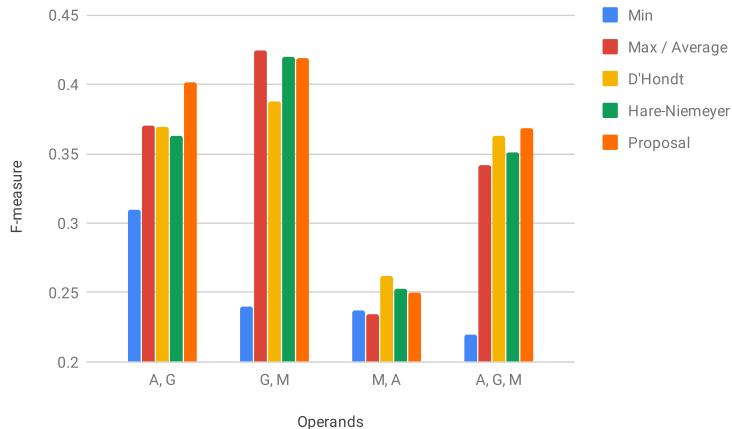


Figure 9.6: F-measure comparison.

4423 The Open Images Dataset uses 19,995 classes for labelling. The human-verified
 4424 labels for the 1,000 images contain 8,878 of these classes. Table 9.6 shows the
 4425 correspondence between each service's labels and the Open Images Dataset classes.
 4426 For instance, Amazon Rekognition outputs 11,416 labels in total for 1,000 images.
 4427 There are 1,409 unique labels in 11,416 labels. 1,111 labels out of 1,409 can be
 4428 found in Open Images Dataset classes. Rekognition's labels matches to Open Images
 4429 Dataset classes at 78.9% ratio, while Google has an outstanding matched percentage
 4430 of 94.1%. This high match is likely due to Google providing both Google Cloud
 4431 Vision and the Open Images Dataset—it is likely that they are trained on the same
 4432 data and labels. An endpoint with higher matched percentage has a more similar
 4433 label set to the Open Images Dataset classes. However, a higher matched percentage
 4434 does not mean imply *better quality* of an API endpoint; it will increase apparent
 4435 precision, recall, and F-measure only.

4436 The true and false positive (TP/FP) label averages and the TP/FP ratio is shown
 4437 in Table 9.7. Where the TP/FP ratio is larger, the scores are more reliable, however
 4438 it is possible to increase the TP/FP ratio by adding more false labels with low scores.
 4439 On the other hand, it is impossible to increase F-measure intentionally, because
 4440 increasing precision will decrease recall, and vice versa. Hence, the importance of
 4441 the F-measure statistic is critical for our analysis.

4442 Let R_A , R_G , and R_M be responses from Amazon Rekognition, Google Cloud
 4443 Vision, and Microsoft's Azure Computer Vision, respectively. There are four sets of
 4444 operands, i.e., (R_A, R_G) , (R_G, R_M) , (R_M, R_A) , and (R_A, R_G, R_M) . Table 9.7 shows the
 4445 evaluation of each operands set, Table 9.8 shows the averages of the four operands
 4446 sets, and Figure 9.6 shows the comparison of F-measure for each methods.

4447 9.5.2 Naive Operators

4448 Results of *min*, *max*, and *average* operators are shown in Tables 9.7 and 9.8 and Fig-
 4449 ure 9.6. The *min* operator is similar to *union* operator of set operation, and outputs
 4450 all labels of operands. The precision of the *min* operator is always greater than any

Table 9.6: Matching to human-verified labels.

Endpoint	Total	Unique	Matched	Matched %
Amazon Rekognition	11,416	1,409	1,111	78.9
Google Cloud Vision	8,766	2,644	2,487	94.1
Azure Computer Vision	5,392	746	470	63.0

Table 9.7: Evaluation results. A = Amazon Rekognition, G = Google Cloud Vision, M = Microsoft’s Azure Computer Vision.

Operands	Operator	Precision	Recall	F-measure	TP average	FP average	TP/FP ratio
A		0.217	0.282	0.246	0.848 ± 0.165	0.695 ± 0.185	1.220
G		0.474	0.465	0.469	0.834 ± 0.121	0.741 ± 0.132	1.126
M		0.263	0.164	0.202	0.858 ± 0.217	0.716 ± 0.306	1.198
A, G	Min	0.771	0.194	0.310	0.805 ± 0.142	0.673 ± 0.141	1.197
A, G	Max	0.280	0.572	0.376	0.850 ± 0.136	0.712 ± 0.171	1.193
A, G	Average	0.280	0.572	0.376	0.546 ± 0.225	0.368 ± 0.114	1.485
A, G	D’Hondt	0.350	0.389	0.369	0.713 ± 0.249	0.518 ± 0.202	1.377
A, G	Hare-Niemeyer	0.344	0.384	0.363	0.723 ± 0.242	0.527 ± 0.199	1.371
A, G	Proposal	0.380	0.423	0.401	0.706 ± 0.239	0.559 ± 0.190	1.262
G, M	Min	0.789	0.142	0.240	0.794 ± 0.209	0.726 ± 0.210	1.093
G, M	Max	0.357	0.521	0.424	0.749 ± 0.135	0.729 ± 0.231	1.165
G, M	Average	0.357	0.521	0.424	0.504 ± 0.201	0.375 ± 0.141	1.342
G, M	D’Hondt	0.444	0.344	0.388	0.696 ± 0.250	0.551 ± 0.254	1.262
G, M	Hare-Niemeyer	0.477	0.375	0.420	0.696 ± 0.242	0.591 ± 0.226	1.179
G, M	Proposal	0.414	0.424	0.419	0.682 ± 0.238	0.597 ± 0.209	1.143
M, A	Min	0.693	0.143	0.237	0.822 ± 0.201	0.664 ± 0.242	1.239
M, A	Max	0.185	0.318	0.234	0.863 ± 0.178	0.703 ± 0.229	1.228
M, A	Average	0.185	0.318	0.234	0.589 ± 0.262	0.364 ± 0.144	1.616
M, A	D’Hondt	0.271	0.254	0.262	0.737 ± 0.261	0.527 ± 0.223	1.397
M, A	Hare-Niemeyer	0.260	0.245	0.253	0.755 ± 0.251	0.538 ± 0.218	1.402
M, A	Proposal	0.257	0.242	0.250	0.769 ± 0.244	0.571 ± 0.205	1.337
A, G, M	Min	0.866	0.126	0.220	0.774 ± 0.196	0.644 ± 0.219	1.202
A, G, M	Max	0.241	0.587	0.342	0.857 ± 0.142	0.714 ± 0.210	1.201
A, G, M	Average	0.241	0.587	0.342	0.432 ± 0.233	0.253 ± 0.106	1.712
A, G, M	D’Hondt	0.375	0.352	0.363	0.678 ± 0.266	0.455 ± 0.208	1.492
A, G, M	Hare-Niemeyer	0.362	0.340	0.351	0.693 ± 0.260	0.444 ± 0.216	1.559
A, G, M	Proposal	0.380	0.357	0.368	0.684 ± 0.259	0.484 ± 0.200	1.414

Table 9.8: Average of the evaluation result.

Operator	Precision	Recall	F-measure	TP/FP ratio
Min	0.780	0.151	0.252	1.183
Max	0.266	0.500	0.344	1.197
Average	0.266	0.500	0.344	1.539
D’Hondt	0.361	0.335	0.346	1.382
Hare-Niemeyer	0.361	0.336	0.347	1.378
Proposal	0.358	0.362	0.360	1.289

4451 precision of operands, and the recall is always lesser than any precision of operands.
 4452 *Max* and *average* operators are similar to *intersection* operator of set operations.
 4453 Both operators output intersection of labels of operands and there is no clear relation
 4454 to the precision and recall of operands. Since both operators have the same preci-
 4455 sion, recall, and F-measure, Figure 9.6 groups them into one. The *average* operator
 4456 performs well on the TP/FP ratio, where most of the same labels from multiple
 4457 endpoints are TPs. In many cases of the four operand sets, all naive operators'
 4458 F-measures are between F-measures of operands. None of naive operators therefore
 4459 improve results by merging responses from multiple endpoints.

4460 9.5.3 Traditional Proportional Representation Operators

4461 There are many existing allocation algorithms in proportional representation, e.g.,
 4462 the Niemeyer and Niemeyer method [254]. These methods may be replacements of
 4463 those in Section 9.4.3. Other steps, i.e., Sections 9.4.1, 9.4.2 and 9.4.4, are the same
 4464 as for our proposed technique. Tables 9.7 and 9.8 and Figure 9.6 show the result of
 4465 these traditional proportional representation algorithms. Averages of F-measures by
 4466 traditional proportional representation operators are almost equal to that of the *max*
 4467 and *average* operators. It is worth noting that merging *M* and *A* responses results in
 4468 a better F-measure than each F-measure of *M* and *A* individually. As these are not
 4469 biased to human-verified labels, situations in the real-world usage should, therefore,
 4470 be similar to the case of *M* and *A*. Hence, RQ1 is true.

4471 9.5.4 New Proposed Label Merge Technique

4472 As shown in Table 9.8, our proposed new method performs best in F-measure.
 4473 Instead, the TP/FP ratio is less than *average*, the D'Hondt method, and Hare-
 4474 Niemeyer method. As described in Section 9.5.1, we argue that F-measure is a
 4475 more important measure than the TP/FP ratio (in this case). Therefore, RQ2 is
 4476 true. Shown in Table 9.7, our proposed new method improves the results when
 4477 merging *M* and *A* in non-biased endpoints. It is similar to traditional proportional
 4478 representation operators, but does not perform as well. However, it performs better
 4479 on other operand sets, and performs best overall as shown in Figure 9.6.

4480 9.5.5 Performance

4481 We used AWS EC2 m5.large instance (2 vCPUs, 2.5 GHz Intel Xeon, 8 GiB RAM);
 4482 Amazon Linux 2 AMI (HVM), SSD Volume Type; Node.js 8.12.0. It takes 0.370
 4483 seconds to merge responses from three endpoints. Computational complexity of the
 4484 algorithm in Section 9.4.3 is $O(n^2)$, where n is total number of labels in responses.
 4485 (The estimation assumes that the number of endpoints is a constant.) Complexity
 4486 of Step I in Section 9.4.3 is $O(n \log n)$, as the worst case is that all n labels are from
 4487 one single endpoint and all n labels are in one CC. Complexity of Step II to Step V
 4488 is $O(n^2)$, as the number of CCs is less than or equal to n and number of iterations
 4489 are less than or equal to n . As Table 9.1 shows, the averaged total number of three
 4490 endpoints is 25.58. Most of time for merging is consumed by looking up WordNet

4491 synsets (Section 9.4.1). The API facade calls each APIs on actual endpoints in
4492 parallel. It takes about 5 seconds, which is much longer than 0.370 seconds taken
4493 for the merging of responses.

4494 9.6 Conclusions and Future Work

4495 In this paper, we propose a method to merge responses from CVSSs. Our method
4496 merges API responses better than naive operators and other proportional represen-
4497 tation methods (i.e., D'Hondt and Hare-Niemeyer). The average of F-measure of
4498 our method marks 0.360; the next best method, Hare-Niemeyer, marks 0.347. Our
4499 method and other proportional representation methods are able to improve the F-
4500 measure from original responses in some cases. Merging non-biased responses
4501 results in an F-measure of 0.250, while original responses have an F-measure be-
4502 tween 0.246 and 0.242. Therefore, users can improve their applications' precision
4503 with small modification, i.e., by switching from a singular URL endpoint to a facade-
4504 based architecture. The performance impact by applying facades is small, because
4505 overhead in facades is much smaller than API invocation. Our proposal method
4506 conforms identity, commutativity, reflexivity, and additivity properties and these
4507 properties are advisable for integrating multiple responses.

4508 Our idea of a proportional representation approach can be applied to other IWSs.
4509 If the response of such a service is list consisting of an entity and score, and if there is a
4510 way to group entities, a proposal algorithm can be applied. The opposite approach is
4511 to improve results by inferring labels. Our current approach picks some of the labels
4512 returned by endpoints. IWSs are not only based on supervised ML—thus to cover a
4513 wide range of IWSs, it is necessary to classify and analyse each APIs and establish
4514 a method to improve results by merging. Currently graph structures of labels and
4515 synsets (Figure 9.2) are not considered when merging results. Propagating scores
4516 from labels could be used, losing the additivity property but improving results for
4517 users. There are many ways to propagate scores. For instance, setting propagation
4518 factors for each link type would improve merging and could be customised for users'
4519 preferences. It would be possible to generate an API facade automatically. APIs
4520 with the same functionality have same or similar signatures. Machine-readable API
4521 documentation, for instance, OpenAPI Specification, could help a generator to build
4522 an API facade.

CHAPTER 10

4523

4524

4525 Threshy: Supporting Safe Usage of Intelligent Web Services[†]

4526

4527 **Abstract** Increased popularity of ‘intelligent’ web services provides end-users with machine-
4528 learnt functionality at little effort to developers. However, these services require a decision
4529 threshold to be set which is dependent on problem-specific data. Developers lack a systematic
4530 approach for evaluating intelligent services and existing evaluation tools are predominantly
4531 targeted at data scientists for pre-development evaluation. This paper presents a workflow
4532 and supporting tool, Threshy, to help *software developers* select a decision threshold suited
4533 to their problem domain. Threshy is designed for tuning the confidence scores returned by
4534 intelligent web services and does not deal with hyper-parameter optimisation used in ML
4535 models. Additionally, it considers the financial impacts of false positives. Unlike existing
4536 tools, Threshy is designed to operate in multiple workflows including pre-development, pre-
4537 release, and support. Threshold configuration files exported by Threshy can be integrated
4538 into client applications and monitoring infrastructure. Demo: <https://bit.ly/2YKeYhE>.

4539 10.1 Introduction

4540 Machine learning algorithm adoption is increasing in modern software. End users
4541 routinely benefit from machine-learnt functionality through personalised recom-
4542 mendations [82], voice-user interfaces [249], and intelligent digital assistants [52]. The
4543 easy accessibility and availability of intelligent web services (IWSs)¹ is contribut-
4544 ing to their adoption. These IWSs simplify the development of machine learning
4545 solutions as they (i) do not require specialised machine learning expertise to build
4546 and maintain, (ii) abstract away infrastructure related issues associated with machine

[†]This chapter is originally based on A. Cummaudo, S. Barnett, R. Vasa, and J. Grundy, “Threshy: Supporting Safe Usage of Intelligent Web Services,” 2020, Unpublished. Terminology has been updated to fit this thesis.

¹Such as Azure Computer Vision (<https://azure.microsoft.com/en-au/services/cognitive-services/computer-vision/>), Google Cloud Vision (<https://cloud.google.com/vision/>), or Amazon Rekognition (<https://aws.amazon.com/rekognition/>).

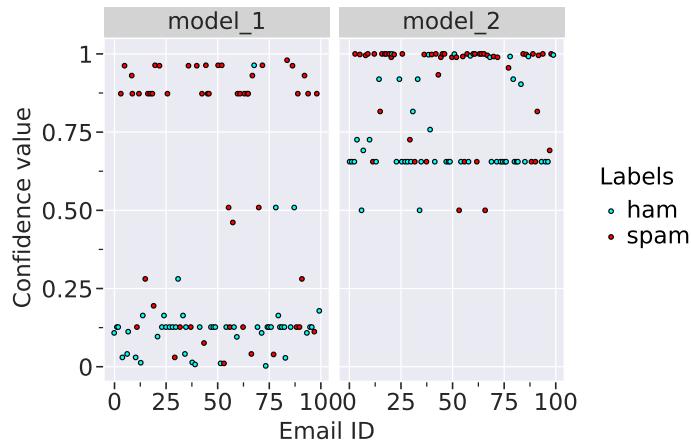


Figure 10.1: Predictions for 100 emails from two spam classifiers. Decision thresholds are classifier-dependent: a single threshold for both classifiers is *not* appropriate as ham emails are clustered at 0.12 (model_1) and at 0.65 (model_2). Developers must evaluate performance for *both* thresholds.

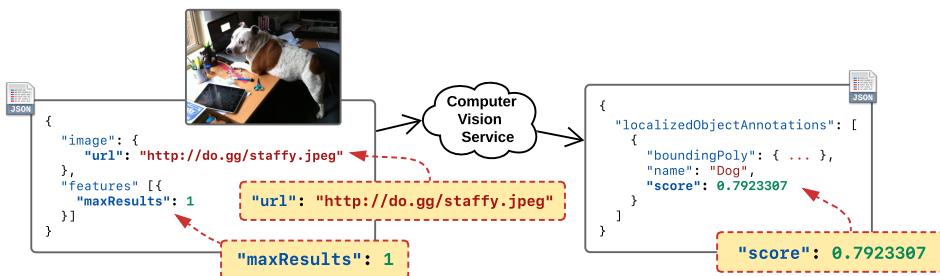


Figure 10.2: Request and response for an intelligent computer vision web service with only three configuration parameters: the image’s `url`, `maxResults` and `score`.

learning [15, 312], and (iii) provide web APIs for ease of integration.

However, unlike traditional web services, the functionality of these *intelligent services* is dependent on a set of assumptions unique to machine learning [87]. These assumptions are based on the data used to train machine learning algorithms, the choice of algorithm, and the choice of data processing steps—most of which are not documented. For developers, these assumptions mean that the performance characteristics of an intelligent service in any particular application problem domain is not fully knowable. Intelligent services represent this uncertainty through a confidence value associated with their predictions. As an example, consider Figure 11.5, which illustrates an image of a dog uploaded to a real computer vision service. Developers have very few configuration parameters in the upload payload (`url` for the image to analyse and `maxResults` for the number of objects to detect). The JSON output payload provides the confidence value of its estimated bounding box and label of the dog object via its `score` field (0.792). Developers can only modify these parameters to influence the score to improve the performance of the intelligent web service. This is unlike hyper-parameter optimisation, which configures the internal

4563 parameters of the algorithm for training a model. In this case, developers have no
4564 insight into which hyperparameters are used or the algorithm selected and cannot
4565 tune the trained model. Thus an evaluation procedure must be followed as a part of
4566 using an intelligent service for an application.

4567 A typical evaluation process would involve a test data set (curated by the devel-
4568 opers using the intelligent service) that is used to determine an appropriate threshold.
4569 Choice of a decision threshold is a critical element of the evaluation procedure [149].
4570 This is especially true for classification problems such as detecting if an image con-
4571 tains cancer or identifying all of the topics in a document. Simple approaches
4572 to selecting a threshold are often insufficient, as highlighted in Google’s machine
4573 learning course: “*It is tempting to assume that [a] classification threshold should
4574 always be 0.5, but thresholds are problem-dependent, and are therefore values that
4575 you must tune.*”² As an example consider the predictions from two email spam
4576 classifiers shown in Figure 10.1. The predicted safe emails, ‘ham’, are in two sepa-
4577 rate clusters (a simple threshold set to approx. 0.2 for model 1 and 0.65 for model
4578 2, indicating that different decision thresholds may be required depending on the
4579 classifier. Also note that some emails have been misclassified; how many depends on
4580 the choice of decision threshold. An appropriate threshold considers factors outside
4581 algorithmic performance, such as financial cost and impact of wrong decisions. To
4582 select an appropriate decision threshold, developers using intelligent services need
4583 approaches to reason about and consider trade-offs between competing *cost fac-
4584 tors*. These include impact, financial costs, and maintenance implications. Without
4585 considering these trade-offs, sub-optimal decision thresholds will be selected.

4586 The standard approach for tuning thresholds in classification problems involve
4587 making trade-offs between the number of false positives and false negatives using
4588 the receiver operating characteristic (ROC) curve. However, developers (i) need
4589 to realise that this trade-off between false positives and false negatives is a data
4590 dependent optimisation process [311], (ii) often need to develop custom scripts
4591 and follow a trial-and-error based approach to determine a threshold, (iii) must
4592 have appropriate statistical training and expertise, and (iv) be aware that multi-
4593 label classification require more complex optimisation methods when setting label
4594 specific costs. However, current intelligent services do not sufficiently guide or
4595 support software engineers through the evaluation process, nor do they make this
4596 need clear in the documentation.

4597 In this paper we present **Threshy**³, a tool to assist developers in selecting de-
4598 cision thresholds when using intelligent services. The motivation for developing
4599 Threshy arose from our consultancy work with industry. Unlike existing tooling (see
4600 Section 10.4), **Threshy serves as a means to up-skill and educate software en-**
4601 **gineers in selecting machine-learnt decision thresholds**, for example, on aspects
4602 such as confusion matrices. We re-iterate that the end-users of Threshy are software
4603 engineers and not data scientists—**Threshy is not designed for hyper-parameter**
4604 **tuning of models**, but for threshold tuning of intelligent web services where internal
4605 models are not exposed. Threshy provides a visually interactive interface for devel-

²See <https://bit.ly/36oMgWb>.

³Threshy is available for use at <http://bit.ly/a2i2threshy{}>.

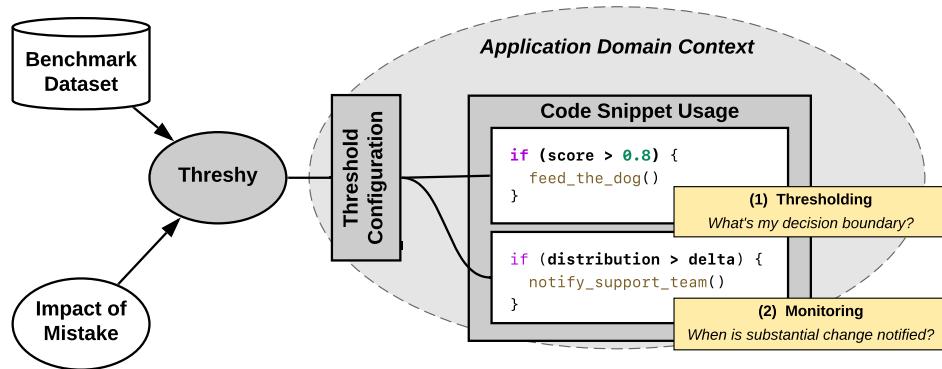


Figure 10.3: Threshy supports two key aspects for intelligent web services: threshold selection and monitoring.

4606 operates to fine-tune thresholds and explore trade-offs of prediction hits/misses. This
 4607 exposes the need for optimisation of thresholds, which is dependent on particular
 4608 use cases.

4609 Threshy improves developer productivity through automation of the threshold
 4610 selection process by leveraging an optimisation algorithm to propose thresholds.
 4611 Figure 10.3 illustrates the two key aspects by which Threshy can assist the developer's application domain context. Developers input a representative dataset of
 4612 their application data (a benchmark dataset) in addition to cost factors to Threshy.
 4613 Threshy's output helps developers select appropriate thresholds within their applica-
 4614 tions and can be used for monitoring if substantial change occurs within the service.
 4615 The algorithm considers different cost factors providing developers with summary
 4616 information so they can make more informed trade-offs. Developers also benefit
 4617 from the workflow implemented in Threshy by providing a reproducible procedure
 4618 for testing and tuning thresholds for any category of classification problem (binary,
 4619 multi-class, and multi-label). Threshy has also been designed to work for different
 4620 input data types including images, text and categorical values. The output, is a
 4621 text file and can be integrated into client applications ensuring that the thresholds
 4622 can be updated without code changes (if needed), and continuously monitored in a
 4623 production setting.
 4624

4625 10.2 Motivating Example

4626 As a motivating example consider Nina, a fictitious developer, who has been em-
 4627 ployed by Lucy's Tomato Farm to automate the picking of tomatoes from their vines
 4628 (when ripe) using computer vision and a harvesting robot. Lucy's Farm grow five
 4629 types of tomatoes (roma, cherry, plum, green, and yellow tomatoes). Nina's robot—
 4630 using an attached webcam—will crawl and take a photo of each vine to assess it
 4631 for harvesting. Nina's automated harvester needs to sort picked tomatoes into a
 4632 respective container, and thus several business rules need to be encoded into the
 4633 prediction logic to sort each tomato detected based on its *ripeness* (ripe or not ripe)

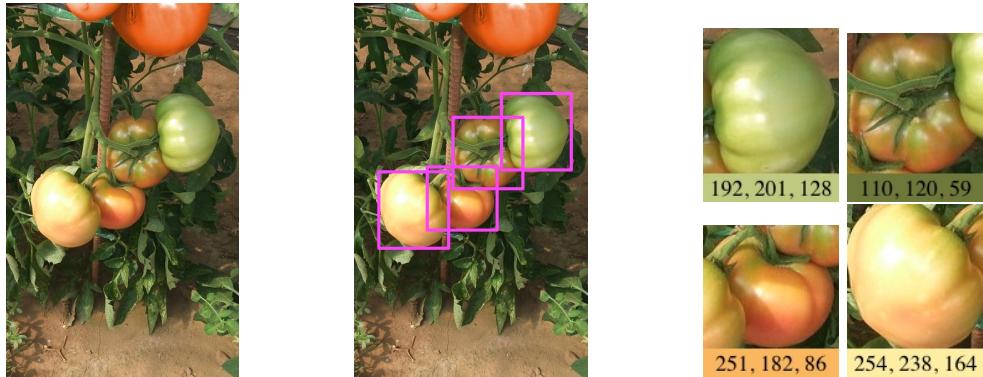


Figure 10.4: Pipeline of Nina’s harvesting robot. *Left:* Photo from harvesting robot’s webcam. *Centre:* Classification detecting different types of tomatoes. *Right:* Binary classification for ripeness (ripe/unripe) based on (R, G, B values).

4634 and *type of tomato* (as above).

4635 Nina uses a two-stage pipeline consisting of a multi-class and a binary classi-
4636 fication model. She has decided to evaluate the viability of cloud based intelligent
4637 services and use them if operationally effective. Figure 10.4 illustrates an example
4638 of the pipeline as listed below:

- 4639 1. **Classify tomato ‘type’.** This stage uses an object localisation service to detect
4640 all tomato-like objects in the frame and classifies each tomato into one of the
4641 following labels: [‘roma’, ‘cherry’, ‘plum’, ‘green’, ‘yellow’].
- 4642 2. **Assess tomato ‘ripeness’.** This stage uses a crop of the localised tomatoes
4643 from the original frame to assess the crop’s colour properties (i.e., average
4644 colour must have $R > 200$ and $G < 240$). This produces a binary classification
4645 to deduce whether the tomato is ripe or not.

4646 Nina only has a minimal appreciation of the evaluation method to use for off-
4647 the-shelf computer vision (classification) services. She also needs to consider the
4648 financial costs of mis-classifying either the tomato type or the ripeness. Missing a
4649 few ripe tomatoes isn’t a problem as the robot travels the field twice a week during
4650 harvest season. However, picking an unripe tomato is expensive as Lucy cannot sell
4651 them. Therefore, Nina needs a better (automated) way to assess the performance
4652 of the service and set optimal thresholds for her picking robot, thereby maximising
4653 profit.

4654 To assist in developing Nina’s pipeline, Lucy sampled a section of 1000 tomatoes
4655 by taking a photo of each tomato, labelling its type, and assessing whether the vine
4656 was ‘ripe’ or ‘not_ripe’. Nina ran the labelled images through an intelligent
4657 service, with each image having a predicted type (multi-class) and ripeness (binary),
4658 with respective confidence values.

4659 Nina combined the predictions, their respective confidence values, and Lucy’s
4660 labelled ground truths into a CSV file which was then uploaded to Threshy. Nina
4661 asked Lucy to assist in setting relevant costs for correct predictions and false predictions.
4662 Threshy then recommended a choice of decision threshold which Nina then

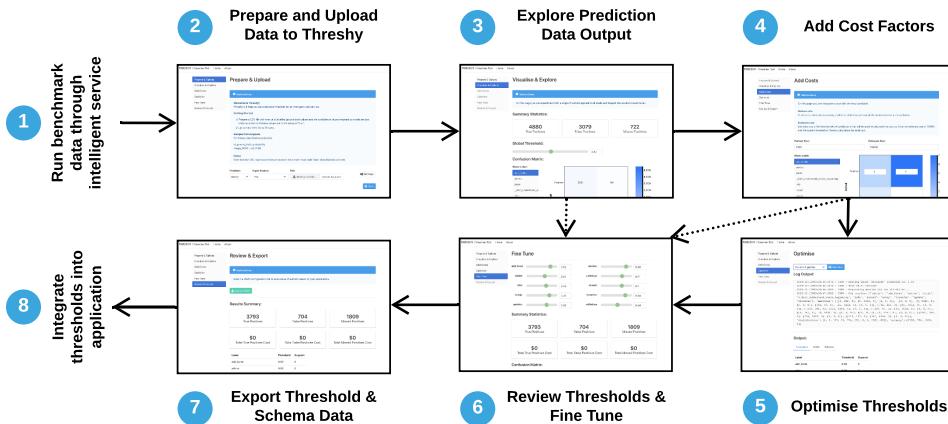


Figure 10.5: UI workflow for interacting with Threshy to optimise the thresholds for classification problem.

fine tuned while considering the performance and cost implications.

10.3 Threshy

Threshy is a tool to assist software engineers with setting decision thresholds when integrating machine-learnt components in a system. Our tool also serves as a method to inform and educate engineers about the nuances to consider. The novel features of Threshy are:

- Automating threshold selection using an optimisation algorithm (NSGA-II [96]), optimising the results for each label.
- Support for additional user defined weights when optimising thresholds such as financial costs and impact to society (different type of cost). This allows decision thresholds to be set within a business context as they differ from application to application [107].
- Handles nuances of classification problems such as dealing with multi-objective optimisation, and metric selection—reducing errors of omission.
- Support key classification problems including binary (e.g. email is either spam or ham), multi-class (e.g. predicting the colour of a car), and multi-label (e.g. assign multiple topics to a document). Existing tools ignore multi-label classification.

Setting thresholds in Threshy is an eight step process as shown in Figure 10.5. Software engineers ① run a benchmark dataset through the machine-learnt component to create a CSV file with true labels and predicted labels along with the predicted confidence values. The CSV file is then ② uploaded for initial exploration where engineers can ③ experiment with modifying a single global threshold for the dataset. Developers may choose to exit at this point (as indicated by dotted arrows in Figure 10.5). Optionally, the engineer ④ defines costs for missed predictions followed by selecting optimisation settings. The optional optimisation step of

4689 Threshy (5) considers the performance and costs when deriving the thresholds.
4690 Finally, the engineer can (6) review and fine tune the calculated thresholds, associated
4691 costs, and (7) download generated threshold meta-data to be (8) integrated into their
4692 application.

4693 Threshy runs a client/server architecture with a thin-client (see Figure 10.6). The
4694 web-based application consists of an interactive front-end where developers upload
4695 benchmark results—consisting of both human annotated labels (ground truths) and
4696 machine predictions (from the intelligent service)—and use threshold tuners (via
4697 sliders) to present a data summary of the uploaded CSV. Predicted performances
4698 and costs are entered manually into the web interface by the developer. The back-end
4699 of Threshy asynchronously runs a data analyser, cost processor and metrics calculator
4700 when relevant changes are made to the front-end’s tuning sliders. Separating the
4701 two concerns allows for high intensity processing to be done on the server and not
4702 the front end.

4703 The data analyser provides a comprehensive overview of confusion matrices
4704 compatible for multi-label multi-class classification problems. When representing
4705 the confusion matrix, it is trivial to represent instances where multi-label multi-
4706 classification is not considered. For example, in the simplest case, a single row in
4707 the matrix represents a single label out of two classes, or each row has one label but
4708 it has multiple classes. However, a more challenging case to visualise the confusion
4709 arises when you have n labels and n classes; the true/false matches become too
4710 excessive to visualise as it is disproportionate to the true results. To deal with this
4711 issue, we condense the summary statistics down to three constructs: (i) number of
4712 true positives, (ii) false positives, (iii) missed positives. This therefore allows us to
4713 optimise against the true positives and minimise the other two constructs.

4714 Threshy is a fully self-contained repository containing implementation of the
4715 tool, scripting and exploratory notebooks, which we make available at <https://github.com/a2i2/threshy>.

4717 10.4 Related work

4718 10.4.1 Decision Boundary Estimation

4719 Optimal machine-learnt decision boundaries depend on identifying the operating
4720 conditions of the problem domain. A systematic study by Drummond and Holte
4721 [107] classifies four such operating conditions to determine a decision threshold: (i)
4722 the operating condition is known and thus the model trained matches perfectly; (ii)
4723 where the operating conditions are known but change with time, and thus the model
4724 must be adaptable to such changes; (iii) where there is uncertainty in the knowledge
4725 of the operating conditions certain changes in the operating condition are more likely
4726 than others; (iv) where there is no knowledge of the operating conditions and the
4727 conditions may change from the model in any possible way. Various approaches
4728 to determine appropriate thresholds exist for all four of these cases, such as cost-
4729 sensitive learning, ROC analysis, cost curves, and Brier scores.

4730 However, an *automated* attempt to calibrate decision threshold boundaries is

not considered, and is largely pitched at a non-software engineering audience. A more recent study touches on this in model management for large-scale adversarial instances in Google’s advertising system [311], however this is only a single component within the entire architecture, and is not a tool that is useful for developer’s in varying contexts. Unlike this study, our work presents a ‘plug-and-play’ style calibration method where any context/domain can have thresholds automatically calibrated (in-context) *and* optimised for engineers; Threshy’s architecture and design facilitates operating in a headless mode enabling use in monitoring and support workflows.

10.4.2 Tooling for ML Frameworks

Support tools for ML frameworks generally fall into two categories; the first attempts to illuminate the ‘black box’ by offering ways in which developers can better understand the internals of the model to improve its performance. (For extensive analyses and surveys into this area, see [159, 270].) However, a recent emphasis to probe only inputs and outputs of a model has been explored, exploring off-the-shelf models without knowledge of its unknowns (see Figure 10.1) to reflect the nature of real-world development. Google’s *What-If Tool* [367] for Tensorflow provides a means for data scientists to visualise, measure and assess model performance and fairness with various hypothetical scenarios and data features; similarly, Microsoft’s *Gamut* tool [158] provides an interface to test hypotheticals (although only on Generalized Additive Models) and their *ModelTracker* tool [12] collates summary statistics on a set of sample data to enable rich visualisation of model behaviour and access to key performance metrics.

However, these tools are largely focused toward pre-development model evaluation and are not designed for the software engineering workflow. They are also targeted to data scientists and not engineers, and certain tools are tied to specific machine learning frameworks (e.g., What-If and Tensorflow). Our work attempts to bridge these gaps through a structured workflow with an automated tool targeted to software developers. We also consider the need to have a consistent tool that works across development, test, and production environments.

10.5 Conclusions & Future Work

Primary contributions of this work include Threshy, a tool for automating threshold selection, and the overall meta-workflow proposed in Threshy that developers can use as a point of reference for calibrating thresholds. In future work, we plan to evaluate Threshy with software engineers to identify additional insights required to make decision thresholds in practice and add code synthesis for monitoring concept drift and for implementing decision thresholds.

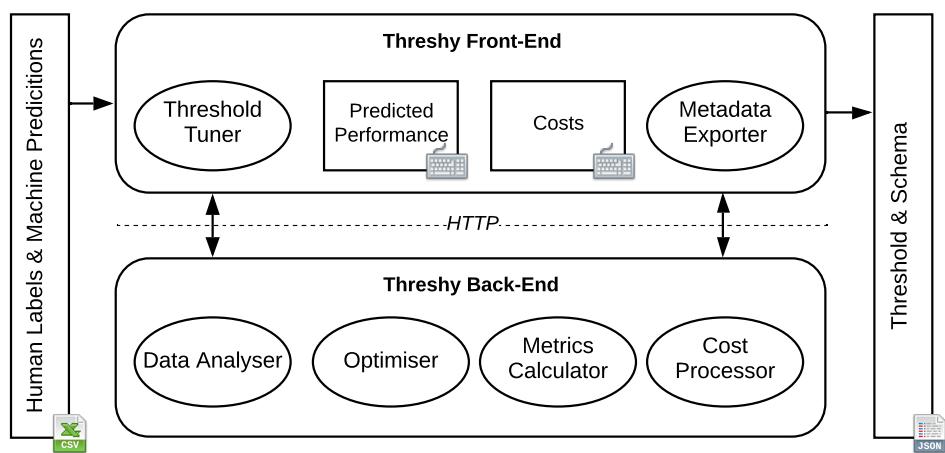


Figure 10.6: Architecture of Threshy.

4768

4769

4770 An Integration Architecture Tactic to guard AI-first Components[†]

4771

4772 **Abstract** Intelligent web services provide the power of AI to developers via simple REST-
4773 ful API endpoints, abstracting away many complexities of machine learning. However,
4774 most of these intelligent web services (IWSs)—such as computer vision—continually learn
4775 with time. When the internals within the abstracted ‘black box’ become hidden and evolve,
4776 pitfalls emerge in the robustness of applications that depend on these evolving services.
4777 Without adapting the way developers plan and construct projects reliant on IWSs, signifi-
4778 cant gaps and risks result in both project planning and development. Therefore, how can
4779 software engineers best mitigate software evolution risk moving forward, thereby ensuring
4780 that their own applications maintain quality? Our proposal is an architectural tactic designed
4781 to improve intelligent service-dependent software robustness. The tactic involves creating
4782 an application-specific benchmark dataset baselined against an intelligent service, enabling
4783 evolutionary behaviour changes to be mitigated. A technical evaluation of our implemen-
4784 tation of this architecture demonstrates how the tactic can identify 1,054 cases of substantial
4785 confidence evolution and 2,461 cases of substantial changes to response label sets using a
4786 dataset consisting of 331 images that evolve when sent to a service.

4787 11.1 Introduction

4788 The introduction of intelligent web services (IWSs) into the software engineering
4789 ecosystem allows developers to leverage the power of artificial intelligence (AI)
4790 without implementing complex AI algorithms, source and label training data, or
4791 orchestrate powerful and large-scale hardware infrastructure. This is extremely

[†]This chapter is originally based on A. Cummaudo, S. Barnett, R. Vasa, J. Grundy, and M. Abd-elrazek, “Beware the evolving ‘intelligent’ web service! An integration architecture tactic to guard AI-first components,” in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. Sacramento, CA, USA: ACM, November 2020, In Press. Terminology has been updated to fit this thesis.

4792 enticing for developers to embrace due to the effort, cost and non-trivial expertise
4793 required to implement AI in practice [281, 312].

4794 However, the vendors that offer these services also periodically update their
4795 behaviour (responses). The ideal practice for communicating the evolution of a
4796 web service involves updating the version number and writing release notes. The
4797 release notes typically describe new capabilities, known problems, and requirements
4798 for proper operation [50]. Developers anticipate changes in behaviour between ver-
4799 sioned releases although they expect the behaviour of a specific version to remain
4800 stable over time [353]. However, emerging evidence indicates that ‘intelligent’ ser-
4801 vices *do not* communicate changes explicitly [86]. Intelligent services evolve in
4802 unpredictable ways, provide no notification to developers and changes are undocu-
4803 mented [90]. To illustrate this, consider Figure 11.1, which shows the evolution of a
4804 popular computer vision service (CVS) with examples of labels and associated confi-
4805 dence scores changing are shown. This behaviour change severely negatively affects
4806 reliability. Applications may no longer function correctly if labels are removed or
4807 confidence scores change beyond predefined thresholds.

4808 Unlike traditional web services, the functionality of these IWSs is dependent
4809 on a set of assumptions unique to their machine learning principles and algorithms.
4810 These assumptions are based on the data used to train machine learning algorithms,
4811 the choice of algorithm, and the choice of data processing steps—most of which
4812 are not documented to service end users. The behaviour of these services evolve
4813 over time [87]—typically this implies the underlying model has been updated or
4814 re-trained.

4815 Vendors do not provide any guidance on how best to deal with this evolution in
4816 client applications. For developers to discover the impact on their applications they
4817 need to know the behavioural deviation and the associated impact on the robustness
4818 and reliability of their system. Currently, there is no guidance on how to deal with
4819 this evolution, nor do developers have an explicit checklist of the likely errors and
4820 changes that they must test for [90].

4821 In this paper, we present a reference architecture to detect the evolution of such
4822 IWSs, using a mature subset of these services that provide computer vision as an
4823 exemplar. This tactic can be used both by intelligent service consumers, to defend
4824 their applications against the evolutionary issues present in IWSs, and by service
4825 vendors to make their services more robust. We also present a set of error conditions
4826 that occur in existing CVSs.

4827 The key contributions of this paper are:

- 4828 • A set of new service error codes for describing the empirically observed error
4829 conditions in IWSs.
- 4830 • A new reference architecture for using IWSs with a Proxy Server that returns
4831 error codes based on an application specific benchmark dataset.
- 4832 • A labelled data set of evolutionary patterns in CVSs.
- 4833 • An evaluation of the new architecture and tactic showing its efficacy for
4834 supporting IWS evolution from both provider and consumer perspectives.

4835 The rest of this paper is organised thus: Section 11.2 presents a motivating



'natural foods' (.956) → 'granny smith' (.986)



'skiing' (.937) → 'snow' (.982)



'girl' (.660) → 'photography' (.738)



'water' (.972) → 'wave' (.932)



'tennis' (.982) → 'sports' (.989)



'neighbourhood' (.925) → 'blue' (.927)

Figure 11.1: Prominent CVSSs evolve with time which is not effectively communicated to developers. Each image was uploaded in November 2018 and March 2019 and the topmost label was captured. Specialisation in labels (*Left*), generalisation in labels (*Centre*) and emphasis change in labels (*Right*) are all demonstrated from the same service with no API change and limited release note documentation. Confidence values indicated in parentheses.

example that anchors our work; Section 11.3 presents a landscape analysis on IWSs; Section 11.4 presents an overview of our architecture; Section 11.5 describes the technical evaluation; Section 11.6 presents a discussion into the implications of our architecture, its limitations and potential future work; Section 11.7 discusses related work; Section 11.8 provides concluding remarks.

11.2 Motivating Example

We identify the key requirements for managing evolution of IWSs using a motivating example. Consider Michelina, a software engineer tasked with developing a fall detector system for helping aged care facilities respond to falls promptly. Michelina decides to build the fall detector with an intelligent service for detecting people as she has no prior experience with machine learning. The initial system built by Michelina consists of a person detector and custom logic to identify a fall based on rapid shape deformation (i.e., a vertical ‘person’ changing to a horizontal ‘person’ greater than specified probability threshold value). Due to the inherent uncertainty present in an intelligent service and the importance of correctly identifying falls, Michelina informs the aged care facility that they should manually verify falls before dispatching a nurse to the location. The aged care facility is happy with this approach but inform Michelina that only a certain percentage of falls can be manually verified based on the availability of staff. In order to reduce the manual work Michelina sets thresholds for a range of confidence scores where the system is uncertain. Michelina completes the fall detector using a well-known cloud-based intelligent image classification web service and her client deploys this new fall detection application.

Three months go by and then the aged care facility contact Michelina saying the percentage of manual inspections is far too high and could she fix it. Michelina is mystified why this is occurring as she thoroughly tested the application with a large dataset provided by the aged care facility. On further inspection Michelina notices that the problem is caused by some images classifying the person with a ‘child’ label rather than a ‘person’ label. Michelina is frustrated and annoyed at this behaviour as (i) the cloud vendor did not document or notify her of the change of the intelligent service behaviour, (ii) she does not know the best practice for dealing with such a service evolution, and (iii) she cannot predict how the service will change in the future. This experience also makes Michelina wonder what other types of evolution can occur and how can she minimise these behavioural changes on her critical care application. Michelina then begins building an ad-hoc solution hoping that what she designs will be sufficient.

For Michelina to build a robust solution she needs to support the following requirements:

- 4873 **R1.** Define a set of error conditions that specify the types of evolution that occur
4874 for an intelligent service.
- 4875 **R2.** Provide a notification mechanism for informing client applications of be-
4876 havioural changes to ensure the robustness and reliability of the application.

- 4877 **R3.** Monitor the evolution of IWSs for changes that affect the application's behaviour.
- 4878
- 4879 **R4.** Implement a flexible architecture that is adaptable to different IWSs and application contexts to facilitate reuse.
- 4880

4881 11.3 Intelligent Services

4882 We present background information on IWSs describing how they differ from traditional web services, the dimensions of their evolution and the currently limited configuration options available to users.

4883

4884

4885 11.3.1 ‘Intelligent’ vs ‘Traditional’ Web Services

4886 Unlike conventional web services, IWSs are built using AI-based components. These
4887 components are unlike traditional software engineering paradigms as they are data-
4888 dependent and do not result in deterministic outcomes. These services make future
4889 predictions on new data based solely against its training dataset; outcomes are
4890 expressed as probabilities that the inference made matches a label(s) within its
4891 training data. Further, these services are often marketed as forever evolving and
4892 ‘improving’. This means that their large training datasets may continuously update
4893 the prediction classifiers making the inferences, resulting both in probabilistic and
4894 non-deterministic outcomes [87, 161]. Critically for software engineers using the
4895 services, these non-deterministic aspects have not been sufficiently documented in
4896 the service’s API documented, which has been shown to confuse developers [90].

4897 A strategy to combat such service changes, which we often observe in traditional
4898 software engineering practices, are for such services to be versioned upon substantial
4899 change. Unfortunately emerging evidence indicates that prominent cloud vendors
4900 providing these IWSs do not release new versioned endpoints of the APIs when the
4901 *internal model* changes [87]. For IWSs, it is impossible to invoke requests specific
4902 to a particular version model that was trained at a particular date in time. This means
4903 that developers need to consider how evolutionary changes to the IWSs they make
4904 use of may impact their solutions *in production*.

4905 11.3.2 Dimensions of Evolution

4906 The various key dimensions of the evolution of IWSs is illustrated in Figure 11.2.
4907 There are two primary dimensions of evolution: *changes to the label sets* returned
4908 per image submitted and *changes to the confidences* per label in the set of labels
4909 returned per image. In the former, we identify two key aspects: cardinality changes
4910 and ontology changes. Cardinality changes occur when the service either introduces
4911 or drops a label for the same image at two different generations. Alternatively, the
4912 cardinality may remain stagnant, although this is not guaranteed. This results in
4913 an expectation mismatch by developers as to what labels can or will be returned by
4914 the service. For instance, the terms ‘black’ and ‘black and white’ may be found to
4915 be categorised as two separate labels. Secondly, the ontologies of these labels are

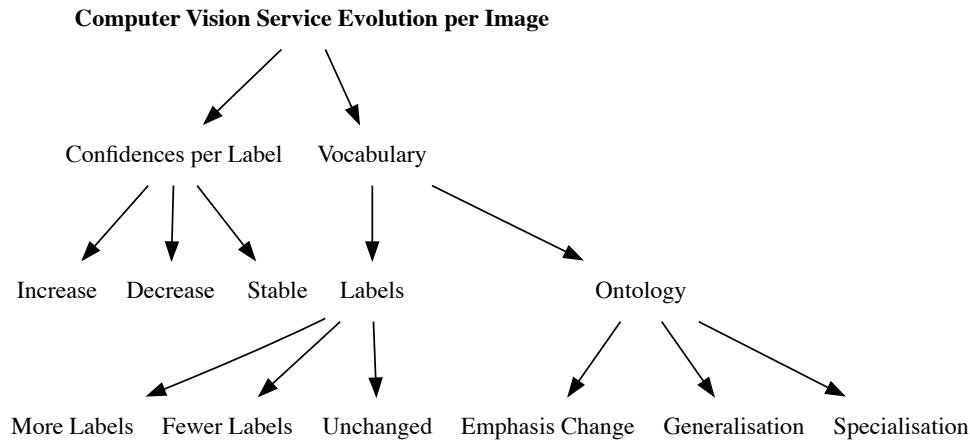


Figure 11.2: The dimensions of evolution identified within CVSSs.



Figure 11.3: A significant confidence increase ($\delta = +0.425$) from ‘window’ (0.559) to ‘water transportation’ (0.984) goes beyond simple decision boundaries.

4916 non-static, and a label may become more generalised into a hypernym, specialised
 4917 into a hyponym, or the emphasis of the label may change either to a co-hyponym or
 4918 another aspect in the image, such as the colour or scene, rather than the subject of
 4919 the image [87].

4920 Secondly, we have identified that the confidence values returned per label are also
 4921 non-static. While some services may present minor changes to labels’ confidences
 4922 resulting from statistical noise, other labels had significant changes that were beyond
 4923 basic decision boundaries. An example is shown in Figure 11.3. Developer code
 4924 written to assume certain ranges/confidence intervals will fail if the service evolves
 4925 in this way.

4926 11.3.3 Limited Configurability

4927 As an example, consider Figure 11.5, which illustrates an image of a dog uploaded to
 4928 a well-known cloud-based CVS. Developers have very few configuration parameters
 4929 in the upload payload (`url` for the image to analyse and `maxResults` for the number
 4930 of objects to detect). The JSON output payload provides the confidence value of its
 4931 estimated bounding box and label of the dog object via its `score` field (0.792). This
 4932 value indicates the level of confidence in the label returned, and is dependent on the
 4933 input to the underlying ML model used by that service. Developers set thresholds

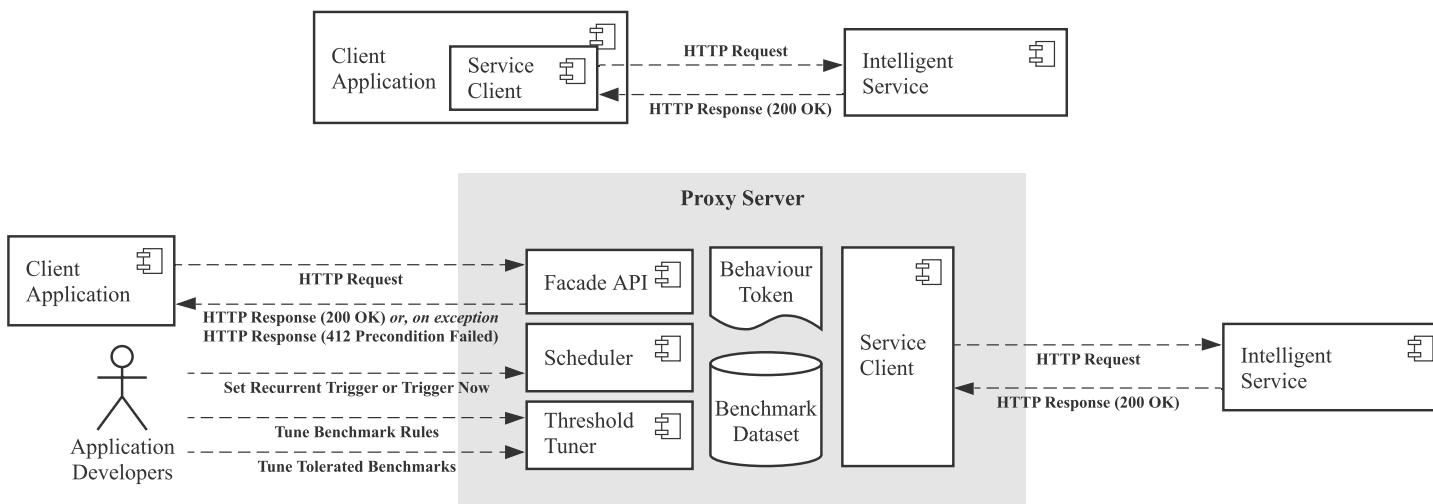


Figure 11.4: Top: Accessing an intelligent service directly. Bottom: Primary components of the Proxy Server approach.

as a decision boundary in this case, a threshold of “greater than 0.7” could indicate that the image contains a dog where as any other value the system is uncertain. These decision boundaries determine if the service’s output is accepted or rejected. However, these confidence scores change whenever a model is re-trained and these changes are not communicated or propagated to developers [87]. Developers can only modify these parameters to influence the score to improve the performance of the IWS. This is unlike many machine learning toolkit hyper-parameter optimisation facilities, which can be used to configure the internal parameters of the algorithm for training a model. In this case, developers using the IWS have no insight into which hyperparameters were used when training the model or the algorithm selected, and cannot tune the trained model. Thus an evaluation procedure must be followed as a part of using an intelligent service for an application to tune their output confidence values. and select appropriate threshold boundaries. While some service providers provide some guidance to thresholding,¹ they do not provide domain-specific tooling. This is because choice of appropriate thresholds is dependent on the data and must consider factors, such as algorithmic performance, financial cost, and impact of false-positives/negatives.

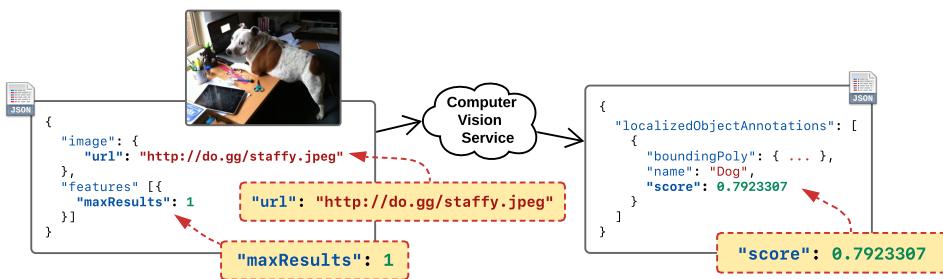


Figure 11.5: Request and response for an intelligent computer vision web service with only three configuration parameters: the image’s url, maxResults and score.

However, decision boundaries in service client code using simple If conditions around confidence scores is not a sufficient enough strategy, as evidence shows intelligent, non-deterministic web services change sporadically and unknowingly. Most traditional, deterministic code bases handle unexpected behaviour of called APIs via *error codes* and exception handling. Thus the non-deterministic components of the client code, such as those using CVSs, will also tend to conflict with their traditional deterministic components as the latter do not deal in terms of probabilities but in using error codes. This makes achieving robust component integration in client code bases hard. More sophisticated monitoring of IWSs in client code is therefore required to map the non-deterministic service behaviour changes to errors such that the surrounding infrastructure can support it and reduce interface boundary problems. While data science literature acknowledges the need for such an architecture [112] they do not offer any technical software engineering solutions to mitigate the issues such that software engineers have a pattern to work against it. To date, there do not

¹<https://bit.ly/36oMgWb> last accessed 20 May 2020.

Table 11.1: Potential reasons for a 412 Precondition Failed response.

Error Code	Error Description
No Key Yet	This indicates that the Proxy Server is still initialising its first behaviour token, i.e., k_0 does not yet exist.
Service Mismatch	The service encoded within the behaviour token provided to the Proxy Server does not match the service the Proxy Server is benchmarked against. This makes it possible for one Proxy Server to face multiple CVSs.
Dataset Mismatch	The benchmark dataset B encoded within the behaviour token does not match the benchmark dataset encoded within the Proxy Server.
Success Mismatch	The success of each response within the benchmark dataset must be true for a behaviour token to be used within a request. This error indicates that k_r is, therefore, not successful.
Min Confidence Mismatch	The minimum confidence delta threshold set in k_t does not match that of k_r .
Max Labels Mismatch	The maximum label delta threshold set in k_t does not match that of k_r .
Response Length Mismatch	The number of responses within k_t does not match that within k_r .
Label Delta Mismatch	An image within B has either dropped or gained a number of labels that exceeds the maximum label delta. Thus, k_r exceeds the threshold encoded within k_t .
Confidence Delta Mismatch	One of the labels within an image encoded in k_r exceeds the confidence threshold encoded within k_t .
Expected Labels Mismatch	One of the expected labels for an image within k_t is now missing.

4965 yet exist IWS client code architectures, tactics or patterns that achieve this goal.

4966 11.4 Our Approach

4967 To address the requirements from Section 11.2 we have developed a new Proxy
4968 Service² that includes: (i) evaluation of an intelligent service using an application
4969 specific benchmark dataset, (ii) a Proxy Server to provide client applications with
4970 evolution aware errors, and (iii) a scheduled evolution detection mechanism. The
4971 current approach of using an intelligent API via direct access is shown in Figure 11.4
4972 (top). In contrast, an overview of our approach is shown in Figure 11.4 (bottom).
4973 The following sections describe our approach in detail.

4974 11.4.1 Core Components

4975 For the purposes of this paper we assume that the intelligent service of interest
4976 is an image recognition service, but our approach generalises to other intelligent,
4977 trained model-based services e.g., natural language processing, document recog-

²A reference architecture is provided at <http://bit.ly/2TlMmDh>.

4978 nition, voice, etc. Each image, when uploaded to the intelligent service returns a
 4979 response (R) which is a set describing a label (l) of what is in the image (i) along
 4980 with its associated confidence (c)—thus $R_i = \{(l_1, c_1), (l_2, c_2), \dots (l_n, c_n)\}$. Most
 4981 documentation of these services imply that these confidence values are all what is
 4982 needed to handle evolution in their systems. This means that if a label changes
 4983 beyond a certain threshold, then the developer can deal with the issue then (or ignore
 4984 it). While this approach may work in some simple application contexts, in many it
 4985 may not. Our Proxy Server offers a way to monitor if these changes go beyond a
 4986 threshold of tolerance, checking against a domain-specific dataset over time.

4987 11.4.1.1 Benchmark Dataset

4988 Monitoring an intelligent service for behaviour change requires a Benchmark Dataset,
 4989 a set of n images. For each image (i) in the Benchmark Dataset (B) there is an associ-
 4990 ated label (l) that represents the true value for that item; $B_i = \{(i_1, l_1), (i_2, l_2), \dots (i_n, l_n)\}$.
 4991 This dataset is used to check for evolution in IWSs by periodically sending each im-
 4992 age within the dataset to the service’s API, as per the rules encoded within the
 4993 Scheduler (see Section 11.4.1.6). By using a dataset specific to the application
 4994 domain, developers can detect when evolution affects their application rather than
 4995 triggering all non-impactful changes. This helps achieve our requirement *R3*. *Mon-*
4996 itor the evolution of IWSs for changes that affect the application’s behaviour. Using
 4997 application-specific datasets also ensures that the architectural style can be used for
 4998 different IWSs as only the data used needs to change. This design choice encourages
 4999 reuse satisfying requirement *R4*. *Implement a flexible architecture that is adapt-*
5000 able to different IWSs and application contexts to facilitate reuse. We propose an
 5001 initial set of guidelines on how to create and update the benchmark dataset within
 5002 Section 11.6.3.1.

5003 11.4.1.2 Facade API

5004 An architectural ‘facade’ is the central component to our mitigation strategy for
 5005 monitoring and detecting for changes in called IWSs. The facade acts as a guarded
 5006 gateway to the intelligent service that defends against two key issues: (i) potential
 5007 shifts in model variations that power the cloud vendor services, and (ii) ensures that
 5008 a context-specific dataset specific to the application being developed is validated
 5009 *over time*. By using a facade we can return evolution-aware error codes to the client
 5010 application satisfying requirement *R1*. *Define a set of error conditions that specify*
5011 the types of evolution that occur for an intelligent service and enabling requiremen
5012 R3. *Monitor the evolution of IWSs for changes that affect the application’s behaviour*.
 5013 This works by ensuring every request made by the client application contains a valid
 5014 Behaviour Token (see Section 11.4.1.4) and will reject the request when evolution
 5015 has been identified by the Scheduler with an associated error code. The Facade API
 5016 essentially ‘blocks’ the client application out from accessing the intelligent service
 5017 when an invalid state has occurred.

Table 11.2: Rules encoded within a Behaviour Token.

Rule	Description
Max Labels	The value of n .
Min Confidence	The smallest acceptable value of c .
Max δ Labels	The minimum number of labels dropped or introduced from the current k_t and provided k_r to be considered a violation (i.e $ l(k_t) \Delta l(k_r) $).
Max δ Confidence	The minimum confidence change of <i>any</i> label from the current k_t and provided k_r to be considered a violation.
Expected Labels	A set of labels that every response must include.

5018 11.4.1.3 Threshold Tuner

5019 Selecting an appropriate threshold for detecting behavioural change depends on
 5020 the application context. Setting the threshold too low increases the likelihood of
 5021 incorrect results, while setting the threshold too high means undesired changes are
 5022 being detected. Our approach enables developers to configure these parameters
 5023 through a Threshold Tuner. *< todo: Notification for Threshy paper is 10 August;*
5024 therefore we can cite Threshy here before this paper's camera ready version is
5025 due on 10 September for providing greater detail on how the Threshold Tuner tool
5026 works etc. > This improves robustness as now there is a systematic approach for
 5027 monitoring and responding to incorrect thresholds. Configurable thresholds meet
 5028 our key requirements *R2* and *R3*.

5029 11.4.1.4 Behaviour Token

5030 The Behaviour Token stores the current state of the Proxy Server by encoding specific
 5031 rules regarding the evolution of the intelligent service. The current token (at time t)
 5032 held by the Proxy Server is denoted by k_t . These rules are specified by the developer
 5033 upon initialisation of this Proxy Server, and are presented in Table 11.2. When the
 5034 Proxy Server is first initialised (i.e., at $t = 0$), the first Behaviour Token is created
 5035 based on the Benchmark Dataset and its configuration parameters (Table 11.2) and
 5036 is stored locally (thus k_0 is created). The Behaviour Token is passed to the client
 5037 application to be used in subsequent requests to the proxy server, where k_r represents
 5038 the Behaviour Token passed from the client application to the proxy server. Each
 5039 time the proxy server receives the Behaviour Token from the client the validity of the
 5040 token is validated with a comparison to the Proxy Server's current behaviour token
 5041 (i.e., $k_r \equiv k_t$). An invalid token (i.e., when $k_r \neq k_t$) indicates that an error caused by
 5042 evolution has occurred and the application developer needs to appropriately handle
 5043 the exception. Behaviour Tokens are essential for meeting requirement *R3*. *Monitor*
 5044 *the evolution of IWSs for changes that affect the application's behaviour.*

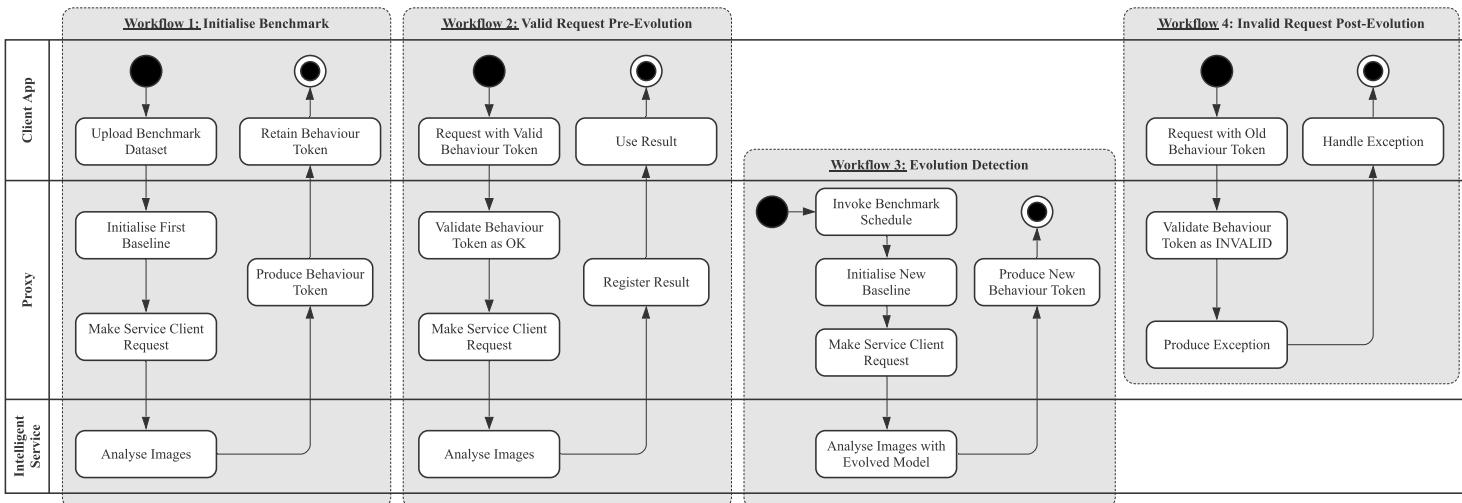


Figure 11.6: State diagram for the four workflows presented.

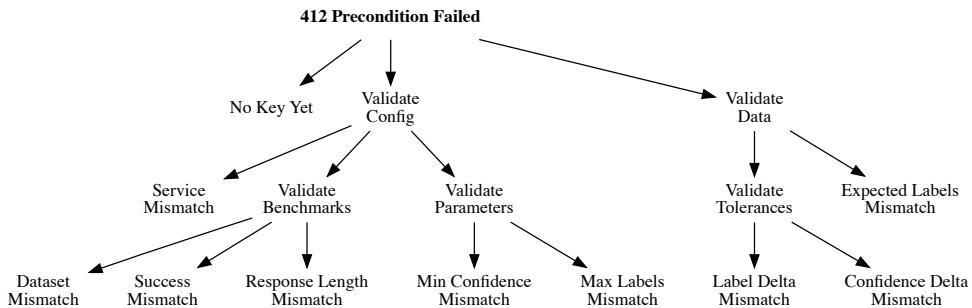


Figure 11.7: Precondition failure taxonomy; leaf nodes indicate error types returned to users.

5045 11.4.1.5 Service Client

5046 If any of the rules above are violated, then the response of the facade request will
 5047 vary depending on the parameter of the behaviour encoded within the behaviour
 5048 token. This can be one of:

- 5049 • **Error:** Where a HTTP non-200 code is returned by the facade to the client
 5050 application, indicating that the client application must deal with the issue
 5051 immediately;
- 5052 • **Warning:** Where a warning ‘callback’ endpoint is called with the violated
 5053 response to be dealt with, but the response is still returned to the client
 5054 application;
- 5055 • **Info:** Where the violated response is logged in the facade’s logger for the
 5056 developer to periodically read and inspect, and the response is returned to the
 5057 client application.

5058 We implement this Proxy Server pattern using HTTP conditional requests. As
 5059 we treat the Label as a first class citizen, we return the labels for a specific image
 5060 (r_i) only where the *Entity Tag* (ETag) or *Last Modified* validators pass. The k_r
 5061 is encoded within either the ETag (i.e., a unique identifier representing t) or as
 5062 the date labels (and thus models) were last modified (i.e., using the *If-Match*
 5063 or *If-Unmodified-Since* conditional headers). We note that the use of *weak*
 5064 ETags should be used, as byte-for-byte equivalence is not checked but only semantic
 5065 equivalence within the tolerances specified. Should t evolve to an invalid state
 5066 (i.e., k_r is no longer valid against k_t) then the behaviour as described above will be
 5067 enacted.

5068 These HTTP header fields are used as the ‘backbone’ to help enforce robustness
 5069 of the services against evolutionary changes and context within the problem domain
 5070 dataset. Responses from the service are forwarded to the clients when such rules
 5071 are met, otherwise alternative behaviour occurs. For example, the most severe of
 5072 violated erroneous behaviour is the ‘Error’ behaviour. To enforce this rule, we
 5073 advocate for use of the 412 Precondition Failed HTTP error if a violation
 5074 occurs, as a *If-** conditional header was violated. An example of this architectural
 5075 pattern with the ‘Error’ behaviour is illustrated in Figure 11.6.

5076 We suggest the 412 Precondition Failed HTTP error be returned in the
5077 event that a behaviour token is violated against a new benchmark. Further details
5078 outlining the reasons why a precondition has failed are encoded within a JSON
5079 response sent back to the consuming application. The following describes the
5080 two broad categories of possible errors returned: *robustness precondition failure*
5081 or *benchmark precondition failure*. These are illustrated in a high level within
5082 Figure 11.7 where leaf nodes are the potential error types that can be returned. A
5083 list of the different error codes are given in Table 11.1, where errors above the rule
5084 are robustness expectations (which check for basic requirements such as whether the
5085 key provided encodes the same data as the dataset in the facade) while those below
5086 are benchmark expectations (which identifies evolution cases).

5087 11.4.1.6 Scheduler

5088 The Scheduler is responsible for triggering the Evolution Detection Workflow (de-
5089 scribed in detail below in Section 11.4.2). Developers set the schedule to run in
5090 the background at regular intervals or to trigger if violations occur z times. The
5091 Scheduler is the component that enables our architectural style to identify called
5092 intelligent service software evolution and to notify the client applications that such
5093 evolution has occurred. Client applications can then respond to this evolution in a
5094 timely manner rather than wait for the system to fail, as in our motivating example.
5095 The Scheduler is necessary to satisfy our requirements $R2$ and $R3$.

5096 11.4.2 Usage Example

5097 We explain how developer Michelina, from our motivating example, would use
5098 our proposed solution to satisfy the requirements described in Section 11.2. Each
5099 workflow is presented in Figure 11.6. Only *Workflow 1 - Initialise Benchmark* is
5100 executed once, while the rest are cycled. The description below assumes Michelina
5101 has implemented the Proxy.

5102 11.4.2.1 Workflow 1. Initialise Benchmark

5103 The first task that Michelina has to do is to prepare and initialise the benchmark
5104 dataset within the Proxy Server. To prepare a representative dataset, Michelina needs
5105 to follow well established guidelines such as those proposed by Pyle. Michelina also
5106 needs to manually assign labels to each image before uploading the dataset to the
5107 Proxy along with the thresholds to use for detecting behavioural change. The full set
5108 of parameters that Michelina has to set are based on the rules shown in Table 11.2.
5109 Michelina cannot use the Proxy to notify her of evolution until a Benchmark Dataset
5110 has been provided. The Proxy then sends each image in the Benchmark Dataset to
5111 the intelligent service and stores the results. From these results, a Behaviour Token
5112 is generated which is passed back to the Client Application. Michelina uses this
5113 token in all future requests to the Proxy as the token captures the current state of the
5114 intelligent service.

5115 **11.4.2.2 Workflow 2. Valid Request Pre-Evolution**

5116 Workflow 2 represents the steps followed when the intelligent service is behaving as
5117 expected. Michelina makes a request to label an image to the Proxy using the token
5118 that she received when registering the Benchmark Dataset. The token is validated
5119 with the Proxy's current state token and then a request to label the image is made to
5120 the intelligent service if no errors have occurred. Results returned by the intelligent
5121 service are registered with the Proxy Server. Michelina can be confident that the
5122 result returned by our service is in line with her expectations.

5123 **11.4.2.3 Workflow 3. Evolution Detection**

5124 Workflow 3 describes how the Proxy functions when behavioural change is present
5125 in the called intelligent service. Michelina sets a schedule for once a day so that the
5126 Proxy's Scheduler triggers Workflow 3. First, each image in the Benchmark Dataset
5127 is sent to the intelligent service. Unlike, Workflow 1, we already have a Behaviour
5128 Token that represents the previous state of the intelligent service. In this case, the
5129 model behind the intelligent service has been updated and provides different results
5130 for the Benchmark Dataset. Second, the Proxy updates the internal Behaviour Token
5131 ready for the next request. At this stage Michelina will be notified that the behaviour
5132 of the intelligent service has changed.

5133 **11.4.2.4 Workflow 4. Invalid Request Post-Evolution**

5134 Workflow 4 provides Michelina with an error message when evolution has been
5135 detected. Michelina's client application makes a request to the Proxy Server with
5136 an old Behaviour Token. The Proxy Server then validates the client token which is
5137 invalid as the Behaviour Token has been updated. In this case, an exception is raised
5138 and an appropriate error message as discussed above is included in the response
5139 back to Michelina's client application. Michelina can code her application to handle
5140 each error class in appropriate ways for her domain.

5141 **11.5 Evaluation**

5142 Our evaluation of our novel intelligent service Proxy Server approach uses a technical
5143 evaluation based on the results of an observational study. We used existing datasets
5144 from observational studies [87, 214] to identify problematic evolution in computer
5145 vision labelling services. This technical evaluation is designed to show: (i) what
5146 the responses are with and without our architecture present (highlighting errors); (ii)
5147 the overall increased robustness using enhanced responses; and (iii) the technical
5148 soundness of the approach. Thus, we propose the following research question which
5149 we answer in Section 11.5.2: "*Can the architecture identify evolutionary issues of*
5150 *computer vision services via error codes?*" Based on our findings we proposed and
5151 implemented the Proxy Server using a Ruby development framework which we have
5152 made available online for experimentation.³ Additional data was collected from the

³<http://bit.ly/2TlMmDh> last accessed 5 March 2020.

5153 CVS and sent to the Proxy Server to evaluate how the service handles behavioural
5154 change.

5155 11.5.1 Data Collection and Preparation

5156 To minimise reviewer bias, we do not identify the name of the service used, however
5157 this service was one of the most adopted cloud vendors used in enterprise applications
5158 in 2018 [294]. The two existing datasets used [87, 214] consisted of 6,680 images.

5159 We initialised the benchmark (workflow 1) in November 2018, and sent each
5160 image to the service every eight days and captured the JSON responses through the
5161 facade API (workflow 2) until March 2019. This resulted in 146,960 JSON responses
5162 from the target CVS. We then selected the first and last set of JSON responses (i.e.,
5163 13,360 responses) and independently identified 331 cases of evolution of the original
5164 6,680 images. This was achieved by analysing the JSON responses for each image
5165 taken in using an evaluation script.⁴

5166 For each JSON response, evolution (as classified by Figure 11.2) was determined
5167 either by a vocabulary or confidence per label change in the first and last responses
5168 sent. For the 331 evolving responses, we calculated the delta of the label's confidence
5169 between the two timestamps and the delta in the number of labels recorded in the
5170 entire response. Further, for the highest-ranking label (by confidence), we manually
5171 classified whether its ontology became more specific, more generalised or whether
5172 there was substantial emphasis change. The distribution of confidence differences per
5173 these three groups are shown in Figure 11.8, with the mean confidence delta indicated
5174 with a vertical dotted line. This highlights that, on average, labels that change
5175 emphasis generally have a greater variation, such as the example in Figure 11.3.
5176 Further, we grouped each image into one of four broad categories—*food*, *animals*,
5177 *vehicles*, *humans*—and assessed the breakdown of ontology variance as provided
5178 in Table 11.3. We provide this dataset as an additional contribution and to permit
5179 replication.⁵ The parameters set for our initial benchmark were a delta label value of
5180 3 and delta confidence value of 0.01. Expected labels for relevant groups were also
5181 assigned as mandatory label sets (e.g., *animal* images used ‘animal’, ‘fauna’ and
5182 ‘organism’; *human* images used ‘human’ etc.).

5183 11.5.2 Results

5184 Examples of the March 2019 responses contrasting the proxy and direct service
5185 responses in our evaluation are shown in Figures 11.9 to 11.11. (Due to space limita-
5186 tions, the entire JSON response is partially redacted using ellipses.) These examples
5187 identify the label identified with the highest level of confidence in three examples
5188 against the ground truth label in the benchmark dataset. In total, the Proxy Server
5189 identified 1,334 labels added to the responses and 1,127 labels dropped, with, on
5190 average, a delta of 8 labels added. The topmost labels added were ‘architecture’
5191 at 32 cases, ‘building’ at 20 cases and ‘ingredient’ at 20 cases; the topmost
5192 labels dropped were ‘tree’ at 21 cases, ‘sky’ at 19 cases and ‘fun’ at 17 cases.

⁴<http://bit.ly/2G7saFJ> last accessed 2 March 2020.

⁵<http://bit.ly/2VQrAUU> last accessed 5 March 2020.

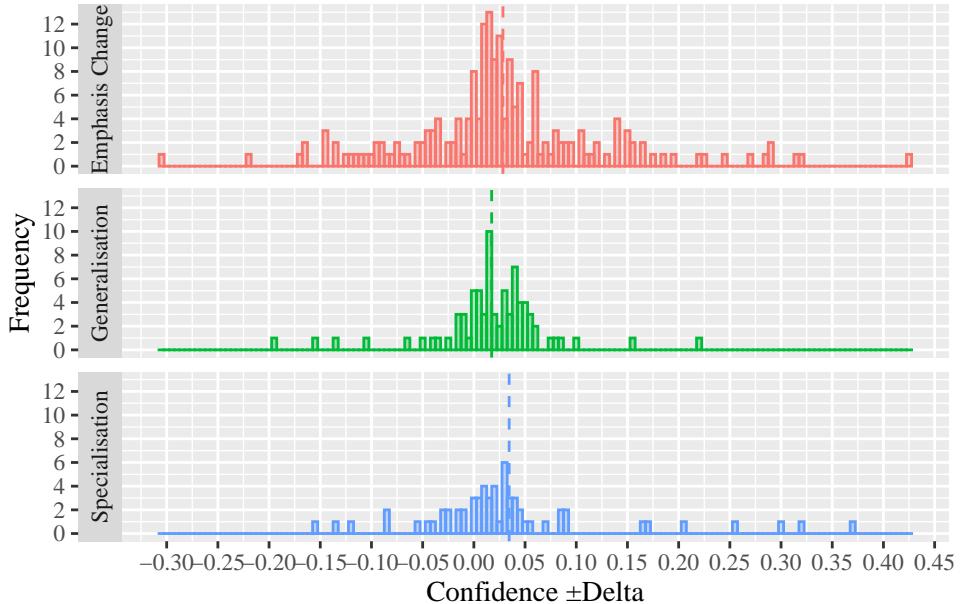


Figure 11.8: Histogram of confidence variation.

5193 1054 confidence changes were also observed by the Proxy Server, on average a delta
 5194 increase of 0.0977.

5195 In Figure 11.9, we highlight an image of a sheep that was identified as a ‘sheep’
 5196 (at 0.9622) in November 2018 and then a ‘mammal’ in March 2019. This evolution
 5197 was classified by the Proxy Server as a confidence change error as the delta in
 5198 the confidences between the two timestamps exceeds the parameter set of 0.01—in
 5199 this case, ‘sheep’ was downgraded to the third-ranked label at 0.9816, thereby
 5200 increasing by a value of 0.0194. As shown in the example, four other labels evolved
 5201 for this image between the two time stamps (‘herd’, ‘livestock’, ‘terrestrial
 5202 animal’ and ‘snout’) with an average increase of 0.1174 found. Such information
 5203 is encoded as a 412 HTTP error returned back to the user by the Proxy Server,
 5204 rejecting the request as substantial evolution has occurred, however the response
 5205 *directly* from the service indicates no error at all (indicating by a 200 HTTP response).

5206 Similarly, Figure 11.10 shows a violation of the number of acceptable changes in
 5207 the number of labels a response should have between two timestamps. In November
 5208 2018, the response includes the labels ‘car’, ‘motor vehicle’, ‘city’ and
 5209 ‘road’, however these labels are not present in the 2019 response. The response
 5210 in 2019 introduces ‘transport’, ‘building’, ‘architecture’, and ‘house’.
 5211 Therefore, the combined delta is 4 dropped and 4 introduced labels, exceeding our
 5212 threshold set of 3.

5213 Lastly, Figure 11.11 indicates an expected label failure. In this example, the
 5214 label ‘fauna’ was dropped in the 2018 label set, which was an expected label
 5215 of all animals we labelled in our dataset. Additionally, this particular response
 5216 introduced ‘green iguana’, ‘iguanidae’, and ‘marine iguana’ to its label
 5217 set. Therefore, not only was this response in violation of the label delta mismatch, it

Table 11.3: Variance in ontologies for the five broad categories.

Ontology Change	Food	Animal	Vehicles	Humans	Other	Total
Generalisation	8	13	11	8	38	78
Specialisation	5	12	1	1	43	62
Emphasis Change	18	4	10	21	138	191
Total	31	29	22	30	219	331

⁵²¹⁸ was also in violation of the expected labels mismatch error, and thus is caught twice
⁵²¹⁹ by the Proxy Server.

⁵²²⁰ 11.5.3 Threats to Validity

⁵²²¹ 11.5.3.1 Internal Validity

⁵²²² As mentioned, we selected a popular CVS provider to test our proxy server against.
⁵²²³ However, there exist many other CVSs, and due to language barriers of the authors,
⁵²²⁴ no non-English speaking service were selected despite a large number available from
⁵²²⁵ Asia. Further, no user evaluation has been performed on the architectural tactic so
⁵²²⁶ far, and therefore developers may suggest improvements to the approach we have
⁵²²⁷ taken in designing our tactic. We intend to follow this up with a future study.

⁵²²⁸ 11.5.3.2 External Validity

⁵²²⁹ This paper only evaluates the object detection endpoint of a computer vision-based
⁵²³⁰ intelligent service. While this type of intelligent service is one of the more mature
⁵²³¹ AI-based services available on the market—and is largely popular with develop-
⁵²³² ers [90]—further evaluations of the our tactic may need to be explored against other
⁵²³³ endpoints (i.e., object localisation) or, indeed, other types of services, such as natural
⁵²³⁴ language processing, audio transcription, or on time-series data. Future studies may
⁵²³⁵ need to explore this avenue of research.

⁵²³⁶ 11.5.3.3 Construct Validity

⁵²³⁷ The evaluation of our experiment was largely conducted under clinical conditions,
⁵²³⁸ and a real-world case study of the design and implementation of our proposed tactic
⁵²³⁹ would be beneficial to learn about possible side-effects from implementing such a
⁵²⁴⁰ design (e.g., implications to cost etc.). Therefore, our evaluation does not consider
⁵²⁴¹ more practical considerations that a real-world, production-grade system may need
⁵²⁴² to consider.



Label: Animal
Nov 2018: 'sheep' (0.9622)
Mar 2019: 'mammal' (0.9890)
Category: Confidence Change

Intelligent Service Response in March 2019

```

1 { "responses": [ { "label_annotations": [
2   { "mid": "/m/04rky",
3     "description": "mammal",
4     "score": 0.9890478253364563,
5     "topicality": 0.9890478253364563 },
6   { "mid": "/m/09686",
7     "description": "vertebrate",
8     "score": 0.9851104021072388,
9     "topicality": 0.9851104021072388 },
10  { "mid": "/m/07bgp",
11    "description": "sheep",
12    "score": 0.9815810322761536,
13    "topicality": 0.9815810322761536 },
14    ... ] } ]

```

Proxy Server Response in March 2019

```

1 { "error_code": 8,
2   "error_type": "CONFIDENCE_DELTA_MISMATCH",
3   "error_data": {
4     "source_key": { ... },
5     "source_response": { ... },
6     "violating_key": { ... },
7     "violating_response": { ... },
8     "delta_confidence_threshold": 0.01,
9     "delta_confidences_detected": {
10       "sheep": 0.01936030388219212,
11       "herd": 0.15035879611968994,
12       "livestock": 0.13112884759902954,
13       "terrestrial animal": 0.1791478991508484,
14       "snout": 0.10682523250579834
15     },
16     "uri": "http://localhost:4567/demo/data/000000005992.jpeg"
17     ↵ ,
      "reason": "Exceeded confidence delta threshold ±0.01 in 5
      ↵ labels (delta mean=+0.1174)." } }

```

Figure 11.9: Example of substantial confidence change due to evolution.



Label: Vehicle
Nov 2018: 'vehicle' (0.9045)
Mar 2019: 'motorcycle' (0.9534)
Category: Label Set Change

Intelligent Service Response in March 2019

```

1 { "responses": [ { "label_annotations": [
2   { "mid": "/m/07yv9",
3     "description": "vehicle",
4     "score": 0.9045347571372986,
5     "topicality": 0.9045347571372986 },
6   { "mid": "/m/07bsy",
7     "description": "transport",
8     "score": 0.9012271165847778,
9     "topicality": 0.9012271165847778 },
10  { "mid": "/m/0dx1j",
11    "description": "town",
12    "score": 0.8946694135665894,
13    "topicality": 0.8946694135665894 },
14    ... ] } ] }
```

Proxy Server Response in March 2019

```

1 { "error_code": 7,
2   "error_type": "LABEL_DELTA_MISMATCH",
3   "error_data": {
4     "source_key": { ... },
5     "source_response": { ... },
6     "violating_key": { ... },
7     "violating_response": { ... },
8     "delta_labels_threshold": 5,
9     "delta_labels_detected": 8,
10    "uri": "http://localhost:4567/demo/data/000000019109",
11    "new_labels": [ "transport", "building", "architecture", "
12      ↪ house" ],
13    "dropped_labels": [ "car", "motor vehicle", "city", "road"
14      ↪ ],
15    "reason": "Exceeded label count delta threshold ±5 (4 new
16      ↪ labels + 4 dropped labels = 8)." } }
```

Figure 11.10: Example of substantial changes of a response's label set due to evolution.



Label: Fauna
Nov 2018: 'reptile' (0.9505)
Mar 2019: 'iguania' (0.9836)
Category: Ontology Specialisation

Intelligent Service Response in March 2019

```

1 | { "responses": [ { "label_annotations": [
2 |   { "mid": "/m/08_jw6",
3 |     "description": "iguania",
4 |     "score": 0.9835183024406433,
5 |     "topicality": 0.9835183024406433 },
6 |   { "mid": "/m/06bt6",
7 |     "description": "reptile",
8 |     "score": 0.9833670854568481,
9 |     "topicality": 0.9833670854568481 },
10 |   { "mid": "/m/01vq7_",
11 |     "description": "iguana",
12 |     "score": 0.9796721339225769,
13 |     "topicality": 0.9796721339225769 },
14 |   ... ] } ]

```

Proxy Server Response in March 2019

```

1 | { "error_code": 9,
2 |   "error_type": "EXPECTED_LABELS_MISMATCH",
3 |   "error_data": {
4 |     "source_key": { ... },
5 |     "violating_response": { ... },
6 |     "uri": "http://localhost:4567/demo/data/0052",
7 |     "expected_labels": [ "fauna" ],
8 |     "labels_detected": [ "iguana", "green iguana", "iguaniidae"
9 |       ↪ , "lizard", "scaled reptile", "marine iguana", "
10 |       ↪ terrestrial animal", "organism" ],
      "labels_missing": [ "fauna" ],
      "reason": "The expected label(s) `fauna' are missing in
        ↪ the response." } }

```

Figure 11.11: Example of an expected label missing due to evolution.

5243 11.6 Discussion

5244 11.6.1 Implications

5245 11.6.1.1 For cloud vendors

5246 Cloud vendors that provide IWSs may wish to adopt the architectural tactic presented
5247 in this paper by providing a proxy, auxiliary service (or similar) to their existing ser-
5248 vices, thereby improving the current robustness of these services. Further, they
5249 should consider enabling developers of this technical domain knowledge by pre-
5250 venting client applications from using the service without providing a benchmark
5251 dataset, such that the service will return HTTP error codes. These procedures should
5252 be well-documented within the service's API documentation, thereby indicating to
5253 developers how they can build more robust applications with their IWSs. Lastly,
5254 cloud vendors should consider updating the internal machine learning models less
5255 frequently unless substantial improvements are being made. Many different appli-
5256 cations from many different domains are using these IWSs so it is unlikely that
5257 the model changes are improving all applications. Versioned endpoints would help
5258 with this issue, although—as we have discussed—context using benchmark datasets
5259 should be provided.

5260 11.6.1.2 For application developers

5261 Developers need to monitor all IWSs for evolution using a benchmark dataset and
5262 application specific thresholds before diving straight into using them. It is clear that
5263 the evolutionary issues have significant impact in their client applications [87], and
5264 therefore they need to check the extent this evolution has between versions of an
5265 intelligent service (should versioned APIs be available). Lastly, application devel-
5266 opers should leverage the concept of a proxy server (or other form of intermediary)
5267 when using IWSs to make their applications more robust.

5268 11.6.1.3 For project managers

5269 Project managers need to consider the cost of evolution changes on their application
5270 when using IWSs, and therefore should schedule tasks for building maintenance
5271 infrastructure to detect evolution. Consider scheduling tasks that evaluates and
5272 identifies the frequency of evolution for the specific intelligent service being used.
5273 Our research we have found some IWSs that are not versioned but rarely show
5274 behavioural changes due to evolution.

5275 11.6.2 Limitations

5276 In the situation where a solo developer implements the Proxy Service the main
5277 limitation is the cost vs response time trade-off. Developers may want to be notified
5278 as soon as possible when a behavioural change occurs which requires frequent
5279 validation of the Benchmark Dataset. Each time the Benchmark Dataset is validated
5280 each item is sent as a request to the intelligent service. As cloud vendors charge

5281 per request to an intelligent service there are financial implications for operating
5282 the Proxy Service. If the developer optimises for cost then the application will take
5283 longer to respond to the behavioural change potentially impact end users. Developers
5284 need to consider the impact of cost vs response time when using the Proxy Service.

5285 Another limitation of our approach is the development effort required to imple-
5286 ment the Proxy Service. Developers need to build a scheduling component, batch
5287 processing pipeline for the Benchmark Dataset, and a web service. These com-
5288 ponents require developing and testing which impact project schedules and have
5289 maintenance implications. Thus, we advise developers to consider the overhead of
5290 a Proxy Service and way up the benefits with have incorrect behaviour caused by
5291 evolution of IWSs.

5292 11.6.3 Future Work

5293 11.6.3.1 Guidelines to construct and update the Benchmark Dataset

5294 Our approach assumes that each category of evolution is present in the Benchmark
5295 Dataset prepared by the developer. Further guidelines are required to ensure that the
5296 developer knows how to validate the data before using the Proxy Service. While the
5297 focus of this paper was to present and validate our architectural tactic, guidelines
5298 on how to construct and update benchmark datasets for this tactic will need to be
5299 considered in future work. Data science literature extensively covers dataset prepara-
5300 tion (e.g., [199, 283]), and many example benchmark datasets are readily available
5301 [24, 145, 370]. An initial set of guidelines are proposed as follows: data must be
5302 contextualised and appropriately sampled to be representative of the client applica-
5303 tion in particular the patterns present in the data, contain both positive and negative
5304 examples (this is/is not a cat); where to source data from (existing datasets, Google
5305 Images/Flickr, crowdsourced etc.); whether the dataset is synthetically generated to
5306 increase sample size; and how large a benchmark dataset size should be (i.e., larger
5307 the better but takes more effort and costs more). Benchmark datasets can also be
5308 used by software engineers provided the domain and context is appropriate for their
5309 specific application’s context. Software engineers also benefit from our approach
5310 even if these guidelines are not strictly adhered to provided they use an application-
5311 specific dataset (i.e., data collected from the input source for their application). The
5312 main reason for this is that without our proposed tactic there are limited ways to
5313 build robust software with intelligent services. Future testing and evaluation of these
5314 guidelines should be considered.

5315 11.6.3.2 Extend the evolution categories to support other IWSs

5316 This paper has used computer vision services to assess our proposed tactic, and
5317 therefore further investigation is needed into the evolution characteristics of other
5318 IWSs. The evolution challenges with services that provide optimisation algorithms
5319 such as route planning are likely to differ from CVSs. These characteristics of an
5320 application domain have shown to greatly influence software architecture [25] and
5321 further development of the Proxy Service will need to account for these differences.

5322 As an example, we have identified many similar issues that exist for natural language
 5323 processing, where topic modelling produces labels on large bodies of text with
 5324 associated confidences. Therefore, the *broader* concepts of our contribution (e.g.,
 5325 behaviour token parameters, error codes etc.) can be used to handle issues in natural
 5326 language processing to demonstrate the generalisability of the architecture to other
 5327 intelligent services. We plan to apply our tactic to natural language processing and
 5328 other intelligent services in our future work.

5329 *11.6.3.3 Provide tool support for optimising parameters for an application context*

5330 Appropriately using the Proxy Service requires careful selection of thresholds,
 5331 benchmark rules and schedule. Further work is required to support the developer
 5332 in making these decisions so an optimal application specific outcome is achieved.
 5333 One approach is a to present the trade-offs to the developer and let them visualise
 5334 the impact of their decisions.

5335 *11.6.3.4 Improvements for a more rigorous approach*

5336 Conducting a more formal evaluation of our proposed architecture would benefit
 5337 the robustness of the solution presented. This could be done in various ways,
 5338 for example, using a formal architecture evaluation method such as ATAM [188]
 5339 or a similar variant [51]; conducting user evaluation via brainstorming sessions or
 5340 interviews with practitioners who may provide suggestions to improve our approach;
 5341 determining better strategies to fully-automate the approach and reduce manual steps;
 5342 and using real-world industry case studies to identify other factors such as cost and
 5343 maintenance issues. All these are various avenues of research that would ultimately
 5344 benefit in a more well-rounded approach to the architectural tactic we have proposed.

5345 **11.7 Related Work**

5346 *11.7.0.1 Robustness of Intelligent Services*

5347 While usage of IWSs have been proven to have widespread benefits to the commu-
 5348 nity [94, 289], they are still largely understudied in software engineering literature,
 5349 particularly around their robustness in production-grade systems. As an example,
 5350 advancements in computer vision (largely due to the resurgence of convolutional
 5351 neural networks in the late 1990s [208]) have been made available through IWSs and
 5352 are given marketed promises from prominent cloud vendors, e.g., “with Amazon
 5353 Rekognition, you don’t have to build, maintain or upgrade deep learning pipelines”.⁶
 5354 However, while vendors claim this, the state of the art of *computer vision itself*
 5355 is still susceptible to many robustness flaws, as highlighted by many recent stud-
 5356 ies [113, 302, 357]. Further, each service has vastly different (and incompatible)
 5357 ontologies which are non-static and evolve [87, 260], certain services can mislabel
 5358 images when as little as 10% noise is introduced [161], and developers have a shallow

⁶<https://aws.amazon.com/rekognition/faqs/>, accessed 21 November 2019.

5359 understanding of the fundamental AI concepts behind these issues, which presents a
5360 dichotomy of their understanding of the technical domain when contrasted to more
5361 conventional domains such as mobile application development [90].

5362 *11.7.0.2 Proxy Servers as Fault Detectors*

5363 Fault detection is an availability tactic that encompasses robustness of software [31].
5364 Our architecture implements the sanity check and condition monitoring techniques
5365 to detect faults [31, 167], by validating the reasonableness of the response from the
5366 intelligent service against the conditions set out in the rules encoded in the benchmark
5367 dataset and behaviour token. As we do in this study, the proxy server pattern can be
5368 used to both detect and action faults in another service as an intermediary between a
5369 client and a server. For example, addressing accessibility issues using proxy servers
5370 has been widely addressed [41, 42, 341, 380] and, more recently, they have been
5371 used to address in-browser JavaScript errors [108].

5372 **11.8 Conclusions**

5373 IWSs are gaining traction in the developer community, and this is shown with
5374 an evermore growing adoption of CVSSs in applications. These services make
5375 integration of AI-based components far more accessible to developers via simple
5376 RESTful APIs that developers are familiar with, and offer forever-‘improving’ object
5377 localisation and detection models at little cost or effort to developers. However, these
5378 services are dependent on their training datasets and do not return consistent and
5379 deterministic results. To enable robust composition, developers must deal with the
5380 evolving training datasets behind these components and consider how these non-
5381 deterministic components impact their deterministic systems.

5382 This paper proposes an integration architectural tactic to deal with these issues
5383 by mapping the evolving and probabilistic nature of these services to deterministic
5384 error codes. We propose a new set of error codes that deal directly with the erroneous
5385 conditions that has been observed in IWSs, such as computer vision. We provide
5386 a reference architecture via a proxy server that returns these errors when they are
5387 identified, and evaluate our architecture, demonstrating its efficacy for supporting
5388 IWS evolution. Further, we provide a labelled dataset of the evolutionary patterns
5389 identified, which was used to evaluate our architecture.

5390

Part III

5391

Postface

CHAPTER 12

5392

5393

5394

Conclusions & Future Work

5395

5396 In this chapter, we provide a summary of the contributions within the body of
5397 this work. We evaluate the significance of the research outcomes to the software
5398 engineering research community and identify potential criticisms of these outcomes.
5399 Lastly, we indicate future avenues of research resulting from this thesis and provide
5400 concluding remarks.

5401 12.1 Contributions of this Work

5402 This thesis has presented three primary contributions to the body of software engi-
5403 neering knowledge. Namely, we have presented an improved understanding in the
5404 landscape of IWSs—concretely, those that provide computer vision—by examining
5405 their runtime behaviour and evolution profile over a longitudinal study (Chapter 4).
5406 The implications of this work emphasise the caution developers need to take be-
5407 fore diving deep into using these services, and highlight the substantial impacts to
5408 software quality if these considerations are ignored. We showed that developers
5409 find working with this software more frustrating when contrasted to conventional
5410 software engineering domains (Chapter 6), and that the distribution of the types of
5411 issues they face differs from that of the types of issues developers face in established
5412 areas such as mobile and web development (Chapter 5). Furthermore, developers
5413 find the completeness of the existing CVS API documentation poor (Chapter 5),
5414 and therefore an investigation into the attributes of what *constitutes* a complete API
5415 document according to literature and how developers respond to the efficacy of these
5416 attributes produced a taxonomy that, when applied to three CVS service providers,
5417 found 12 areas of improvement of the services’ documentation (Chapter 8). This
5418 taxonomy further serves as a go-to ‘checklist’ for any software engineer to review a
5419 prioritised list of documentation elements worth implementing into their own API
5420 documentation. Lastly our investigations into improved intelligent service integra-
5421 tion architectures proposes several strategies by which developers can guard against

the non-deterministic evolutionary issues found in Chapter 4. Preliminary solutions such as that presented in Chapter 9 helped informed further investigations into how developers can use a novel workflow to better select appropriate confidence thresholds calibrated for their application’s domain (Chapter 10) and prevent evolution evident in CVSs via a client-server intermediary proxy server strategy (Chapter 11). A more extensive discussion into the contributions of this thesis is presented in Section 1.7.

12.1.1 Answers to Research Questions

In this subsection, we directly answer the four primary research questions that were posed in Section 1.4.

12.1.1.1 RQ1: “What is the nature of cloud-based CVSs?”

These services are in nascent stage, are difficult to evaluate, and are not easily interchangeable. They present themselves as conceptually similar, but we find they functionally differ between vendors. Their labels are semantically disparate and work needs to be done on consolidating a standardised vocabulary for labels. Evolution within these services occurs and is not sufficiently versioned or documented to developers as results from services are non-static.

Irrespective of which service is used, the vocabulary used to label an image is disparate. We find that **there exists no common standard vocabulary** (e.g., ‘border collie’ vs. ‘collie’) and **semantic consistency for the same image between services is disparate**, for example as that shown in Figure 12.1 (left). The runtime behaviour of these services when contrasted against *each other* is, therefore, inconsistent, and thus (without semantic comparison of images, such as that suggested in Chapter 9) the vendors are not ‘plug-and-play’. In contrast to deterministic web services, the same result is functionally guaranteed despite which service is used. For instance, conceptually, a cloud storage service will provide the same output for the same input; that is, regardless of whether a developer uses AWS or Google Cloud object storage, when they upload a file, that file is (more or less) guaranteed to be stored. A deterministic input/output is, thereby, conceptually and functionally guaranteed. However, we find that the nature of intelligent services are conceptually similar but functionally different between services, and therefore developers are likely to become vendor locked. For instance, as we show in Section 4.5.1, one service may return the duplicity of objects in an image (e.g., ‘several’), while another service may return the subject of the image (e.g., ‘carrot’) or a hypernym of that subject (e.g., ‘food’), and another service may focus on the environment of the image (e.g., ‘indoors’). Further, even when a label is consistent between services, we find the consistency of how well they agree to that result—as measured by their confidence score in the label—does not always strictly match in their level of agreement. As

5454 we show in Figure 12.1 (right), **distributions of agreement can be disparate even**
5455 **where services agree on a label for the same image.** Lastly, while intelligent
5456 services that provide computer vision are somewhat stable in the responses they
5457 return, **their responses are non-static.** There is no guarantee that a request with
5458 the same image sent in testing will return the same response, and we find that this
5459 potential evolution risk is not sufficiently communicated to developers.



Figure 12.1: *Left:* Semantic consistency between services is not always guaranteed. Two services identified this image as ‘people’, while another identified ‘conversation’, which is not registered at all as a possible label from the other two services. *Right:* Even when services agree on a label, the distribution of their level of agreement is (at times) inconsistent—in the above image, ‘food’ is detected at confidence levels of three services ranging from 94.93% to 41.39%.

5460 12.1.1.2 *RQ2: “Are CVS APIs sufficiently documented?”*

☞ *These services are largely well-documented, but areas of improvement can be identified. By applying the five-dimensional taxonomy we propose in Chapter 8 to three services, we found there to be twelve ways vendors can better improve their services’ documentation. We found the ways in which developers can use these services for their purposes could be improved—such as improved tutorials that integrate multiple components of the service—and by providing better descriptions to improve developers’ conceptual understanding of computer vision.*

5461 To understand if these services are sufficiently documented, we first investigated
5462 what constitutes a complete API document, investigating literature and validating
5463 this against developers using a survey. These consist of five dimensions: usage
5464 description (or *how* developers can use the API); design rationale (or *when* the de-
5465 velopers should use it); domain concepts (or *why* developers should use it in their
5466 application domain); support artefacts (or *what* additional documentation could be
5467 provided to support developers); and, documentation presentation (or *visualisation*
5468 of the prior four dimensions). This taxonomy is presented with further detail in Ap-
5469 pendix C. **Developers and literature agree code snippets are the most important**
5470 **documentation artefact, followed closely by tutorials and low-level reference**

documentation. When we apply this taxonomy to intelligent services such as CVSSs, we find that there can be improvements made to all dimensions except documentation presentation, which is sufficient. **The largest suggested improvements fall into the usage description dimension**, in which quick-start guides, step-by-step tutorials, reference applications, best-practices, listings of all API components, minimum system dependencies, and installation instructions require further detail. The second largest dimension falls into the domain concepts behind computer vision, where vendors should provide a greater emphasis behind computer vision concepts and definitions of relevant computer vision terminology (especially since many vendors refer to the same concept under different terms, such as ‘image tagging’ and ‘label detection’ for what is essentially object recognition). The lack of complete documentation in domain concepts was further reflected in developer discussions on Stack Overflow, as found in Chapter 5. Section 8.6.5 details these suggested improvements in greater detail.

12.1.1.3 RQ3: “Are CVSS more misunderstood than conventional software engineering domains?”

↳ In conventional software engineering domains, where the technical domain is well-established and well-understood by developers, questions asked by developers are of greater depth. In contrast, their shallow understanding of the technical domain of computer vision is reflected by questions that highlight a poor understanding of the behaviour of these services and the contexts by which they work. Thus, simpler questions are asked, such as help with trying to understand basic error codes, or clarification of basic concepts and terminologies in computer vision. Therefore, we argue that they are more misunderstood seeing as the domain of intelligent services is still immature.

As expressed on Stack Overflow, we find developers struggle most with simple debugging issues, which reflects a shallow understanding of the AI concepts that empower these services. **The technical nuances become so abstracted away that developers begin to lack a full appreciation of the context and proper usage of these systems.** These questions reveal how developers do not have a strong grasp of the behaviour of these services and how further functional capability needs to be overcome by secondary phases of work, such as pre- and post-processing. **Their conceptual understanding of these services are poor**, with our findings suggesting that developers present a misunderstanding of the vocabulary used within computer vision, such as the differences between object and facial detection, localisation and recognition. The lack of strong conceptual understanding also reflects in discrepancy-based issues where developers cannot appreciate why services result in specific outcomes contrary to what they believe should happen. **We find these discrepancy-based issues to be the most frustrating for developers**, and argue that this is rooted in a need for developers to have some basic understanding of computer

vision before diving into services such as these. In terms of the documentation of these services, **developers express frustration towards the completeness of the documentation**, whereby they seek additional information from the official documentation sources but are unable to find anything to help resolve this gap. Further, **they question the accuracy of the cloud documentation since it is in contrast with the behaviour they observe**, as related to the discrepancy-based issues they find. In contrast to more established domains, such as mobile and web-development, the distribution of issues are different (see Figure 12.2). Rather than trying to interpret simple errors (as is the case for CVSs), developers question API usage and high-level conceptual questions. Developers have a greater appreciation for the technical domain in these mature areas, resulting in fewer shallower questions asked.

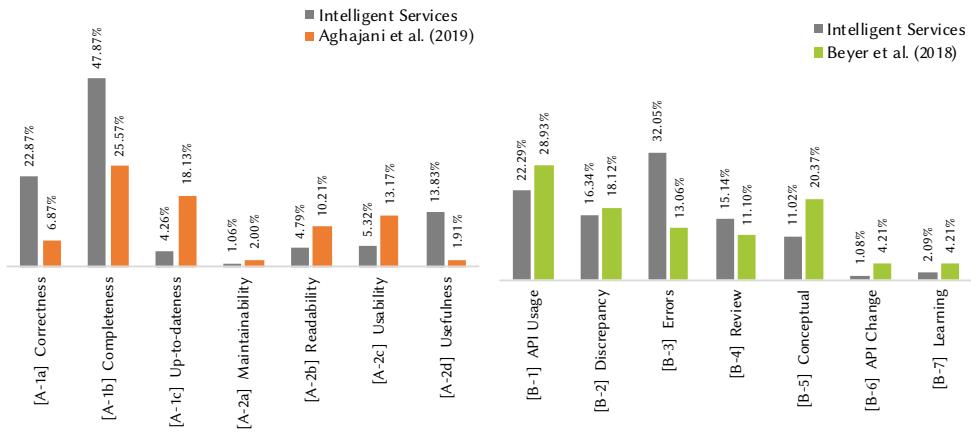


Figure 12.2: The distribution of documentation-specific questions (*left*) and generalised questions (*right*) differs between prior studies. Descriptions of each category for both question types are found in Table 5.1.

12.1.1.4 *RQ4: “What strategies can developers employ to integrate their applications with CVSs while preserving robustness and reliability?”*

☞ Developers can employ the use of a facade-based architecture to merge the responses of multiple vendors using a novel, proportional-representation based approach using lexical databases to resolve ontological issues of labels. An integration strategy consisting of four workflows was presented in Chapter 11 to assist developers monitor and handle substantial evolution change in these services. Developers can deal with the probabilistic nature of these services by using a representative data set of their application’s data to fine-tune a confidence threshold and monitor threshold changes in a production setting.

This thesis offers three strategies targeted at improved integration of developer applications with CVSs. Chapter 9 successfully demonstrated that multiple services

can be combined using lexical databases to better improve the reliability of relying on a single service's label. Further, this strategy outperformed naive merge methods using a novel proportional representation method by 0.015 F-measure. This strategy uses the idea of a client-server intermediary facade to handle these operations and produce a consistent result regardless of which service is being used. This inspired further work presented in Chapter 11. To handle the evolutionary issues found in the services, we developed a novel integration architecture based on the proxy server strategy, integrating four key proposed workflows which can be used to guard against evolution and non-determinism in these services: (i) initialising a representative benchmark of domain-specific data used in the client application; (ii) validating that the service is behaving as expected against that benchmark; (iii) periodically detecting for evolution if behavioural change occurs, thereby notifying change; and lastly (iv) invalidating future requests if substantial evolution is detected in step (iii). This, in turn, resolves a non-deterministic response into a deterministic error when evolution is raised. Lastly, to deal with the uncertainty arising from probabilistic confidence values, we proposed Threshy (see Chapter 10), a tool to help developers select appropriate threshold boundaries resulting from their benchmark data sets and cost factors (due to missed predictions). Ultimately these strategies aim at improving the robustness of applications that are dependent on CVSs.

12.1.2 Limitations to Research Answers & Future Research

Throughout this thesis, we have used computer vision as a primary exemplar of intelligent AI components provided via the web. Limiting this research to such a narrow scope is an illustrative example that enables more concrete findings and potential solutions to a specific subset of intelligent web services (IWSs). As discussed in Section 1.2, these particular type of IWSs were selected due to both their increasing enthusiasm and uptake in developer communities (see Figure A.3) and their maturity in the area. However, we acknowledge that there are myriad domains in the IWS space, such as audio and sentiment analysis, text-to-speech and speech-to-text, natural language processing, or time-series data analysis. Our analyses of CVSs chiefly targets content analysis (or object detection) endpoints of these services; other endpoints such as image description or object localisation exist, and were not considered as the main unit of analysis in this work. Further, this thesis selects only three prominent vendors of CVSs: Google, Microsoft and Amazon. While these vendors are considered to be the ubiquitous ‘go-to’ providers for cloud-based services (given their AWS, Google Cloud and Azure platforms) and were the most adopted for enterprise solutions [294], many other providers of computer vision intelligence exist [383, 399, 400, 401, 407, 420, 421, 423, 472, 473], including those from Asian market [397, 398, 419, 438, 439] where language barriers prevented analysis of these services.

Thus, the generalisability of our findings are a substantial threat to the external validity of our research answers and future research needs to investigate both other areas of IWSs to assess whether our findings and solutions are applicable to other intelligent domains and other types of services in the CVS market. Further, this

5560 thesis strongly emphasises investigations into identifying issues within web-based
5561 intelligent services. We establish a better understanding on their nature and run-
5562 time behaviour (RQ1), how they are documented (RQ2), and how well they are
5563 understood by developers (RQ3), but only offer limited solutions to these issues
5564 (e.g., RQ4). We encourage the software engineering community to use the issues
5565 identified in this work as a stepping-stone into future solutions, identifying other
5566 ways (beyond improved integration techniques) in which developers can handle
5567 these issues. For example, the broader concepts of our contributed architecture (e.g.,
5568 use of a behaviour token, its parameters, and the error codes proposed) can be shifted
5569 to handle issues in natural language processing to demonstrate the generalisability of
5570 the architecture to other intelligent services, since topic modelling produces labels
5571 with confidences and the approach can be largely transferred to this area.

5572 Other future work stemming from this thesis would be to explore the nature of
5573 other IWSs and understanding if similar evolution and behavioural runtime patterns
5574 exist with their computer vision equivalents (as identified in this thesis). Chiefly,
5575 future work on how to better support developers using different types of intelligent
5576 components would be an interesting area to explore, especially in applying our
5577 design strategies to combat the robustness issues we have identified to these other
5578 types of services and identify any potential pitfalls of our design. As our proposed
5579 architectural usage framework is a preliminary design, rigorous testing in real-
5580 world scenarios, such as a long-term industry case study implementing our design
5581 or conducting formal architecture evaluations such as ATAM [188], would be a
5582 possible avenue of research to verify the design. Further, our proposal makes use
5583 of the benchmark data set approach, but we are yet to explore and test potential
5584 guidelines in developing a benchmark data set. While we provide some potential
5585 guidelines in Section 11.6.3.1, these will need to be evaluated for practical use.

5586 Another key aspect would revolve around the documentation contributions of
5587 this study and investigating whether our suggested documentation improvements
5588 are applicable to these different services. Developing improved documentation and
5589 tooling that better support developers when using these IWSs (and how our proposed
5590 architecture fits in) should be explored.

5591 Moreover, since we find these services to be not yet as matured as traditional
5592 software development domains and—like similar emerging software engineering
5593 domains such as web development in the mid-1990s and early-2000s or mobile
5594 development from the mid-2000s to early-2010s—we suspect there to be substantial
5595 growth in the understanding of how we will use these services and maturity in the
5596 developer’s appreciation of its surrounding technical domain. Therefore, it would be
5597 beneficial to repeat some of the studies within this thesis and assess whether there is
5598 an improved understanding of the phenomena occurring within IWSs and whether
5599 developers have a developed mindset of these services and how they can be used.
5600 Thus, different tools, designs or suggestions may result from repetitional studies 5-10
5601 years in the future. This, therefore, identifies evolution in the *maturity* of intelligent
5602 services, and to highlight whether developers are showing a stronger understanding
5603 of the surrounding technical domain behind these services. We strongly encourage
5604 the software engineering community to explore these in such time to identify maturity

5605 in this emerging domain.

5606 12.2 Concluding Remarks

5607 To our knowledge, little prior investigation has been conducted to understand IWSs
5608 via the lenses of software quality—primarily the robustness, reliability of the services
5609 and completeness of its documentation. In this thesis, we have shown that the non-
5610 deterministic and probabilistic properties of computer vision IWSs present non-
5611 trivial impacts to the quality of software that they are integrated with, and it is
5612 pivotal that developers have a greater appreciation of the technical domain behind
5613 the AI techniques that empower such services.

5614 In identifying evolutionary and run-time issues of these services, the ways in
5615 which they are (currently) documented and these issues communicated (or not!), and
5616 analysing how developers perceive these services with a deterministic mindset, we
5617 have shown just how fragile the use of such services (as they stand) are. We strongly
5618 encourage vendors to use suggestions made within this research to improve both
5619 their documentation and their integration strategies so that developers can ensure
5620 more robust applications when using these services. Ultimately, intelligent AI
5621 components are still in a nascent stage, and therefore strongly suggest one message
5622 to eager developers: use with caution and be aware of the consequences!

5623

5624

References

5625

- 5626 [1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving,
5627 M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker,
5628 V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: A system for large-
5629 scale machine learning,” in *Proceedings of the 12th USENIX Symposium on Operating Systems
5630 Design and Implementation*. Savannah, GA, USA: ACM, 2016. ISBN 978-1-93-197133-1 pp.
5631 265–283.
- 5632 [2] R. Abdalkareem, E. Shihab, and J. Rilling, “What Do Developers Use the Crowd For?
5633 A Study Using Stack Overflow,” *IEEE Software*, vol. 34, no. 2, pp. 53–60, 2017,
5634 DOI 10.1109/MS.2017.31.
- 5635 [3] E. Aghajani, C. Nagy, G. Bavota, and M. Lanza, “A Large-scale empirical study on linguistic
5636 antipatterns affecting apis,” in *Proceedings of the 34th International Conference on Software
5637 Maintenance and Evolution*. Madrid, Spain: IEEE, September 2018. DOI 10.1109/IC-
5638 SME.2018.00012. ISBN 978-1-53-867870-1 pp. 25–35.
- 5639 [4] E. Aghajani, C. Nagy, O. L. Vega-Marquez, M. Linares-Vasquez, L. Moreno, G. Bavota,
5640 and M. Lanza, “Software Documentation Issues Unveiled,” in *Proceedings of the 41st Interna-
5641 tional Conference on Software Engineering*. Montreal, QC, Canada: IEEE, May 2019.
5642 DOI 10.1109/ICSE.2019.00122. ISBN 978-1-72-810869-8. ISSN 0270-5257 pp. 1199–1210.
- 5643 [5] M. Ahsanuzzaman, M. Asaduzzaman, C. K. Roy, and K. A. Schneider, “Classifying stack
5644 overflow posts on API issues,” in *Proceedings of the 25th International Conference on
5645 Software Analysis, Evolution and Reengineering*. Campobasso, Italy: IEEE, March 2018.
5646 DOI 10.1109/SANER.2018.8330213. ISBN 978-1-53-864969-5 pp. 244–254.
- 5647 [6] R. E. Al-Qutaish, “Quality Models in Software Engineering Literature: An Analytical and
5648 Comparative Study,” *Journal of American Science*, vol. 6, no. 3, pp. 166–175, 2010.
- 5649 [7] H. Allahyari and N. Lavesson, “User-oriented assessment of classification model under-
5650 standability,” in *Proceedings of the 11th Scandinavian Conference on Artificial Intelligence*, vol.
5651 227. Trondheim, Norway: IOS Press, May 2011. DOI 10.3233/978-1-60750-754-3-11.
5652 ISBN 978-1-60-750753-6. ISSN 0922-6389 pp. 11–19.
- 5653 [8] M. Allamanis and C. Sutton, “Why, when, and what: Analyzing stack overflow questions
5654 by topic, type, and code,” in *Proceedings of the 10th IEEE International Working Con-
5655 ference on Mining Software Repositories*. San Francisco, CA, USA: IEEE, May 2013.
5656 DOI 10.1109/MSR.2013.6624004. ISBN 978-1-46-732936-1. ISSN 2160-1852 pp. 53–56.
- 5657 [9] C. O. Alm, D. Roth, and R. Sproat, “Emotions from Text: Machine Learning for Text-Based
5658 Emotion Prediction,” in *Proceedings of the Conference on Human Language Technology and
5659 Empirical Methods in Natural Language Processing*. Vancouver, BC, Canada: Association
5660 for Computational Linguistics, October 2005. DOI 10.3115/1220575.1220648, pp. 579–586.
- 5661 [10] J. Alway and C. Calhoun, *Critical Social Theory: Culture, History, and the Challenge of
5662 Difference*. American Sociological Association, 1997, vol. 26, no. 1, DOI 10.2307/2076647.

- [11] S. Aman and S. Szpakowicz, "Identifying Expressions of Emotion in Text," in *Proceedings of the 10th International Conference on Text, Speech and Dialogue*. Pilsen, Czech Republic: Springer, September 2007. DOI 10.1007/978-3-540-74628-7_27, pp. 196–205.
- [12] S. Amershi, M. Chickering, S. M. Drucker, B. Lee, P. Simard, and J. Suh, "Modeltracker: Redesigning performance analysis tools for machine learning," in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. Seoul, Republic of Korea: ACM, April 2015. DOI 10.1145/2702123.2702509. ISBN 978-1-45-033145-6 pp. 337–346.
- [13] K. Arnold, "Programmers are People, Too," *ACM Queue*, vol. 3, no. 5, pp. 54–59, 2005, DOI 10.1145/1071713.1071731. ISSN 1542-7749
- [14] M. Arnold, D. Piorkowski, D. Reimer, J. Richards, J. Tsay, K. R. Varshney, R. K. E. Bellamy, M. Hind, S. Houde, S. Mehta, A. Mojsilovic, R. Nair, K. N. Ramamurthy, and A. Olteanu, "FactSheets: Increasing trust in AI services through supplier's declarations of conformity," *IBM Journal of Research and Development*, vol. 63, no. 4-5, pp. 6:1 – 6:13, 2019, DOI 10.1147/JRD.2019.2942288.
- [15] A. Arpteg, B. Brinne, L. Crnkovic-Friis, and J. Bosch, "Software engineering challenges of deep learning," in *Proceedings of the 44th Euromicro Conference on Software Engineering and Advanced Applications*. Prague, Czech Republic: IEEE, August 2018. DOI 10.1109/SEAA.2018.00018. ISBN 978-1-53-867382-9 pp. 50–59.
- [16] W. R. Ashby and J. R. Pierce, "An Introduction to Cybernetics," *Physics Today*, vol. 10, no. 7, pp. 34–36, July 1957.
- [17] L. Aversano, D. Guardabascio, and M. Tortorella, "Analysis of the Documentation of ERP Software Projects," *Procedia Computer Science*, vol. 121, pp. 423–430, January 2017, DOI 10.1016/j.procs.2017.11.057. ISSN 1877-0509
- [18] D. Baehrens, T. Schroeter, S. Harmeling, M. Kawanabe, K. Hansen, and K. R. Müller, "How to explain individual classification decisions," *Journal of Machine Learning Research*, vol. 11, pp. 1803–1831, 2010. ISSN 1532-4435
- [19] B. Baesens, C. Mues, M. De Backer, J. Vanthienen, and R. Setiono, "Building intelligent credit scoring systems using decision tables," in *Proceedings of the 5th International Conference on Enterprise Information Systems*, vol. 2. Angers, France: IEEE, April 2003. DOI 10.1007/1-4020-2673-0_15. ISBN 9-72-988161-8 pp. 19–25.
- [20] X. Bai, Y. Wang, G. Dai, W. T. Tsai, and Y. Chen, "A framework for contract-based collaborative verification and validation of Web services," in *Proceedings of the 10th International Symposium of Component-Based Software Engineering*. Medford, MA, USA: Springer, July 2007. DOI 10.1007/978-3-540-73551-9_18. ISBN 978-3-54-073550-2. ISSN 0302-9743 pp. 258–273.
- [21] K. Bajaj, K. Pattabiraman, and A. Mesbah, "Mining questions asked by web developers," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. Hyderabad, India: ACM, May 2014. DOI 10.1145/2597073.2597083. ISBN 978-1-45-032863-0 pp. 112–121.
- [22] K. Ballinger, "Simplicity and Utility, or, Why SOAP Lost," [Online] Available: <http://bit.ly/37vLms0>, December 2014, Accessed: 28 August 2018.
- [23] A. Bangor, P. T. Kortum, and J. T. Miller, "An empirical evaluation of the system usability scale," *International Journal of Human-Computer Interaction*, 2008, DOI 10.1080/10447310802205776. ISSN 10447318
- [24] O. Baños, M. Damas, H. Pomares, I. Rojas, M. A. Tóth, and O. Amft, "A Benchmark Dataset to Evaluate Sensor Displacement in Activity Recognition," in *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*. Pittsburgh, PA, USA: ACM, 2012. DOI 10.1145/2370216.2370437. ISBN 9781450312240 pp. 1026–1035.
- [25] S. Barnett, "Extracting technical domain knowledge to improve software architecture," Ph.D. dissertation, Swinburne University of Technology, Hawthorn, VIC, Australia, 2018.
- [26] S. Barnett, R. Vasa, and J. Grundy, "Bootstrapping Mobile App Development," in *Proceedings of the 37th International Conference on Software Engineering*. Florence, Italy: IEEE, May 2015. DOI 10.1109/ICSE.2015.216. ISBN 978-1-47-991934-5. ISSN 0270-5257 pp. 657–660.
- [27] S. Barnett, R. Vasa, and A. Tang, "A Conceptual Model for Architecting Mobile Applications," in *Proceedings of the 12th Working IEEE/IFIP Conference on Software Architecture*. Montreal,

- QC, Canada: IEEE, May 2015. DOI 10.1109/WICSA.2015.28. ISBN 978-1-47-991922-2 pp. 105–114.
- [28] A. Barua, S. W. Thomas, and A. E. Hassan, “What are developers talking about? An analysis of topics and trends in Stack Overflow,” *Empirical Software Engineering*, vol. 19, no. 3, pp. 619–654, 2014, DOI 10.1007/s10664-012-9231-y. ISSN 1573-7616
- [29] Y. Baruch, “Response rate in academic studies - A comparative analysis,” *Human Relations*, vol. 52, no. 4, pp. 421–438, 1999, DOI 10.1177/00182679905200401. ISSN 0018-7267
- [30] O. Barzilay, C. Treude, and A. Zagalsky, “Facilitating crowd sourced software engineering via stack overflow,” in *Finding Source Code on the Web for Remix and Reuse*, 2014, no. 4, pp. 289–308. ISBN 978-1-46-146596-6
- [31] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 2nd ed. Addison-Wesley, 2003. ISBN 0-32-115495-9
- [32] B. E. Bejnordi, M. Veta, P. J. Van Diest, B. Van Ginneken, N. Karssemeijer, G. Litjens, J. A. W. M. Van Der Laak, M. Hermsen, Q. F. Manson, M. Balkenhol, O. Geessink, N. Stathonikos, M. C. R. F. Van Dijk, P. Bult, F. Beca, A. H. Beck, D. Wang, A. Khosla, R. Gargya, H. Irshad, A. Zhong, Q. Dou, Q. Li, H. Chen, H. J. Lin, P. A. Heng, C. Haß, E. Bruni, Q. Wong, U. Halici, M. Ü. Öner, R. Cetin-Atalay, M. Berseth, V. Khvatkov, A. Vylegzhannin, O. Kraus, M. Shaban, N. Rajpoot, R. Awan, K. Sirinukunwattana, T. Qaiser, Y. W. Tsang, D. Tellez, J. Annuscheit, P. Hufnagl, M. Valkonen, K. Kartasalo, L. Latonen, P. Ruusuvuori, K. Liimatainen, S. Albarqouni, B. Mungal, A. George, S. Demirci, N. Navab, S. Watanabe, S. Seno, Y. Takenaka, H. Matsuda, H. A. Phoulady, V. Kovalev, A. Kalinovsky, V. Liauchuk, G. Bueno, M. M. Fernandez-Carrobles, I. Serrano, O. Deniz, D. Racoceanu, and R. Venâncio, “Diagnostic assessment of deep learning algorithms for detection of lymph node metastases in women with breast cancer,” *Journal of the American Medical Association*, vol. 318, no. 22, pp. 2199–2210, December 2017, DOI 10.1001/jama.2017.14585. ISSN 1538-3598
- [33] R. Bellazzi and B. Zupan, “Predictive data mining in clinical medicine: Current issues and guidelines,” *International Journal of Medical Informatics*, vol. 77, no. 2, pp. 81–97, 2008, DOI 10.1016/j.ijmedinf.2006.11.006. ISSN 1386-5056
- [34] A. Ben-David, “Monotonicity Maintenance in Information-Theoretic Machine Learning Algorithms,” *Machine Learning*, vol. 19, no. 1, pp. 29–43, 1995, DOI 10.1023/A:1022655006810. ISSN 1573-0565
- [35] T. Berners-Lee, R. Fielding, and L. Masinter, “Uniform resource identifier (URI): Generic syntax,” Tech. Rep., 2004.
- [36] L. L. Berry, A. Parasuraman, and V. A. Zeithaml, “SERVQUAL: A multiple-item scale for measuring consumer perceptions of service quality,” *Journal of Retailing*, vol. 64, no. 1, pp. 12–40, 1988, DOI 10.1016/S0148-2963(99)00084-3. ISBN 00224359. ISSN 0022-4359
- [37] J. Besson, “The Business Value of Quality,” [Online] Available: <https://ibm.co/2u0UDK0>, June 2004.
- [38] S. Beyer and M. Pinzger, “A manual categorization of android app development issues on stack overflow,” in *Proceedings of the 30th International Conference on Software Maintenance and Evolution*. Victoria, BC, Canada: IEEE, September 2014. DOI 10.1109/ICSME.2014.88. ISBN 978-0-76-955303-0 pp. 531–535.
- [39] S. Beyer, C. MacHo, M. Pinzger, and M. Di Penta, “Automatically classifying posts into question categories on stack overflow,” in *Proceedings of the 26th International Conference on Program Comprehension*. Gothenburg, Sweden: ACM, May 2018. DOI 10.1145/3196321.3196333. ISBN 978-1-45-035714-2. ISSN 0270-5257 pp. 211–221.
- [40] J. Biggs and K. Collis, “Evaluating the Quality of Learning: The SOLO Taxonomy (Structure of the Observed Learning Outcome),” *Management in Education*, vol. 1, no. 4, p. 20, 1987, DOI 10.1177/089202068700100412. ISBN 0-12-097551-1. ISSN 0892-0206
- [41] J. P. Bigham, R. S. Kaminsky, R. E. Ladner, O. M. Danielsson, and G. L. Hempton, “WebInSight: Making web images accessible,” in *Proceedings of the 8th International ACM SIGACCESS Conference on Computers and Accessibility*. Portland, OR, USA: ACM, October 2006. DOI 10.1145/1168987.1169018, pp. 181–188.
- [42] J. P. Bigham, C. M. Prince, and R. E. Ladner, “WebAnywhere,” in *Proceedings of the 2008 International Cross-Disciplinary Conference on Web Accessibility*. Beijing, China: ACM, April 2008. DOI 10.1145/1368044.1368060, pp. 73–82.

- 5775 [43] J. J. Blake, L. P. Maguire, T. M. McGinnity, B. Roche, and L. J. McDaid, “The implementation of
5776 fuzzy systems, neural networks and fuzzy neural networks using FPGAs,” *Information Sciences*,
5777 vol. 112, no. 1-4, pp. 151–168, 1998, DOI 10.1016/S0020-0255(98)10029-4. ISSN 0020-0255
- 5778 [44] B. S. Bloom, *Taxonomy of Educational Objectives, Handbook 1: Cognitive Domain*, 2nd ed.
5779 Addison-Wesley Longman, 1956. ISBN 978-0-58-228010-6
- 5780 [45] B. W. Boehm, J. R. Brown, and M. Lipow, “Quantitative evaluation of software quality,” in
5781 *Proceedings of the 2nd International Conference on Software Engineering*. San Francisco,
5782 California, USA: IEEE, October 1976. ISSN 0270-5257 pp. 592–605.
- 5783 [46] B. Boehm and V. R. Basili, “Software defect reduction top 10 list,” *Software Management*, pp.
5784 419–421, 2007, DOI 10.1109/9780470049167.ch12. ISBN 978-0-47-004916-7
- 5785 [47] B. W. Boehm, *Software engineering economics*. Englewood Cliffs, NJ, USA: Prentice-Hall,
5786 1981. ISBN 0-13-822122-7
- 5787 [48] S. Borsci, S. Federici, and M. Lauriola, “On the dimensionality of the System Usability Scale:
5788 A test of alternative measurement models,” *Cognitive Processing*, 2009, DOI 10.1007/s10339-
5789 009-0268-9. ISSN 16124782
- 5790 [49] C. Bottomley, “What part writer? What part programmer? A survey of practices
5791 and knowledge used in programmer writing,” in *Proceedings of the 2005 IEEE Interna-*
5792 *tional Professional Communication Conference*. Limerick, Ireland: IEEE, July 2005.
5793 DOI 10.1109/IPCC.2005.1494255, pp. 802–812.
- 5794 [50] P. Bourque and R. E. Fairley, Eds., *Guide to the Software Engineering Body of Knowledge*,
5795 3rd ed. Washington, DC, USA: IEEE, 2014. ISBN 978-0-7695-5166-1
- 5796 [51] E. Bouwers and A. van Deursen, “A Lightweight Sanity Check for Implemented Architectures,”
5797 *IEEE Software*, vol. 27, no. 4, pp. 44–50, July 2010, DOI 10.1109/MS.2010.60. ISSN 0740-
5798 7459
- 5799 [52] M. Boyd and N. Wilson, “Just ask Siri? A pilot study comparing smartphone digital assistants
5800 and laptop Google searches for smoking cessation advice,” *PLoS ONE*, vol. 13, no. 3, 2018,
5801 DOI 10.1371/journal.pone.0194811. ISSN 1932-6203
- 5802 [53] O. Boz, “Extracting decision trees from trained neural networks,” in *Proceedings of the 8th ACM*
5803 *SIGKDD International Conference on Knowledge Discovery and Data Mining*. Edmonton,
5804 AB, Canada: ACM, July 2002. DOI 10.1145/775107.775113, pp. 456–461.
- 5805 [54] H. B. Braiek and F. Khomh, “On Testing Machine Learning Programs,” *arXiv preprint*
5806 *arXiv:1812.02257*, December 2018.
- 5807 [55] M. Brammer, *Principles of Data Mining*, ser. Undergraduate Topics in Computer Science. Lon-
5808 don, England, UK: Springer, 2016, vol. 180, DOI 10.1007/978-1-4471-7307-6. ISBN 978-1-
5809 44-717306-9
- 5810 [56] J. Brandt, P. J. Guo, J. Lewenstein, M. Dontcheva, and S. R. Klemmer, “Two studies of oppor-
5811 tunistic programming: Interleaving web foraging, learning, and writing code,” in *Proceedings*
5812 *of the SIGCHI Conference on Human Factors in Computing System*. Boston, MA, USA: ACM,
5813 April 2009. DOI 10.1145/1518701.1518944. ISBN 978-1-60-558247-4 pp. 1589–1598.
- 5814 [57] L. Bratthall and M. Jørgensen, “Can you trust a single data source exploratory software engi-
5815 neering case study?” *Empirical Software Engineering*, 2002, DOI 10.1023/A:1014866909191.
5816 ISSN 1382-3256
- 5817 [58] E. Breck, S. Cai, E. Nielsen, M. Salib, and D. Sculley, “What’s your ML Test Score? A rubric for
5818 ML production systems,” in *Proceedings of the 30th Annual Conference on Neural Information*
5819 *Processing Systems*. Barcelona, Spain: Curran Associates Inc., December 2016.
- 5820 [59] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and regression trees*.
5821 New York, NY, USA: CRC press, 1984. DOI 10.1201/9781315139470. ISBN 978-1-35-
5822 146049-1
- 5823 [60] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, and M. Khalil, “Lessons from applying
5824 the systematic literature review process within the software engineering domain,” *Journal of*
5825 *Systems and Software*, vol. 80, no. 4, pp. 571–583, April 2007, DOI 10.1016/j.jss.2006.07.009.
5826 ISSN 0164-1212
- 5827 [61] J. Brooke, “SUS-A quick and dirty usability scale,” in *Usability Evaluation in Industry*. Corn-
5828 wall, England, UK: Taylor & Francis Ltd, 1996, ch. 21, pp. 189–194. ISBN 978-0-74-840460-5
- 5829 [62] ——, “SUS: a retrospective,” *Journal of Usability Studies*, vol. 8, no. 2, pp. 29–40, 2013. ISSN
5830 1931-3357

- 5831 [63] O. Bruna, H. Avetisyan, and J. Holub, "Emotion models for textual emotion classification,"
5832 *Journal of Physics: Conference Series*, vol. 772, p. 12063, November 2016, DOI 10.1088/1742-
5833 6596/772/1/012063.
- 5834 [64] M. Bunge, "A General Black Box Theory," *Philosophy of Science*, vol. 30, no. 4, pp. 346–358,
5835 October 1963, DOI 10.1086/287954. ISSN 0031-8248
- 5836 [65] BusinessWire, "FileShadow Delivers Machine Learning to End Users with Google Vision API
5837 | Business Wire," [Online] Available: <https://bwnews.pr/2O5qv78>, July 2018, Accessed: 25
5838 January 2019.
- 5839 [66] A. Bussone, S. Stumpf, and D. O'Sullivan, "The role of explanations on trust and reliance in
5840 clinical decision support systems," in *Proceedings of the 2015 IEEE International Conference on
5841 Healthcare Informatics*. Dallas, TX, USA: IEEE, October 2015. DOI 10.1109/ICHI.2015.26.
5842 ISBN 978-1-46-739548-9 pp. 160–169.
- 5843 [67] F. Calefato, F. Lanobile, and N. Novielli, "EmoTxt: a toolkit for emotion recognition from
5844 text," in *Proceedings of the 7th International Conference on Affective Computing and Intel-
5845 ligent Interaction Workshops and Demos*. San Antonio, TX, USA: IEEE, October 2017.
5846 DOI 10.1109/ACIIW.2017.8272591, pp. 79–80.
- 5847 [68] F. Calefato, F. Lanobile, F. Maiorano, and N. Novielli, "Sentiment polarity detection for
5848 software development," *Empirical Software Engineering*, vol. 23, no. 3, pp. 1352–1382, 2018,
5849 DOI 10.1007/s10664-017-9546-9.
- 5850 [69] G. Canfora, "User-side testing of Web Services," in *Proceedings of the 9th European Conference
5851 on Software Maintenance and Reengineering*. Manchester, England, UK: IEEE, March 2005.
5852 DOI 10.1109/csmr.2005.57. ISSN 1534-5351 p. 301.
- 5853 [70] G. Canfora and M. Di Penta, "Testing services and service-centric systems: Challenges and
5854 opportunities," *IT Professional*, vol. 8, no. 2, pp. 10–17, 2006, DOI 10.1109/MITP.2006.51.
5855 ISSN 1520-9202
- 5856 [71] R. Caruana, Y. Lou, J. Gehrke, P. Koch, M. Sturm, and N. Elhadad, "Intelligible models for
5857 healthcare: Predicting pneumonia risk and hospital 30-day readmission," in *Proceedings of
5858 the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*,
5859 vol. 2015-Augus. Sydney, Australia: ACM, August 2015. DOI 10.1145/2783258.2788613.
5860 ISBN 978-1-45-033664-2 pp. 1721–1730.
- 5861 [72] F. Casati, H. Kuno, G. Alonso, and V. Machiraju, *Web Services-Concepts, Architectures and
5862 Applications*, 2004. ISBN 978-3-64-207888-0
- 5863 [73] J. P. Cavano and J. A. McCall, "A framework for the measurement of software quality," in
5864 *Proceedings of the Software Quality Assurance Workshop on Functional and Performance
5865 Issues*, vol. 3, no. 5, November 1978, DOI 10.1145/800283.811113, pp. 133–139.
- 5866 [74] C. V. Chambers, D. J. Balaban, B. L. Carlson, and D. M. Grasberger, "The effect of
5867 microcomputer-generated reminders on influenza vaccination rates in a university-based family
5868 practice center." *The Journal of the American Board of Family Practice / American Board of
5869 Family Practice*, vol. 4, no. 1, pp. 19–26, 1991, DOI 10.3122/jabfm.4.1.19. ISSN 0893-8652
- 5870 [75] J. Cheng and R. Greiner, "Learning bayesian belief network classifiers: Algorithms and system,"
5871 in *Proceedings of the 14th Biennial Conference of the Canadian Society for Computational
5872 Studies of Intelligence*, vol. 2056. Ottawa, ON, Canada: Springer, June 2001. DOI 10.1007/3-
5873 540-45153-6_14. ISBN 3-54-042144-0. ISSN 1611-3349 pp. 141–151.
- 5874 [76] R. Chillarege, I. S. Bhandari, J. K. Chaar, M. J. Halliday, B. K. Ray, and D. S. Moebus, "Or-
5875 thogonal Defect Classification—A Concept for In-Process Measurements," *IEEE Transactions
5876 on Software Engineering*, vol. 18, no. 11, pp. 943–956, 1992, DOI 10.1109/32.177364. ISSN
5877 0098-5589
- 5878 [77] E. Choi, N. Yoshida, R. G. Kula, and K. Inoue, "What do practitioners ask about code clone? a
5879 preliminary investigation of stack overflow," in *Proceedings of the 9th International Workshop
5880 on Software Clones*, Montreal, QC, Canada, March 2015, DOI 10.1109/IWSC.2015.7069890.
5881 ISBN 978-1-46-736914-5 pp. 49–50.
- 5882 [78] D. V. Cicchetti, "Guidelines, Criteria, and Rules of Thumb for Evaluating Normed and Stan-
5883 dardized Assessment Instruments in Psychology," *Psychological Assessment*, vol. 6, no. 4, pp.
5884 284–290, 1994, DOI 10.1037/1040-3590.6.4.284. ISSN 10403590
- 5885 [79] Cigital, "Case Study: Finding defects earlier yields enormous savings," [Online] Available:
5886 <http://bit.ly/36Il2cE>, 2003.

- [5887] [80] P. Clark and R. Boswell, "Rule induction with CN2: Some recent improvements," in *Proceedings of the 1991 European Working Session on Learning*. Porto, Portugal: Springer, March 1991. DOI 10.1007/BFb0017011. ISBN 978-3-54-053816-5. ISSN 1611-3349 pp. 151–163.
- [5888]
- [5889]
- [5890] [81] J. Cohen, "A Coefficient of Agreement for Nominal Scales," *Educational and Psychological Measurement*, vol. 20, no. 1, pp. 37–46, 1960, DOI 10.1177/001316446002000104. ISSN 1552-3888
- [5891]
- [5892]
- [5893] [82] P. Covington, J. Adams, and E. Sargin, "Deep neural networks for youtube recommendations," in *Proceedings of the 10th ACM Conference on Recommender Systems*. Boston, MA, USA: ACM, September 2016. DOI 10.1145/2959100.2959190. ISBN 978-1-45-034035-9 pp. 191–198.
- [5894]
- [5895]
- [5896]
- [5897] [83] M. W. Craven and J. W. Shavlik, "Extracting tree-structured representations of trained neural networks," in *Proceedings of the 8th International Conference on Neural Information Processing Systems*, vol. 8. Denver, CO, USA: MIT Press, December 1996. ISBN 978-0-26-220107-0 pp. 24–30.
- [5898]
- [5899]
- [5900]
- [5901] [84] J. W. Creswell, *Research design: Qualitative, quantitative, and mixed methods approaches*, 4th ed. SAGE, 2017. ISBN 860-1-40-429618-5
- [5902]
- [5903] [85] P. B. Crosby, *Quality is free: The art of making quality certain*. McGraw-Hill, 1979. ISBN 978-0-07-014512-2
- [5904]
- [5905] [86] A. Cummaudo, R. Vasa, and J. Grundy, "What should I document? A preliminary systematic mapping study into API documentation knowledge," in *Proceedings of the 13th International Symposium on Empirical Software Engineering and Measurement*. Porto de Galinhas, Recife, Brazil: IEEE, October 2019. DOI 10.1109/ESEM.2019.8870148. ISBN 978-1-72-812968-6. ISSN 1949-3789 pp. 1–6.
- [5906]
- [5907]
- [5908]
- [5909]
- [5910] [87] A. Cummaudo, R. Vasa, J. Grundy, M. Abdelrazek, and A. Cain, "Losing Confidence in Quality: Unspoken Evolution of Computer Vision Services," in *Proceedings of the 35th IEEE International Conference on Software Maintenance and Evolution*. Cleveland, OH, USA: IEEE, December 2019. DOI 10.1109/ICSME.2019.00051. ISBN 978-1-72-813094-1 pp. 333–342.
- [5911]
- [5912]
- [5913]
- [5914]
- [5915] [88] A. Cummaudo, S. Barnett, R. Vasa, and J. Grundy, "Threshy: Supporting Safe Usage of Intelligent Web Services," 2020, Unpublished.
- [5916]
- [5917] [89] A. Cummaudo, S. Barnett, R. Vasa, J. Grundy, and M. Abdelrazek, "Beware the evolving 'intelligent' web service! An integration architecture tactic to guard AI-first components," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. Sacramento, CA, USA: ACM, November 2020, In Press.
- [5918]
- [5919]
- [5920]
- [5921]
- [5922] [90] A. Cummaudo, R. Vasa, S. Barnett, J. Grundy, and M. Abdelrazek, "Interpreting Cloud Computer Vision Pain-Points: A Mining Study of Stack Overflow," in *Proceedings of the 42nd International Conference on Software Engineering*. Seoul, Republic of Korea: ACM, July 2020, In Press.
- [5923]
- [5924]
- [5925]
- [5926] [91] A. Cummaudo, R. Vasa, and J. Grundy, "Assessing API documentation knowledge for computer vision services," 2020, Unpublished.
- [5927]
- [5928] [92] M. K. Curumsing, "Emotion-Oriented Requirements Engineering," Ph.D. dissertation, Swinburne University of Technology, Hawthorn, VIC, Australia, 2017.
- [5929]
- [5930] [93] M. K. Curumsing, A. Cummaudo, U. M. Graetsch, S. Barnett, and R. Vasa, "Ranking Computer Vision Service Issues using Emotion," *arXiv preprint arXiv:2004.03120*, 2020.
- [5931]
- [5932] [94] H. da Mota Silveira and L. C. Martini, "How the New Approaches on Cloud Computer Vision can Contribute to Growth of Assistive Technologies to Visually Impaired in the Following Years?" *Journal of Information Systems Engineering & Management*, vol. 2, no. 2, pp. 1–3, 2017, DOI 10.20897/jisem.201709. ISSN 2468-4376
- [5933]
- [5934]
- [5935]
- [5936] [95] R. M. Davison, M. G. Martinsons, and N. Kock, "Principles of canonical action research," *Information Systems Journal*, vol. 14, no. 1, pp. 65–86, 2004, DOI 10.1111/j.1365-2575.2004.00162.x. ISSN 1350-1917
- [5937]
- [5938]
- [5939] [96] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, April 2002, DOI 10.1109/4235.996017. ISSN 1089778X
- [5940]
- [5941]

- 5942 [97] K. Dejaeger, F. Goethals, A. Giangreco, L. Mola, and B. Baesens, "Gaining insight into student
5943 satisfaction using comprehensible data mining techniques," *European Journal of Operational*
5944 *Research*, vol. 218, no. 2, pp. 548–562, 2012, DOI 10.1016/j.ejor.2011.11.022. ISSN 0377-2217
- 5945 [98] I. Dey, *Qualitative Data Analysis: A User-Friendly Guide for Social Scientists*. New York,
5946 NY: Routledge, 1993. DOI 10.4324/9780203412497. ISBN 978-0-41-505852-0
- 5947 [99] V. Dhar, D. Chou, and F. Provost, "Discovering interesting patterns for investment decision
5948 making with GLOWER - A genetic learner overlaid with entropy reduction," *Data Mining and*
5949 *Knowledge Discovery*, vol. 4, no. 4, pp. 69–80, 2000, DOI 10.1023/A:1009848126475. ISSN
5950 1384-5810
- 5951 [100] V. Dibia, A. Cox, and J. Weisz, "Designing for Democratization: Introducing Novices to
5952 Artificial Intelligence Via Maker Kits," in *Proceedings of the 2017 CHI Conference Extended*
5953 *Abstracts on Human Factors in Computing Systems*. Denver, CO, USA: ACM, May 2017, pp.
5954 381–384.
- 5955 [101] M. Doderer, K. Yoon, J. Salinas, and S. Kwek, "Protein subcellular localization prediction
5956 using a hybrid of similarity search and Error-Correcting Output Code techniques that produces
5957 interpretable results," *In Silico Biology*, vol. 6, no. 5, pp. 419–433, 2006. ISSN 1386-6338
- 5958 [102] P. Domingos, "Occam's Two Razors: The Sharp and the Blunt," in *Proceedings of the 4th*
5959 *International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA:
5960 AAAI, August 1998. DOI 10.1.1.40.3278, pp. 37–43.
- 5961 [103] B. Dorn and M. Guzdial, "Learning on the job: Characterizing the programming knowl-
5962 edge and learning strategies of web designers," in *Proceedings of the 28th ACM Conference*
5963 *on Human Factors in Computing Systems*, vol. 2. Atlanta, GA, USA: ACM, April 2010.
5964 DOI 10.1145/1753326.1753430. ISBN 978-1-60-558929-9 pp. 703–712.
- 5965 [104] F. Doshi-Velez and B. Kim, "Towards A Rigorous Science of Interpretable Machine Learning,"
5966 *arXiv preprint arXiv:1702.08608*, 2017.
- 5967 [105] F. Doshi-Velez, M. Kortz, R. Budish, C. Bavitz, S. J. Gershman, D. O'Brien, S. Shieber,
5968 J. Waldo, D. Weinberger, and A. Wood, "Accountability of AI Under the Law: The Role of
5969 Explanation," *SSRN Electronic Journal*, November 2017, In Press, DOI 10.2139/ssrn.3064761.
- 5970 [106] R. G. Dromey, "A model for software product quality," *IEEE Transactions on Software Engi-
5971 neering*, vol. 21, no. 2, pp. 146–162, 1995, DOI 10.1109/32.345830. ISBN 978-1-11-815666-7.
5972 ISSN 0098-5589
- 5973 [107] C. Drummond and R. C. Holte, "Cost curves: An improved method for visualizing classifier per-
5974 formance," *Machine Learning*, vol. 65, no. 1, pp. 95–130, October 2006, DOI 10.1007/s10994-
5975 006-8199-5. ISSN 0885-6125
- 5976 [108] T. Durieux, Y. Hamadi, and M. Monperrus, "Fully Automated HTML and Javascript Rewrit-
5977 ing for Constructing a Self-Healing Web Proxy," in *Proceedings of the 29th International*
5978 *Symposium on Software Reliability Engineering*. Memphis, TN, USA: IEEE, October 2018.
5979 DOI 10.1109/ISSRE.2018.00012. ISSN 1071-9458 pp. 1–12.
- 5980 [109] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian, "Selecting empirical methods for
5981 software engineering research," in *Guide to Advanced Empirical Software Engineering*, F. Shull,
5982 J. Singer, and D. I. K. Sjøberg, Eds. Springer, November 2007, ch. 11, pp. 285–311. ISBN
5983 978-1-84-800043-8
- 5984 [110] P. Ekman, W. V. Friesen, and J. Hager, *Facial Action Coding System: A Technique for the*
5985 *Measurement of Facial Movement*. Palo Alto, CA, USA: Consulting Psychologists Press,
5986 1978.
- 5987 [111] W. Elazmeh, S. Matwin, D. O'Sullivan, W. Michalowski, and K. Farion, "Insights from pre-
5988 dicting pediatric asthma exacerbations from retrospective clinical data," in *Proceedings of the*
5989 *22nd Conference on Artificial Intelligence*, vol. WS-07-05. Vancouver, BC, Canada: AAAI,
5990 July 2007. ISBN 978-1-57-735332-4 pp. 10–15.
- 5991 [112] N. Elgendi and A. Elragal, "Big data analytics: A literature review paper," in *Proceedings of*
5992 *the 10th Industrial Conference on Data Mining*. St. Petersburg, Russia: Springer, July 2014.
5993 DOI 10.1007/978-3-319-08976-8_16. ISSN 1611-3349 pp. 214–227.
- 5994 [113] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno,
5995 and D. Song, "Robust Physical-World Attacks on Deep Learning Visual Classification," in
5996 *Proceedings of the 2017 IEEE Computer Society Conference on Computer Vision and Pattern*

- 997 *Recognition*, Honolulu, HI, USA, July 2018, DOI 10.1109/CVPR.2018.00175. ISBN 978-1-
5998 53-866420-9. ISSN 1063-6919 pp. 1625–1634.

5999 [114] F. Elder, D. Michie, D. J. Spiegelhalter, and C. C. Taylor, “Machine Learning, Neural, and
6000 Statistical Classification,” *Journal of the American Statistical Association*, vol. 91, no. 433, pp.
6001 436–438, 1996, DOI 10.2307/2291432. ISBN 978-0-13-106360-0. ISSN 0162-1459

6002 [115] A. J. Feelders, “Prior knowledge in economic applications of data mining,” in *Proceedings of
6003 the 4th European Conference on Principles of Data Mining and Knowledge Discovery*, vol.
6004 1910. Lyon, France: Springer, September 2000. DOI 10.1007/3-540-45372-5_42. ISBN
6005 978-3-540-041066-9. ISSN 1611-3349 pp. 395–400.

6006 [116] R. T. Fielding, “Architectural Styles and the Design of Network-based Software Architectures,”
6007 Ph.D. dissertation, University of California, Irvine, 2000.

6008 [117] I. Finalyson, “Nondeterministic Finite Automata,” [Online] Available: <http://bit.ly/319GOF9>,
6009 Fredericksburg, VA, USA, 2018.

6010 [118] J. L. Fleiss, “Measuring nominal scale agreement among many raters,” *Psychological Bulletin*,
6011 vol. 76, no. 5, pp. 378–382, 1971, DOI 10.1037/h0031619.

6012 [119] H. Foster, S. Uchitel, J. Magee, and J. Kramer, “Model-based verification of Web service
6013 compositions,” in *Proceedings of the 18th International Conference on Automated Software
6014 Engineering*. Linz, Austria: IEEE, September 2004. DOI 10.1109/ase.2003.1240303, pp.
6015 152–161.

6016 [120] A. A. Freitas, “A critical review of multi-objective optimization in data mining,” *ACM SIGKDD
6017 Explorations Newsletter*, vol. 6, no. 2, p. 77, 2004, DOI 10.1145/1046456.1046467. ISSN 1931-
6018 0145

6019 [121] ——, “Comprehensible classification models,” *ACM SIGKDD Explorations Newsletter*, vol. 15,
6020 no. 1, pp. 1–10, March 2014, DOI 10.1145/2594473.2594475. ISSN 1931-0145

6021 [122] A. A. Freitas, D. C. Wieser, and R. Apweiler, “On the importance of comprehensible classi-
6022 fication models for protein function prediction,” *IEEE/ACM Transactions on Computational
6023 Biology and Bioinformatics*, vol. 7, no. 1, pp. 172–182, 2010, DOI 10.1109/TCBB.2008.47.
6024 ISSN 1545-5963

6025 [123] B. J. Frey and D. Dueck, “Clustering by passing messages between data points,” *Science*, vol.
6026 315, no. 5814, pp. 972–976, February 2007, DOI 10.1126/science.1136800. ISSN 0036-8075

6027 [124] N. Friedman, D. Geiger, and M. Goldszmidt, “Bayesian Network Classifiers,” *Machine Learn-
6028 ing*, vol. 29, no. 2-3, pp. 131–163, 1997, DOI 10.1002/9780470400531.eorms0099. ISSN
6029 0885-6125

6030 [125] G. Fung, S. Sandilya, and R. B. Rao, “Rule extraction from linear support vector machines,”
6031 *Studies in Computational Intelligence*, vol. 80, no. 1, pp. 83–107, 2009, DOI 10.1007/978-3-
6032 540-75390-2_4.

6033 [126] D. Gachechiladze, F. Lanubile, N. Novielli, and A. Serebrenik, “Anger and its direction in
6034 collaborative software development,” in *Proceedings of the 39th International Conference on
6035 Software Engineering: New Ideas and Emerging Technologies Results Track*, IEEE. Buenos
6036 Aires, Argentina: IEEE, May 2017. DOI 10.1109/ICSE-NIER.2017.18, pp. 11–14.

6037 [127] M. Gamer, J. Lemon, I. Fellows, and P. Singh, “Irr: various coefficients of interrater reliability,”
6038 *R package version 0.83*, 2010.

6039 [128] S. K. Garg, S. Versteeg, and R. Buyya, “SMICloud: A framework for comparing and ranking
6040 cloud services,” in *Proceedings of the 4th IEEE International Conference on Utility and Cloud
6041 Computing*. Melbourne, Australia: IEEE, December 2011. DOI 10.1109/UCC.2011.36.
6042 ISBN 978-0-76-954592-9 pp. 210–218.

6043 [129] V. Garousi and M. Felderer, “Experience-based guidelines for effective and efficient data ex-
6044 traction in systematic reviews in software engineering,” in *Proceedings of the 21st International
6045 Conference on Evaluation and Assessment in Software Engineering*, vol. Part F1286. Karl-
6046 skrona, Sweden: ACM, June 2017. DOI 10.1145/3084226.3084238. ISBN 978-1-45-034804-1
6047 pp. 170–179.

6048 [130] V. Garousi, M. Felderer, and M. V. Mäntylä, “Guidelines for including grey literature and
6049 conducting multivocal literature reviews in software engineering,” *Information and Software
6050 Technology*, vol. 106, pp. 101–121, 2019, DOI 10.1016/j.infsof.2018.09.006. ISSN 0950-5849

6051 [131] D. A. Garvin, “What Does ‘Product Quality’ Really Mean?” *MIT Sloan Management Review*,
6052 vol. 26, no. 1, pp. 25–43, 1984. ISSN 0019-848X

- 6053 [132] T. Gebru, J. Morgenstern, B. Vecchione, J. W. Vaughan, H. Wallach, H. Daumeé, and K. Crawford, "Datasheets for Datasets," *arXiv preprint arXiv:1803.09010*, 2018.
- 6054
- 6055 [133] R. S. Geiger, N. Varoquaux, C. Mazel-Cabasse, and C. Holdgraf, "The Types, Roles, and Practices of Documentation in Data Analytics Open Source Software Libraries: A Collaborative Ethnography of Documentation Work," *Computer Supported Cooperative Work: CSCW: An International Journal*, vol. 27, no. 3-6, pp. 767–802, May 2018, DOI 10.1007/s10606-018-9333-1. ISSN 1573-7551
- 6056
- 6057
- 6058
- 6059
- 6060 [134] GeoSpatial World, "Mapillary and Amazon Rekognition collaborate to build a parking solution for US cities through computer vision," [Online] Available: <http://bit.ly/36AdRmS>, September 2018, Accessed: 25 January 2019.
- 6061
- 6062
- 6063 [135] M. Gethsiyal Augusta and T. Kathirvalavakumar, "Reverse engineering the neural networks for rule extraction in classification problems," *Neural Processing Letters*, vol. 35, no. 2, pp. 131–150, 2012, DOI 10.1007/s11063-011-9207-8. ISSN 1370-4621
- 6064
- 6065
- 6066 [136] D. Ghazi, D. Inkpen, and S. Szpakowicz, "Hierarchical approach to emotion recognition and classification in texts," in *Proceedings of the 23rd Canadian Conference on Artificial Intelligence*, vol. 6085 LNAI. Ottawa, ON, Canada: Springer, May 2010. DOI 10.1007/978-3-642-13059-5_7, pp. 40–50.
- 6067
- 6068
- 6069
- 6070 [137] H. L. Gilmore, "Product conformance cost," *Quality progress*, vol. 7, no. 5, pp. 16–19, 1974.
- 6071 [138] R. L. Glass, I. Vessey, and V. Ramesh, "RESRES: The story behind the paper "Research in software engineering: An analysis of the literature"," *Information and Software Technology*, vol. 51, no. 1, pp. 68–70, 2009, DOI 10.1016/j.infsof.2008.09.015. ISSN 0950-5849
- 6072
- 6073
- 6074 [139] M. W. Godfrey and D. M. German, "The past, present, and future of software evolution," in *Proceedings of the 2008 Frontiers of Software Maintenance*, Beijing, China, October 2008, DOI 10.1109/FOSM.2008.4659256. ISBN 978-1-42-442655-3 pp. 129–138.
- 6075
- 6076
- 6077 [140] M. W. Godfrey and Q. Tu, "Evolution in open source software: a case study," in *Conference on Software Maintenance*. San Jose, CA, USA: IEEE, August 2000. DOI 10.1109/icsm.2000.883030, pp. 131–142.
- 6078
- 6079
- 6080 [141] Google LLC, "Classification: Thresholding | Machine Learning Crash Course," [Online] Available: <http://bit.ly/36oMgWb>, 2019, Accessed: 5 February 2020.
- 6081
- 6082 [142] U. M. Graetsch, A. Cummaudo, R. Vasa, and M. K. Curumsing, "Lessons learnt using a pre-trained AI model," 2020, Unpublished.
- 6083
- 6084 [143] D. Graziotin, F. Fagerholm, X. Wang, and P. Abrahamsson, "What happens when software developers are (un)happy," *Journal of Systems and Software*, 2018, DOI 10.1016/j.jss.2018.02.041. ISSN 0164-1212
- 6085
- 6086
- 6087 [144] P. D. Grünwald, *The Minimum Description Length Principle*. MIT press, 2019. DOI 10.7551/mitpress/4643.001.0001.
- 6088
- 6089 [145] Y. Guo, L. Zhang, Y. Hu, X. He, and J. Gao, "MS-Celeb-1M: A Dataset and Benchmark for Large-Scale Face Recognition," in *Proceedings of the 16th European Conference on Computer Vision*. Amsterdam, The Netherlands: Springer, 2016. DOI 10.1007/978-3-319-46487-9_6, pp. 87–102.
- 6090
- 6091
- 6092
- 6093 [146] M. J. Hadley and H. Marc, "Web Application Description Language," [Online] Available: <http://bit.ly/2RXRhQ1>, August 2009.
- 6094
- 6095 [147] H. A. Haenssle, C. Fink, R. Schneiderbauer, F. Toberer, T. Buhl, A. Blum, A. Kalloo, A. Ben Hadj Hassen, L. Thomas, A. Enk, L. Uhlmann, C. Alt, M. Arenbergerova, R. Bakos, A. Baltzer, I. Bertlich, A. Blum, T. Bokor-Billmann, J. Bowling, N. Braghierioli, R. Braun, K. Budern-Bakhaya, T. Buhl, H. Cabo, L. Cabrijan, N. Cevic, A. Classen, D. Deltgen, C. Fink, I. Georgieva, L. E. Hakim-Meibodi, S. Hanner, F. Hartmann, J. Hartmann, G. Haus, E. Hoxha, R. Karls, H. Koga, J. Kreusch, A. Lallas, P. Majenka, A. Marghoob, C. Massone, L. Mekokishvili, D. Mestel, V. Meyer, A. Neuberger, K. Nielsen, M. Oliviero, R. Pampena, J. Paoli, E. Pawlik, B. Rao, A. Rendon, T. Russo, A. Sadek, K. Samhaber, R. Schneiderbauer, A. Schweizer, F. Toberer, L. Trennheuser, L. Vlahova, A. Wald, J. Winkler, P. Wo'lbing, and I. Zalaudek, "Man against Machine: Diagnostic performance of a deep learning convolutional neural network for dermoscopic melanoma recognition in comparison to 58 dermatologists," *Annals of Oncology*, vol. 29, no. 8, pp. 1836–1842, May 2018, DOI 10.1093/annonc/mdy166. ISSN 1569-8041
- 6096
- 6097
- 6098
- 6099
- 6100
- 6101
- 6102
- 6103
- 6104
- 6105
- 6106

- [148] K. A. Hallgren, "Computing Inter-Rater Reliability for Observational Data: An Overview and Tutorial," *Tutorials in Quantitative Methods for Psychology*, vol. 8, no. 1, pp. 23–34, February 2012, DOI 10.20982/tqmp.08.1.p023. ISSN 1913-4126
- [149] M. Hardt, E. Price, and N. Srebro, "Equality of opportunity in supervised learning," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*. Barcelona, Spain: Curran Associates Inc., December 2016. DOI 978-1-51-083881-9. ISSN 1049-5258 pp. 3323–3331.
- [150] M. Hasan, E. Agu, and E. Rundensteiner, "Using Hashtags as Labels for Supervised Learning of Emotions in Twitter Messages," in *Proceedings of the 2014 ACM SIGKDD Workshop on Healthcare Informatics*. New York, NY, USA: ACM, August 2014, pp. 187–193.
- [151] S. Haselböck, R. Weinreich, G. Buchgeher, and T. Kriegbaum, "Microservice Design Space Analysis and Decision Documentation: A Case Study on API Management," in *Proceedings of the 11th International Conference on Service-Oriented Computing and Applications, SOCA 2018*, Paris, France, November 2019, DOI 10.1109/SOCA.2018.00008, pp. 1–8.
- [152] T. Hastie, R. Tibshirani, and J. H. Friedman, *The Elements of Statistical Learning*, 2nd ed., ser. Data Mining, Inference, and Prediction. Springer, January 2001.
- [153] B. Hayete and J. R. Bienkowska, "Gotrees: Predicting go associations from protein domain composition using decision trees," in *Proceedings of the Pacific Symposium on Biocomputing 2005, PSB 2005*. Hawaii, USA: World Scientific Publishing Company, January 2005. DOI 10.1142/9789812702456_0013. ISBN 9-81-256046-7 pp. 127–138.
- [154] A. Head, C. Sadowski, E. Murphy-Hill, and A. Knight, "When not to comment: Questions and tradeoffs with API documentation for C++ projects," in *Proceedings of the 40th International Conference on Software Engineering*, ser. questions and tradeoffs with API documentation for C++ projects. Gothenburg, Sweden: ACM, May 2018. DOI 10.1145/3180155.3180176. ISSN 0270-5257 pp. 643–653.
- [155] R. Heckel and M. Lohmann, "Towards Contract-based Testing of Web Services," *Electronic Notes in Theoretical Computer Science*, vol. 116, pp. 145–156, January 2005, DOI 10.1016/j.entcs.2004.02.073. ISSN 1571-0661
- [156] D. Heckerman, D. M. Chickering, C. Meek, R. Rounthwaite, and C. Kadie, "Dependency networks for inference, collaborative filtering, and data visualization," *Journal of Machine Learning Research*, vol. 1, no. 1, pp. 49–75, 2001, DOI 10.1162/153244301753344614. ISSN 1532-4435
- [157] M. Henning, "API design matters," *Communications of the ACM*, vol. 52, no. 5, pp. 46–56, 2009, DOI 10.1145/1506409.1506424. ISSN 0001-0782
- [158] F. Hohman, A. Head, R. Caruana, R. DeLine, and S. M. Drucker, "Gamut: A design probe to understand how data scientists understand machine learning models," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. Glasgow, Scotland, UK: ACM, May 2019. DOI 10.1145/3290605.3300809. ISBN 978-1-45-035970-2
- [159] F. Hohman, M. Kahng, R. Pienta, and D. H. Chau, "Visual Analytics in Deep Learning: An Interrogative Survey for the Next Frontiers," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 8, pp. 2674–2693, 2019, DOI 10.1109/TVCG.2018.2843369. ISSN 1941-0506
- [160] J. W. Horch, *Practical Guide To Software Quality Management*. Artech House, 2003. ISBN 978-1-58-053604-2
- [161] H. Hosseini, B. Xiao, and R. Poovendran, "Google's cloud vision API is not robust to noise," in *Proceedings of the 16th IEEE International Conference on Machine Learning and Applications*. Cancun, Mexico: IEEE, December 2017. DOI 10.1109/ICMLA.2017.0-172. ISBN 978-1-53-861417-4 pp. 101–105.
- [162] D. Hou and L. Mo, "Content categorization of API discussions," in *Proceedings of the 29th International Conference on Software Maintenance*. Eindhoven, Netherlands: IEEE, September 2013. DOI 10.1109/ICSM.2013.17, pp. 60–69.
- [163] C. Howard, "Introducing Google AI," [Online] Available: <http://bit.ly/2uI6vAr>, May 2018, Accessed: 28 August 2018.
- [164] J. Huang and C. X. Ling, "Using AUC and accuracy in evaluating learning algorithms," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 3, pp. 299–310, 2005, DOI 10.1109/TKDE.2005.50. ISSN 1041-4347

- 6163 [165] J. Huysmans, K. Dejaeger, C. Mues, J. Vanthienen, and B. Baesens, “An empirical evaluation
6164 of the comprehensibility of decision table, tree and rule based predictive models,” *Decision
6165 Support Systems*, vol. 51, no. 1, pp. 141–154, April 2011, DOI 10.1016/j.dss.2010.12.003.
6166 ISSN 0167-9236
- 6167 [166] IEEE, “IEEE Standard Glossary of Software Engineering Terminology,” 1990.
- 6168 [167] J. Ingino, *Software Architect’s Handbook: Become a Successful Software Architect by Imple-
6169 menting Effective Architecture Concepts*. Birmingham, England, UK: Packt Publishing, Ltd.,
6170 2018. ISBN 978-1-78862-406-0
- 6171 [168] International Organization for Standardization, “ISO/IEC 25010:2011 Systems and software
6172 engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System
6173 and software quality models,” [Online] Available: <http://bit.ly/2S4yzGs>, 2011.
- 6174 [169] ———, “ISO 8402:1986 Information Technology - Software Product Evaluation - Quality Char-
6175 acteristics and Guidelines for Their Use,” [Online] Available: <http://bit.ly/37SK4HP>, 1986.
- 6176 [170] ———, “ISO 9000:2015 Quality management systems – Fundamentals and vocabulary,” [Online]
6177 Available: <http://bit.ly/37O4oKo>, 2015.
- 6178 [171] ———, “ISO/IEC 9126 Information Technology - Software Product Evaluation - Quality Char-
6179 acteristics and Guidelines for Their Use,” [Online] Available: <http://bit.ly/2tgMHUE>, November
6180 1999.
- 6181 [172] S. Inzunza, R. Juárez-Ramírez, and S. Jiménez, “API Documentation,” in *Proceedings of the
6182 6th World Conference on Information Systems and Technologies*. Naples, Italy: Springer,
6183 March 2018. DOI 10.1007/978-3-319-77712-2_22, pp. 229–239.
- 6184 [173] A. Iyengar, “Supporting Data Analytics Applications Which Utilize Cognitive Services,” in
6185 *Proceedings of the 37th International Conference on Distributed Computing Systems*. Atlanta,
6186 GA, USA: IEEE, June 2017. DOI 10.1109/ICDCS.2017.172. ISBN 978-1-53-861791-5 pp.
6187 1856–1864.
- 6188 [174] N. Japkowicz and M. Shah, *Evaluating learning algorithms: A classification perspective*.
6189 Cambridge University Press, 2011, vol. 9780521196, DOI 10.1017/CBO9780511921803. ISBN
6190 978-0-51-192180-3
- 6191 [175] M. W. M. Jaspers, M. Smeulers, H. Vermeulen, and L. W. Peute, “Effects of clinical decision-
6192 support systems on practitioner performance and patient outcomes: A synthesis of high-quality
6193 systematic review findings,” *Journal of the American Medical Informatics Association*, vol. 18,
6194 no. 3, pp. 327–334, 2011, DOI 10.1136/amiainjnl-2011-000094. ISSN 1067-5027
- 6195 [176] S. Y. Jeong, Y. Xie, J. Beaton, B. A. Myers, J. Stylos, R. Ehret, J. Karstens, A. Efeoglu, and
6196 D. K. Busse, “Improving documentation for eSOA APIs through user studies,” in *Proceedings
6197 of the First International Symposium on End User Development*, vol. 5435 LNCS. Siegen,
6198 Germany: Springer, March 2009. DOI 10.1007/978-3-642-00427-8_6. ISSN 0302-9743 pp.
6199 86–105.
- 6200 [177] T. Jiang and A. E. Keating, “AVID: An integrative framework for discovering functional rela-
6201 tionship among proteins,” *BMC Bioinformatics*, vol. 6, no. 1, p. 136, 2005, DOI 10.1186/1471-
6202 2105-6-136. ISSN 1471-2105
- 6203 [178] B. Jimerson and B. Gregory, “Pivotal Cloud Foundry, Google ML, and Spring,” [Online]
6204 Available: <http://bit.ly/2RUBIIL>, San Francisco, CA, USA, December 2017.
- 6205 [179] Y. Jin, *Multi-Objective Machine Learning*, ser. Studies in Computational Intelligence. Berlin,
6206 Heidelberg: Springer, 2006. DOI 10.1007/3-540-33019-4. ISBN 978-3-54-030676-4
- 6207 [180] U. Johansson and L. Niklasson, “Evolving decision trees using oracle guides,” in *Proceedings
6208 of the 2009 IEEE Symposium on Computational Intelligence and Data Mining*. Nashville,
6209 TN, USA: IEEE, May 2009. DOI 10.1109/CIDM.2009.4938655. ISBN 978-1-42-442765-9
6210 pp. 238–244.
- 6211 [181] M. Jørgensen, T. Dybå, K. Liestøl, and D. I. K. Sjøberg, “Incorrect results in software engineer-
6212 ing experiments: How to improve research practices,” *Journal of Systems and Software*, vol.
6213 116, pp. 133–145, 2016, DOI 10.1016/j.jss.2015.03.065. ISSN 0164-1212
- 6214 [182] J. M. Juran, *Juran on Planning for Quality*. New York, NY, USA: The Free Press, 1988. ISBN
6215 978-0-02-916681-9
- 6216 [183] N. Juristo and O. S. Gómez, “Replication of software engineering experiments,” in *Proceedings
6217 of the LASER Summer School on Software Engineering*. Elba Island, Italy: Springer, 2011.
6218 DOI 10.1007/978-3-642-25231-0_2. ISBN 978-3-64-225230-3. ISSN 0302-9743 pp. 60–88.

- [184] N. Juristo and A. M. Moreno, *Basics of Software Engineering Experimentation*. Boston, MA, USA: Springer, March 2001. DOI 10.1007/978-1-4757-3304-4.
- [185] A. Karwath and R. D. King, "Homology induction: The use of machine learning to improve sequence similarity searches," *BMC Bioinformatics*, vol. 3, no. 1, p. 11, 2002, DOI 10.1186/1471-2105-3-11. ISSN 1471-2105.
- [186] K. A. Kaufman and R. S. Michalski, "Learning from inconsistent and noisy data: The AQ18 approach," in *Proceedings of the 11th European Conference on Principles and Practice of Knowledge Discovery in Databases*, vol. 1609. Warsaw, Poland: Springer, September 1999. DOI 10.1007/BFb0095128. ISBN 3-540-65965-X. ISSN 1611-3349 pp. 411–419.
- [187] D. Kavaler, D. Posnett, C. Gibler, H. Chen, P. Devanbu, and V. Filkov, "Using and asking: APIs used in the Android market and asked about in StackOverflow," in *Proceedings of the 5th International Conference on Social Infomatics*. Kyoto, Japan: Springer, November 2013. DOI 10.1007/978-3-319-03260-3_35. ISBN 978-3-319-03259-7. ISSN 0302-9743 pp. 405–418.
- [188] R. Kazman, M. Klein, and P. Clements, "ATAM: Method for architecture evaluation," Software Engineering Institute, Pittsburgh, PA, USA, Tech. Rep., 2000.
- [189] B. Kim, "Interactive and Interpretable Machine Learning Models for Human Machine Collaboration," Ph.D. dissertation, Massachusetts Institute of Technology, 2015.
- [190] B. Kim, C. Rudin, and J. Shah, "The Bayesian case model: A generative approach for case-based reasoning and prototype classification," in *Proceedings of the 28th Conference on Neural Information Processing Systems*, Montreal, QC, Canada, December 2014. ISSN 1049-5258 pp. 1952–1960.
- [191] B. Kitchenham and S. Charters, "Guidelines for performing Systematic Literature Reviews in Software Engineering," Software Engineering Group, Keele University and Department of Computer Science, University of Durham, Keele, UK, Tech. Rep., 2007.
- [192] B. A. Kitchenham and S. L. Pfleeger, "Personal opinion surveys," in *Guide to Advanced Empirical Software Engineering*, F. Shull, J. Singer, and D. I. K. Sjøberg, Eds. Springer, November 2007, ch. 3, pp. 63–92. ISBN 978-1-84-800043-8
- [193] B. A. Kitchenham, T. Dybå, and M. Jorgensen, "Evidence-Based Software Engineering," in *Proceedings of the 26th International Conference on Software Engineering*. Edinburgh, Scotland, UK: IEEE, May 2004. ISBN 978-0-76-952163-3 pp. 273–281.
- [194] H. K. Klein and M. D. Myers, "A set of principles for conducting and evaluating interpretive field studies in information systems," *MIS Quarterly: Management Information Systems*, vol. 23, no. 1, pp. 67–94, 1999, DOI 10.2307/249410. ISSN 0276-7783
- [195] A. J. Ko and Y. Riche, "The role of conceptual knowledge in API usability," in *Proceedings of the 2011 IEEE Symposium on Visual Languages and Human Centric Computing*. Pittsburgh, PA, USA: IEEE, September 2011. DOI 10.1109/VLHCC.2011.6070395. ISBN 978-1-45-771245-6 pp. 173–176.
- [196] A. J. Ko, B. A. Myers, and H. H. Aung, "Six learning barriers in end-user programming systems," in *Proceedings of the 2004 IEEE Symposium on Visual Languages and Human Centric Computing*. Rome, Italy: IEEE, September 2004. DOI 10.1109/vlhcc.2004.47. ISBN 0-78-038696-5 pp. 199–206.
- [197] I. Kononenko, "Inductive and bayesian learning in medical diagnosis," *Applied Artificial Intelligence*, vol. 7, no. 4, pp. 317–337, 1993, DOI 10.1080/08839519308949993. ISSN 1087-6545
- [198] J. Kotula, "Using patterns to create component documentation," *IEEE Software*, vol. 15, no. 2, pp. 84–92, 1998, DOI 10.1109/52.663791. ISSN 0740-7459
- [199] S. Krig, *Ground Truth Data, Content, Metrics, and Analysis*. Cham: Springer International Publishing, 2016, pp. 247–271. ISBN 978-3-319-33762-3
- [200] K. Krippendorff, *Content Analysis*, ser. An Introduction to Its Methodology. SAGE, 1980. ISBN 978-1-50-639566-1
- [201] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017, DOI 10.1145/3065386. ISSN 1557-7317
- [202] J. A. Krosnick, "Survey Research," *Annual Review of Psychology*, vol. 50, no. 1, pp. 537–567, February 1999, DOI 10.1146/annurev.psych.50.1.537. ISSN 0066-4308

- [203] A. Kurakin, I. J. Goodfellow, and S. Bengio, “Adversarial examples in the physical world,” in *Proceedings of the 5th International Conference on Learning Representations*, Toulon, France, April 2017.
- [204] G. Laforge, “Machine Intelligence at Google Scale,” in *QCon*, London, England, UK, June 2018.
- [205] H. Lakkaraju, S. H. Bach, and J. Leskovec, “Interpretable decision sets: A joint framework for description and prediction,” in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. San Francisco, CA, USA: ACM, August 2016. DOI 10.1145/2939672.2939874. ISBN 978-1-45-034232-2 pp. 1675–1684.
- [206] J. R. Landis and G. G. Koch, “The Measurement of Observer Agreement for Categorical Data,” *Biometrics*, vol. 33, no. 1, p. 159, March 1977, DOI 10.2307/2529310. ISSN 0006-341X
- [207] N. Lavrač, “Selected techniques for data mining in medicine,” *Artificial Intelligence in Medicine*, vol. 16, no. 1, pp. 3–23, 1999, DOI 10.1016/S0933-3657(98)00062-1. ISSN 0933-3657
- [208] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998, DOI 10.1109/5.726791. ISSN 0018-9219
- [209] T. Lei, R. Barzilay, and T. Jaakkola, “Rationalizing neural predictions,” in *Proceedings of the 9th International Joint Conference on Natural Language Processing and Conference on Empirical Methods in Natural Language Processing*. Austin, TX, USA: Association for Computational Linguistics, November 2016. DOI 10.18653/v1/d16-1011. ISBN 978-1-94-562625-8 pp. 107–117.
- [210] T. C. Lethbridge, S. E. Sim, and J. Singer, “Studying software engineers: Data collection techniques for software field studies,” *Empirical Software Engineering*, vol. 10, no. 3, pp. 311–341, July 2005, DOI 10.1007/s10664-005-1290-x. ISSN 1382-3256
- [211] R. J. Light, “Measures of response agreement for qualitative data: Some generalizations and alternatives,” *Psychological Bulletin*, vol. 76, no. 5, pp. 365–377, 1971, DOI 10.1037/h0031643. ISSN 0033-2909
- [212] E. Lima, C. Mues, and B. Baesens, “Domain knowledge integration in data mining using decision tables: Case studies in churn prediction,” *Journal of the Operational Research Society*, vol. 60, no. 8, pp. 1096–1106, 2009, DOI 10.1057/jors.2008.161. ISSN 0160-5682
- [213] B. Lin, F. Zampetti, G. Bavota, M. Di Penta, M. Lanza, and R. Oliveto, “Sentiment analysis for software engineering: How far can we go?” in *Proceedings of the 40th International Conference on Software Engineering*. Gothenburg, Sweden: ACM, May 2018. DOI 10.1145/3180155.3180195, pp. 94–104.
- [214] T. Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, “Microsoft COCO: Common objects in context,” in *Proceedings of the 13th European Conference on Computer Vision*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds., vol. 8693 LNCS, no. PART 5. Zurich, Germany: Springer, September 2014. DOI 10.1007/978-3-319-10602-1_48. ISSN 1611-3349 pp. 740–755.
- [215] M. Linares-Vásquez, G. Bavota, M. Di Penta, R. Oliveto, and D. Poshyvanyk, “How do API changes trigger stack overflow discussions? A study on the android SDK,” in *Proceedings of the 22nd International Conference on Program Comprehension*. Hyderabad, India: ACM, June 2014. DOI 10.1145/2597008.2597155. ISBN 978-1-45-032879-1 pp. 83–94.
- [216] Z. C. Lipton, “The mythos of model interpretability,” *Communications of the ACM*, vol. 61, no. 10, pp. 35–43, 2018, DOI 10.1145/3233231. ISSN 1557-7317
- [217] M. Litwin, *How to Measure Survey Reliability and Validity*. Thousand Oaks, CA, USA: SAGE, 1995, vol. 7, DOI 10.4135/9781483348957. ISBN 978-0-80-395704-6
- [218] Y. Liu, T. Kohlberger, M. Norouzi, G. E. Dahl, J. L. Smith, A. Mohtashamian, N. Olson, L. H. Peng, J. D. Hipp, and M. C. Stumpe, “Artificial Intelligence-Based Breast Cancer Nodal Metastasis Detection.” *Archives of Pathology & Laboratory Medicine*, vol. 143, no. 7, pp. 859–868, July 2017, DOI 10.5858/arpa.2018-0147-OA. ISSN 1543-2165
- [219] D. Lo Giudice, C. Mines, A. LeClair, R. Curran, and A. Homan, “How AI Will Change Software Development And Applications,” [Online] Available: <http://bit.ly/38RiAlN>, Forrester Research, Inc., Tech. Rep., November 2016.
- [220] A. A. Lopez-Lorca, T. Miller, S. Pedell, A. Mendoza, A. Keirnan, and L. Sterling, “One size doesn’t fit all: diversifying the user using personas and emotional scenarios,” in *Proceedings of*

- 6330 *the 6th International Workshop on Social Software Engineering.* Hong Kong, China: ACM,
6331 November 2014. DOI 10.1145/2661685.2661691, pp. 25–32.
- 6332 [221] R. Lori and M. Oded, *Data mining with decision trees.* World Scientific Publishing Company,
6333 2008, vol. 69. ISBN 978-9-81-277171-1
- 6334 [222] R. E. Lyons and W. Vanderkulk, “The Use of Triple-Modular Redundancy to Improve Computer
6335 Reliability,” *IBM Journal of Research and Development*, vol. 6, no. 2, pp. 200–209, April 2010,
6336 DOI 10.1147/rd.62.0200. ISSN 0018-8646
- 6337 [223] W. Maalej and M. P. Robillard, “Patterns of knowledge in API reference documentation,” *IEEE
6338 Transactions on Software Engineering*, 2013, DOI 10.1109/TSE.2013.12. ISSN 0098-5589
- 6339 [224] G. Malgieri and G. Comandé, “Why a right to legibility of automated decision-making exists
6340 in the general data protection regulation,” *International Data Privacy Law*, vol. 7, no. 4, pp.
6341 243–265, June 2017, DOI 10.1093/idpl/ixp019. ISSN 2044-4001
- 6342 [225] L. Mandel, “Describe REST Web services with WSDL 2.0.” [Online] Available: <https://ibm.co/313RoNV>, May 2008, Accessed: 28 August 2018.
- 6344 [226] T. E. Marshall and S. L. Lambert, “Cloud-based intelligent accounting applications: Accounting
6345 task automation using IBM watson cognitive computing,” *Journal of Emerging Technologies
6346 in Accounting*, vol. 15, no. 1, pp. 199–215, 2018, DOI 10.2308/jeta-52095. ISSN 1558-7940
- 6347 [227] D. Martens, J. Vanthienen, W. Verbeke, and B. Baesens, “Performance of classification mod-
6348 els from a user perspective,” *Decision Support Systems*, vol. 51, no. 4, pp. 782–793, 2011,
6349 DOI 10.1016/j.dss.2011.01.013. ISSN 0167-9236
- 6350 [228] A. I. Martins, A. F. Rosa, A. Queirós, A. Silva, and N. P. Rocha, “European Portuguese
6351 Validation of the System Usability Scale (SUS),” in *Procedia Computer Science*, 2015,
6352 DOI 10.1016/j.procs.2015.09.273. ISSN 18770509
- 6353 [229] P. Mayring, “Mixing Qualitative and Quantitative Methods,” in *Mixed Methodology in Psycho-
6354 logical Research.* Sense Publishers, 2007, ch. 6, pp. 27–36. ISBN 978-9-07-787473-8
- 6355 [230] J. A. McCall, P. K. Richards, and G. F. Walters, “Factors in Software Quality: Concept and
6356 Definitions of Software Quality,” General Electric Company, Griffiss Air Force Base, NY, USA,
6357 Tech. Rep. RADC-TR-77-369, November 1977.
- 6358 [231] J. McCarthy, “Programs with common sense,” in *Proceedings of the Symposium on the Mech-
6359 anization of Thought Processes*, Cambridge, MA, USA, 1963, pp. 1–15.
- 6360 [232] B. McGowen, “Machine learning with Google APIs,” [Online] Available: [http://bit.ly/
6361 3aUQpo2](http://bit.ly/3aUQpo2), January 2019.
- 6362 [233] M. L. McHugh, “Interrater reliability: The kappa statistic,” *Biochimia Medica*, vol. 22, no. 3,
6363 pp. 276–282, 2012, DOI 10.11613/bm.2012.031. ISSN 1330-0962
- 6364 [234] S. G. McLellan, A. W. Roesler, J. T. Tempest, and C. I. Spinuzzi, “Building more usable APIs,”
6365 *IEEE Software*, vol. 15, no. 3, pp. 78–86, 1998, DOI 10.1109/52.676963. ISSN 0740-7459
- 6366 [235] L. McLeod and S. G. MacDonell, “Factors that affect software systems development project
6367 outcomes: A survey of research,” *ACM Computing Surveys*, vol. 43, no. 4, p. 24, 2011,
6368 DOI 10.1145/1978802.1978803. ISSN 0360-0300
- 6369 [236] J. Meltzoff and H. Cooper, *Critical thinking about research: Psychology and related fields*,
6370 2nd ed. American Psychological Association, 2018. DOI 10.1037/0000052-000.
- 6371 [237] M. Meng, S. Steinhardt, and A. Schubert, “Application programming interface documentation:
6372 What do software developers want?” *Journal of Technical Writing and Communication*, vol. 48,
6373 no. 3, pp. 295–330, August 2018, DOI 10.1177/0047281617721853. ISSN 1541-3780
- 6374 [238] T. Mens and S. Demeyer, *Software Evolution.* Berlin, Heidelberg: Springer, 2008.
6375 DOI 10.1007/978-3-540-76440-3. ISBN 978-3-54-076439-7
- 6376 [239] T. Mens, S. Demeyer, M. Wermelinger, R. Hirschfeld, S. Ducasse, and M. Jazayeri, “Chal-
6377 lenges in software evolution,” in *Proceedings of the 8th International Workshop on Principles
6378 of Software Evolution*, vol. 2005. Lisbon, Portugal: IEEE, September 2005. DOI 10.1109/I-
6379 WPSE.2005.7. ISBN 0-76-952349-8. ISSN 1550-4077 pp. 13–22.
- 6380 [240] A. C. Michalos and H. A. Simon, *The Sciences of the Artificial.* MIT press, 1970, vol. 11,
6381 no. 1, DOI 10.2307/3102825.
- 6382 [241] D. Michie, “Machine learning in the next five years,” in *Proceedings of the 3rd European
6383 Conference on European Working Session on Learning.* Glasgow, Scotland, UK: Pitman
6384 Publishing, Inc., October 1988. ISBN 978-0-27-308800-4 pp. 107–122.

- 6385 [242] G. A. Miller, "WordNet: A Lexical Database for English," *Communications of the ACM*, vol. 38,
6386 no. 11, pp. 39–41, November 1995, DOI 10.1145/219717.219748. ISSN 1557-7317
- 6387 [243] M. Mitchell, S. Wu, A. Zaldivar, P. Barnes, L. Vasserman, B. Hutchinson, E. Spitzer, I. D.
6388 Raji, and T. Gebru, "Model cards for model reporting," in *Proceedings of the 2nd Conference
6389 on Fairness, Accountability, and Transparency*. Atlanta, GA, USA: ACM, January 2019.
6390 DOI 10.1145/3287560.3287596. ISBN 978-1-45-036125-5 pp. 220–229.
- 6391 [244] D. Moody, "The physics of notations: Toward a scientific basis for constructing visual notations
6392 in software engineering," *IEEE Transactions on Software Engineering*, vol. 35, no. 6, pp.
6393 756–779, 2009, DOI 10.1109/TSE.2009.67. ISSN 0098-5589
- 6394 [245] A. Murgia, P. Tourani, B. Adams, and M. Ortú, "Do developers feel emotions? an ex-
6395 ploratory analysis of emotions in software artifacts," in *Proceedings of the 11th Work-
6396 ing Conference on Mining Software Repositories*. Hyderabad, India: ACM, May 2014.
6397 DOI 10.1145/2597073.2597086, pp. 262–271.
- 6398 [246] C. Murphy and G. Kaiser, "Improving the Dependability of Machine Learning Applications,"
6399 Department of Computer Science, Columbia University, New York, NY, USA, Tech. Rep. Ml,
6400 2008.
- 6401 [247] C. Murphy, G. Kaiser, and M. Arias, "An approach to software testing of machine learning
6402 applications," in *Proceedings of the 19th International Conference on Software Engineering and
6403 Knowledge Engineering*, Boston, MA, USA, July 2007. ISBN 978-1-62-748661-3 pp. 167–172.
- 6404 [248] B. A. Myers, S. Y. Jeong, Y. Xie, J. Beaton, J. Stylos, R. Ehret, J. Karstens, A. Efeoglu, and D. K.
6405 Busse, "Studying the Documentation of an API for Enterprise Service-Oriented Architecture,"
6406 *Journal of Organizational and End User Computing*, vol. 22, no. 1, pp. 23–51, January 2010,
6407 DOI 10.4018/joeuc.2010101903. ISSN 1546-2234
- 6408 [249] C. Myers, A. Furqan, J. Nebolsky, K. Caro, and J. Zhu, "Patterns for how users overcome
6409 obstacles in Voice User Interfaces," in *Proceedings of the 2018 CHI Conference on Human
6410 Factors in Computing Systems*, vol. 2018-April. Montreal, QC, Canada: ACM, April 2018.
6411 DOI 10.1145/3173574.3173580. ISBN 978-1-45-035620-6 p. 6.
- 6412 [250] S. Nakajima, "Model-Checking Verification for Reliable Web Service," in *Proceedings of the
6413 First International Symposium on Cyber World*. Montreal, QC, Canada: IEEE, November
6414 2002. ISBN 978-0-76-951862-6 pp. 378–385.
- 6415 [251] M. Narayanan, E. Chen, J. He, B. Kim, S. Gershman, and F. Doshi-Velez, "How do Humans
6416 Understand Explanations from Machine Learning Systems? An Evaluation of the Human-
6417 Interpretability of Explanation," *IEEE Transactions on Evolutionary Computation*, 2018, In
6418 Press.
- 6419 [252] S. Narayanan and S. A. McIlraith, "Simulation, verification and automated composition of web
6420 services," in *Proceedings of the 11th International Conference on World Wide Web*. Honolulu,
6421 HI, USA: ACM, May 2002. DOI 10.1145/511446.511457. ISBN 1-58-113449-5 pp. 77–88.
- 6422 [253] B. J. Nelson, "Remote Procedure Call," Ph.D. dissertation, Carnegie Mellon University, 1981.
- 6423 [254] H. F. Niemeyer and A. C. Niemeyer, "Apportionment methods," *Mathematical Social Sciences*,
6424 vol. 56, no. 2, pp. 240–253, 2008. ISSN 0165-4896
- 6425 [255] Y. Nishi, S. Masuda, H. Ogawa, and K. Uetsuki, "A test architecture for machine learn-
6426 ing product," in *Proceedings of the 11th International Conference on Software Test-
6427 ing, Verification and Validation Workshops*. Västerås, Sweden: IEEE, April 2018.
6428 DOI 10.1109/ICSTW.2018.00060. ISBN 978-1-53-866352-3 pp. 273–278.
- 6429 [256] N. Novielli, F. Calefato, and F. Lanobile, "The challenges of sentiment detection in the social
6430 programmer ecosystem," in *Proceedings of the 7th International Workshop on Social Software
6431 Engineering*. Bergamo, Italy: ACM, August 2015. DOI 10.1145/2804381.2804387. ISBN
6432 978-1-45-033818-9 pp. 33–40.
- 6433 [257] ———, "A gold standard for emotion annotation in stack overflow," in *Proceedings of the 15th
6434 International Conference on Mining Software Repositories*. Gothenburg, Sweden: ACM, May
6435 2018. DOI 10.1145/3196398.3196453. ISBN 9781450357166 pp. 14–17.
- 6436 [258] K. Nybom, A. Ashraf, and I. Porres, "A systematic mapping study on API documen-
6437 tation generation approaches," in *Proceedings of the 44th Euromicro Conference on Software
6438 Engineering and Advanced Applications*. Prague, Czech Republic: IEEE, August 2018.
6439 DOI 10.1109/SEAA.2018.00081. ISBN 978-1-53-867382-9 pp. 462–469.

- [259] J. Nykaza, R. Messinger, F. Boehme, C. L. Norman, M. Mace, and M. Gordon, "What programmers really want: Results of a needs assessment for SDK documentation," in *Proceedings of the 20th Annual International Conference on Computer Documentation*. Toronto, ON, Canada: ACM, October 2002. DOI 10.1145/584955.584976, pp. 133–141.
- [260] T. Ohtake, A. Cummaudo, M. Abdelrazek, R. Vasa, and J. Grundy, "Merging intelligent API responses using a proportional representation approach," in *Proceedings of the 19th International Conference on Web Engineering*. Daejeon, Republic of Korea: Springer, June 2019. DOI 10.1007/978-3-030-19274-7_28. ISBN 978-3-03-019273-0. ISSN 1611-3349 pp. 391–406.
- [261] Open Software Foundation, "Part 3: DCE Remote Procedure Call (RPC)," in *OSF DCE application development guide: revision 1.0*. Prentice Hall, December 1991.
- [262] N. Oreskes, K. Shrader-Frechette, and K. Belitz, "Verification, validation, and confirmation of numerical models in the earth sciences," *Science*, vol. 263, no. 5147, pp. 641–646, 1994, DOI 10.1126/science.263.5147.641. ISSN 0036-8075
- [263] A. L. M. Ortiz, "Curating Content with Google Machine Learning Application Programming Interfaces," in *EIAPortugal*, July 2017.
- [264] M. Ortú, A. Murgia, G. Destefanis, P. Tourani, R. Tonelli, M. Marchesi, and B. Adams, "The emotional side of software developers in JIRA," in *Proceedings of the 13th International Conference on Mining Software Repositories*, ACM. Austin, TX, USA: ACM, May 2016. DOI 10.1145/2901739.2903505, pp. 480–483.
- [265] F. E. B. Otero and A. A. Freitas, "Improving the interpretability of classification rules discovered by an ant colony algorithm: Extended results," in *Evolutionary Computation*, vol. 24, no. 3. ACM, 2016. DOI 10.1162/EVCO_a_00155. ISSN 1530-9304 pp. 385–409.
- [266] A. Pal, S. Chang, and J. A. Konstan, "Evolution of experts in question answering communities," in *Proceedings of the 6th International AAAI Conference on Weblogs and Social Media*. Dublin, Ireland: AAAI, June 2012. ISBN 978-1-57-735556-4 pp. 274–281.
- [267] R. Parekh, "Designing AI at Scale to Power Everyday Life," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Halifax, NS, Canada: ACM, August 2017. DOI 10.1145/3097983.3105815, p. 27.
- [268] D. L. Parnas and S. A. Vilkomir, "Precise documentation of critical software," in *Proceedings of 10th IEEE International Symposium on High Assurance Systems Engineering*. Plano, TX, USA: IEEE, November 2007. DOI 10.1109/HASE.2007.63. ISSN 1530-2059 pp. 237–244.
- [269] W. G. Parrott, Ed., *Emotions in Social Psychology: Essential Readings*. Philadelphia: Psychology Press, 2001. ISBN 978-0-86-377682-3
- [270] K. Patel, J. Fogarty, J. A. Landay, and B. Harrison, "Investigating statistical machine learning as a tool for software development," in *Proceedings of the 26th SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '08. Florence, Italy: ACM, April 2008. DOI 10.1145/1357054.1357160. ISBN 978-1-60-558011-1 pp. 667–676.
- [271] C. Pautasso, O. Zimmermann, and F. Leymann, "RESTful web services vs. "Big" web services: Making the right architectural decision," in *Proceedings of the 17th International Conference on World Wide Web*. Beijing, China: ACM, April 2008. DOI 10.1145/1367497.1367606. ISBN 978-1-60-558085-2
- [272] M. Pazzani, "Comprehensible knowledge discovery: gaining insight from data," in *Proceedings of the First Federal Data Mining Conference and Exposition*, Washington, DC, USA, 1997, pp. 73–82.
- [273] M. J. Pazzani, S. Mani, and W. R. Shankle, "Acceptance of rules generated by machine learning among medical experts," *Methods of Information in Medicine*, vol. 40, no. 5, pp. 380–385, 2001, DOI 10.1055/s-0038-1634196. ISSN 0026-1270
- [274] J. Pearl, "The seven tools of causal inference, with reflections on machine learning," *Communications of the ACM*, vol. 62, no. 3, pp. 54–60, 2019, DOI 10.1145/3241036. ISSN 1557-7317
- [275] K. Petersen and C. Gencel, "Worldviews, research methods, and their relationship to validity in empirical software engineering research," in *Proceedings of the Joint Conference of the 23rd International Workshop on Software Measurement and the 8th International Conference on Software Process and Product Measurement*. Ankara, Turkey: IEEE, October 2013. DOI 10.1109/IWSM-Mensura.2013.22. ISBN 978-0-76-955078-7 pp. 81–89.

- 6495 [276] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, “Systematic mapping studies in software engineering,” in *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering, EASE 2008*, 2008, DOI 10.14236/ewic/ease2008.8, pp. 68–77.
- 6496
- 6497
- 6498 [277] Z. Pezzementi, T. Tabor, S. Yim, J. K. Chang, B. Drozd, D. Guttendorf, M. Wagner, and P. Koopman, “Putting Image Manipulations in Context: Robustness Testing for Safe Perception,” in *Proceedings of the 15th IEEE International Symposium on Safety, Security, and Rescue Robotics*. Philadelphia, PA, USA: IEEE, August 2018. DOI 10.1109/SSRR.2018.8468619. ISBN 978-1-53-865572-6 pp. 1–8.
- 6500
- 6501
- 6502
- 6503 [278] H. Pham, *System Software Reliability*, 1st ed. Springer, 2000. ISBN 978-1-84-628295-9
- 6504 [279] M. Piccioni, C. A. Furia, and B. Meyer, “An empirical study of API usability,” in *Proceedings of the 13th International Symposium on Empirical Software Engineering and Measurement*. Baltimore, MD, USA: IEEE, October 2013. DOI 10.1109/ESEM.2013.14. ISSN 1949-3770 pp. 5–14.
- 6505
- 6506
- 6507
- 6508 [280] R. M. Pirsig, *Zen and the art of motorcycle maintenance: An inquiry into values*, 1st ed. HarperTorch, 1974. ISBN 9-780-06-058946-2
- 6509
- 6510 [281] N. Polyzotis, S. Roy, S. E. Whang, and M. Zinkevich, “Data lifecycle challenges in production machine learning: A survey,” *SIGMOD Record*, 2018, DOI 10.1145/3299887.3299891. ISSN 01635808
- 6511
- 6512
- 6513 [282] R. S. Pressman, *Software Engineering: A Practitioner’s Approach*, 8th ed. McGraw-Hill, 2005. ISBN 978-0-07-802212-8
- 6514
- 6515 [283] D. Pyle, *Data Preparation for Data Mining*, 1st ed. Morgan Kaufmann, 1994. ISBN 978-15-5-860529-9
- 6516
- 6517 [284] J. R. Quinlan, “Some elements of machine learning,” in *Proceedings of the 9th International Workshop on Inductive Logic Programming*, vol. 1634. Bled, Slovenia: Springer, June 1999. DOI 10.1007/3-540-48751-4_3. ISBN 3-54-066109-3. ISSN 1611-3349 pp. 15–18.
- 6518
- 6519
- 6520 [285] ———, *C4.5: Programs for machine learning*. San Francisco, CA, USA: Morgan Kauffman, 1993. ISBN 978-1-55-860238-0
- 6521
- 6522 [286] N. Rama Suri, V. S. Srinivas, and M. Narasimha Murty, “A cooperative game theoretic approach to prototype selection,” in *Proceedings of the 11th European Conference on Principles and Practice of Knowledge Discovery in Databases*. Warsaw, Poland: Springer, September 2007. DOI 10.1007/978-3-540-74976-9_58. ISBN 978-3-54-074975-2. ISSN 0302-9743 pp. 556–564.
- 6523
- 6524
- 6525
- 6526
- 6527 [287] C. Raman Anand; Hoder, *Building Intelligent Apps with Cognitive APIs*, 1st ed. Sebastopol, CA, USA: O’Reilly Media, Inc., 2019. ISBN 978-1-49-205862-5
- 6528
- 6529 [288] M. Reboucas, G. Pinto, F. Ebert, W. Torres, A. Serebrenik, and F. Castor, “An Empirical Study on the Usage of the Swift Programming Language,” in *Proceedings of the 23rd International Conference on Software Analysis, Evolution, and Reengineering*. Suita, Japan: IEEE, March 2016. DOI 10.1109/saner.2016.66, pp. 634–638.
- 6530
- 6531
- 6532
- 6533 [289] A. Reis, D. Paulino, V. Filipe, and J. Barroso, “Using online artificial vision services to assist the blind - An assessment of Microsoft Cognitive Services and Google Cloud Vision,” *Advances in Intelligent Systems and Computing*, vol. 746, no. 12, pp. 174–184, 2018, DOI 10.1007/978-3-319-77712-2_17. ISBN 978-3-31-977711-5. ISSN 2194-5357
- 6534
- 6535
- 6536
- 6537 [290] M. T. Ribeiro, S. Singh, and C. Guestrin, “‘Why Should I Trust You?’: Explaining the Predictions of Any Classifier,” in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. San Francisco, CA, USA: ACM, August 2016. DOI 2939672.2939778, pp. 1135–1144.
- 6538
- 6539
- 6540
- 6541 [291] M. Ribeiro, K. Grolinger, and M. A. M. Capretz, “MLaaS: Machine learning as a service,” in *Proceedings of the 14th International Conference on Machine Learning and Applications*. Miami, FL, USA: IEEE, December 2015. DOI 10.1109/ICMLA.2015.152. ISBN 978-1-50-900287-0 pp. 896–902.
- 6542
- 6543
- 6544
- 6545 [292] G. Richards, V. J. Rayward-Smith, P. H. Sönksen, S. Carey, and C. Weng, “Data mining for indicators of early mortality in a database of clinical records,” *Artificial Intelligence in Medicine*, vol. 22, no. 3, pp. 215–231, 2001, DOI 10.1016/S0933-3657(00)00110-X. ISSN 0933-3657
- 6546
- 6547
- 6548 [293] G. Ridgeway, D. Madigan, T. Richardson, and J. O’Kane, “Interpretable Boosted Naïve Bayes Classification,” in *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: AAAI, 1998, pp. 101–104.
- 6549
- 6550

- [294] RightScale Inc., “State of the Cloud Report: DevOps Trends,” Tech. Rep., 2016.
- [295] G. Ritzer and E. Guba, “The Paradigm Dialog,” *Canadian Journal of Sociology*, vol. 16, no. 4, p. 446, 1991, DOI 10.2307/3340973. ISSN 0318-6431
- [296] M. P. Robillard, “What makes APIs hard to learn? Answers from developers,” *IEEE Software*, vol. 26, no. 6, pp. 27–34, 2009, DOI 10.1109/MS.2009.193. ISSN 0740-7459
- [297] M. P. Robillard and R. Deline, “A field study of API learning obstacles,” *Empirical Software Engineering*, vol. 16, no. 6, pp. 703–732, 2011, DOI 10.1007/s10664-010-9150-8. ISSN 1382-3256
- [298] M. P. Robillard, A. Marcus, C. Treude, G. Bavota, O. Chaparro, N. Ernst, M. A. Gerosall, M. Godfrey, M. Lanza, M. Linares-Vásquez, G. C. Murphy, L. Moreno, D. Shepherd, and E. Wong, “On-demand developer documentation,” in *Proceedings of the 33rd IEEE International Conference on Software Maintenance and Evolution*. Shanghai, China: IEEE, September 2017. DOI 10.1109/ICSME.2017.17, pp. 479–483.
- [299] H. Robinson, J. Segal, and H. Sharp, “Ethnographically-informed empirical studies of software practice,” *Information and Software Technology*, vol. 49, no. 6, pp. 540–551, 2007, DOI 10.1016/j.infsof.2007.02.007. ISSN 0950-5849
- [300] C. Rosen and E. Shihab, “What are mobile developers asking about? A large scale study using stack overflow,” *Empirical Software Engineering*, vol. 21, no. 3, pp. 1192–1223, 2016, DOI 10.1007/s10664-015-9379-3. ISSN 1573-7616
- [301] W. Rosenberry, D. Kenney, and G. Fisher, *Understanding DCE*. O’Reilly & Associates, Inc., 1992. ISBN 978-1-56-592005-7
- [302] A. Rosenfeld, R. Zemel, and J. K. Tsotsos, “The Elephant in the Room,” *arXiv preprint arXiv:1808.03305*, 2018.
- [303] A. S. Ross, M. C. Hughes, and F. Doshi-Velez, “Right for the right reasons: Training differentiable models by constraining their explanations,” in *Proceedings of the 26th International Joint Conferences on Artificial Intelligence*, Melbourne, Australia, August 2017, DOI 10.24963/ijcai.2017/371. ISBN 978-0-99-924110-3. ISSN 1045-0823 pp. 2662–2670.
- [304] R. J. Rubey and R. D. Hartwick, “Quantitative measurement of program quality,” in *Proceedings of the 1968 23rd ACM National Conference*. Las Vegas, NV, USA: ACM, August 1968. DOI 10.1145/800186.810631. ISBN 978-1-45-037486-6 pp. 671–677.
- [305] J. Rutten, M. Kwiatkowska, G. Norman, and D. Parker, *Mathematical techniques for analyzing concurrent and probabilistic systems*, ser. CRM Monograph Series, P. Panangaden and F. van Breugel, Eds. American Mathematical Society, 2004, vol. 23.
- [306] M. E. C. Santos, J. Polvi, T. Taketomi, G. Yamamoto, C. Sandor, and H. Kato, “Usability scale for handheld augmented reality,” in *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, VRST, 2014, DOI 10.1145/2671015.2671019. ISBN 9781450332538
- [307] J. Sauro and J. R. Lewis, “When designing usability questionnaires, does it hurt to be positive?” in *Proceedings of the 2011 SIGCHI Conference on Human Factors in Computing Systems*, Vancouver, BC, Canada, May 2011, DOI 10.1145/1978942.1979266, pp. 2215–2223.
- [308] M. Schwabacher and P. Langley, “Discovering communicable scientific knowledge from spatio-temporal data,” in *Proceedings of the 18th International Conference on Machine Learning*. Williamstown, MA, USA: Morgan Kaufmann, June 2001. ISBN 978-1-55-860778-1 pp. 489–496.
- [309] A. Schwaighofer and N. D. Lawrence, *Dataset shift in machine learning*, J. Quiñonero-Candela and M. Sugiyama, Eds. Cambridge, MA, USA: The MIT Press, 2008. ISBN 978-0-26-217005-5
- [310] T. A. Schwandt, “Qualitative data analysis: An expanded sourcebook,” *Evaluation and Program Planning*, vol. 19, no. 1, pp. 106–107, 1996, DOI 10.1016/0149-7189(96)88232-2. ISSN 0149-7189
- [311] D. Sculley, M. E. Otey, M. Pohl, B. Spitznagel, J. Hainsworth, and Y. Zhou, “Detecting adversarial advertisements in the wild,” in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM. San Diego, CA, USA: ACM, August 2011. DOI 10.1145/2020408.2020455, pp. 274–282.
- [312] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J. F. Crespo, and D. Dennison, “Hidden technical debt in machine learning systems,” in *Proceedings*

- 6606 *of the 29th Conference on Neural Information Processing Systems*, Montreal, QC, Canada,
6607 December 2015. ISBN 0262017091, 9780262017091. ISSN 1049-5258 pp. 2503–2511.
- 6608 [313] C. B. Seaman, “Qualitative methods,” in *Guide to Advanced Empirical Software Engineering*,
6609 F. Shull, J. Singer, and D. I. K. Sjøberg, Eds. Springer, November 2007, ch. 2, pp. 35–62.
6610 ISBN 978-1-84-800043-8
- 6611 [314] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, “Grad-CAM:
6612 Visual Explanations from Deep Networks via Gradient-Based Localization,” *International
6613 Journal of Computer Vision*, pp. 618–626, 2019, DOI 10.1007/s11263-019-01228-7. ISSN
6614 1573-1405
- 6615 [315] S. Sen and L. Knight, “A genetic prototype learner,” in *Proceedings of the International Joint
6616 Conference on Artificial Intelligence*. Montreal, QC, Canada: Morgan Kaufmann, August
6617 1995, pp. 725–733.
- 6618 [316] M. P. Sendak, M. Gao, N. Brajer, and S. Balu, “Presenting machine learning model information
6619 to clinical end users with model facts labels,” *npj Digital Medicine*, vol. 3, no. 1, p. 41, 2020,
6620 DOI 10.1038/s41746-020-0253-3. ISSN 2398-6352
- 6621 [317] C. E. Shannon and W. Weaver, “The mathematical theory of communication,” *The Bell
6622 System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948, DOI 10.1002/j.1538-
6623 7305.1948.tb01338.x.
- 6624 [318] P. Shaver, J. Schwartz, D. Kirson, and C. O’Connor, “Emotion knowledge: Further exploration
6625 of a prototype approach,” *Journal of Personality and Social Psychology*, vol. 52, no. 6, pp.
6626 1061–1086, 1987, DOI 10.1037/0022-3514.52.6.1061.
- 6627 [319] M. Shaw, “Writing good software engineering research papers,” in *Proceedings of the 25th
6628 International Conference on Software Engineering*. Portland, OR, USA: IEEE, May 2003.
6629 ISBN 978-0-76-951877-0 pp. 726–736.
- 6630 [320] M. Shepperd, “Replication studies considered harmful,” in *Proceedings of the 40th Interna-
6631 tional Conference on Software Engineering*. Gothenburg, Sweden: ACM, May 2018.
6632 DOI 10.1145/3183399.3183423. ISBN 978-1-45-035662-6. ISSN 0270-5257 pp. 73–76.
- 6633 [321] D. J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures*. New York,
6634 NY, USA: Chapman and Hall/CRC, 2004. DOI 10.4324/9780203489536.
- 6635 [322] L. Si and J. Callan, “A semisupervised learning method to merge search engine results,”
6636 *ACM Transactions on Information Systems*, vol. 21, no. 4, pp. 457–491, October 2003,
6637 DOI 10.1145/944012.944017. ISSN 1046-8188
- 6638 [323] J. Singer, S. E. Sim, and T. C. Lethbridge, “Software engineering data collection for field
6639 studies,” in *Guide to Advanced Empirical Software Engineering*, F. Shull, J. Singer, and D. I. K.
6640 Sjøberg, Eds. Springer, November 2007, ch. 1, pp. 9–34. ISBN 978-1-84-800043-8
- 6641 [324] S. Singh, M. T. Ribeiro, and C. Guestrin, “Programs as Black-Box Explanations,” *arXiv preprint
6642 arXiv:1611.07579*, November 2016.
- 6643 [325] V. S. Sinha, S. Mani, and M. Gupta, “Exploring activeness of users in QA forums,” in *Proceed-
6644 ings of the 10th Working Conference on Mining Software Repositories*. San Francisco, CA,
6645 USA: IEEE, May 2013. DOI 10.1109/MSR.2013.6624010. ISBN 978-1-46-732936-1. ISSN
6646 2160-1852 pp. 77–80.
- 6647 [326] M. Sokolova and G. Lapalme, “A systematic analysis of performance measures for classifi-
6648 cation tasks,” *Information Processing and Management*, vol. 45, no. 4, pp. 427–437, 2009,
6649 DOI 10.1016/j.ipm.2009.03.002. ISSN 0306-4573
- 6650 [327] I. Sommerville, *Software Engineering*, 9th ed. Boston, MA, USA: Addison-Wesley, 2011.
6651 ISBN 978-0-13-703515-1
- 6652 [328] P. Spector, *Summated Rating Scale Construction*. Newbury Park, CA, USA: SAGE, 1992.
6653 DOI 10.4135/9781412986038. ISBN 978-0-80-394341-4
- 6654 [329] R. Stevens, J. Ganz, V. Filkov, P. Devanbu, and H. Chen, “Asking for (and about) permissions
6655 used by Android apps,” in *Proceedings of the 10th Working Conference on Mining Software
6656 Repositories*. San Francisco, CA, USA: IEEE, May 2013. ISBN 978-1-46-732936-1 pp. 31–40.
- 6657 [330] M. A. Storey, L. Singer, B. Cleary, F. F. Filho, and A. Zagalsky, “The (R)evolution of social
6658 media in software engineering,” in *Future of Software Engineering Proceedings*. Hyderabad,
6659 India: ACM, May 2014. DOI 10.1145/2593882.2593887, pp. 100–116.
- 6660 [331] C. Strapparava and A. Valitutti, “WordNet-Affect: an Affective Extension of WordNet,” in
6661 *Proceedings of the 4th International Conference on Language Resources and Evaluation*.

- 6662 Lisbon, Portugal: European Language Resources Association (ELRA), May 2004, pp. 1083–
6663 1086.
- 6664 [332] J. Su, D. V. Vargas, and K. Sakurai, “One Pixel Attack for Fooling Deep Neural Net-
6665 works,” *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 5, pp. 828–841, 2019,
6666 DOI 10.1109/TEVC.2019.2890858. ISSN 1941-0026
- 6667 [333] G. H. Subramanian, J. Nosek, S. P. Raghunathan, and S. S. Kanitkar, “A comparison of
6668 the decision table and tree,” *Communications of the ACM*, vol. 35, no. 1, pp. 89–94, 1992,
6669 DOI 10.1145/129617.129621. ISSN 1557-7317
- 6670 [334] S. Subramanian, L. Inozemtseva, and R. Holmes, “Live API documentation,” in *Proceedings*
6671 *of the 36th International Conference on Software Engineering*. Hyderabad, India: ACM, May
6672 2014. DOI 10.1145/2568225.2568313. ISSN 0270-5257 pp. 643–652.
- 6673 [335] S. Sun, W. Pan, and L. L. Wang, “A Comprehensive Review of Effect Size Reporting and Inter-
6674 preting Practices in Academic Journals in Education and Psychology,” *Journal of Educational*
6675 *Psychology*, vol. 102, no. 4, pp. 989–1004, 2010, DOI 10.1037/a0019507. ISSN 0022-0663
- 6676 [336] D. Szafron, P. Lu, R. Greiner, D. S. Wishart, B. Poulin, R. Eisner, Z. Lu, J. Anvik, C. Macdonell,
6677 A. Fyshe, and D. Meeuwis, “Proteome Analyst: Custom predictions with explanations in a web-
6678 based tool for high-throughput proteome annotations,” *Nucleic Acids Research*, vol. 32, 2004,
6679 DOI 10.1093/nar/gkh485. ISSN 0305-1048
- 6680 [337] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus,
6681 “Intriguing properties of neural networks,” in *Proceedings of the 2nd International Conference*
6682 *on Learning Representations*. Banff, AB, Canada: ACM, April 2014.
- 6683 [338] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the Inception Archi-
6684 tecture for Computer Vision,” in *Proceedings of the 2016 IEEE Computer Society Conference*
6685 *on Computer Vision and Pattern Recognition*. Las Vegas, NV, USA: IEEE, June 2016.
6686 DOI 10.1109/CVPR.2016.308. ISBN 978-1-46-738850-4. ISSN 1063-6919 pp. 2818–2826.
- 6687 [339] M. B. W. Tabor, “Student Proves That S.A.T. Can Be: (D) Wrong,” [Online] Available: <https://nyti.ms/2UiKrrd>, New York, NY, USA, February 1997.
- 6688 [340] A. Tahir, A. Yamashita, S. Licorish, J. Dietrich, and S. Counsell, “Can you tell me if it
6689 smells? A study on how developers discuss code smells and anti-patterns in Stack Overflow,”
6690 in *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software*
6691 *Engineering*. Christchurch, New Zealand: ACM, June 2018. DOI 10.1145/3210459.3210466.
6692 ISBN 978-1-45-036403-4 pp. 68–78.
- 6693 [341] H. Takagi and C. Asakawa, “Transcoding proxy for nonvisual Web access,” in *Proceedings of*
6694 *the 2000 ACM Conference on Assistive Technologies*. Arlington, VA, USA: ACM, November
6695 2000. DOI 10.1145/354324.354371, pp. 164–171.
- 6696 [342] G. Tassey, *The economic impacts of inadequate infrastructure for software testing*. National
6697 Institute of Standards and Technology, September 2002. DOI 10.1080/10438590500197315.
6698 ISBN 978-0-75-672618-8
- 6699 [343] A. Taulavuori, E. Niemelä, and P. Kallio, “Component documentation - A key issue in software
6700 product lines,” *Information and Software Technology*, vol. 46, no. 8, pp. 535–546, June 2004,
6701 DOI 10.1016/j.infsof.2003.10.004. ISSN 0950-5849
- 6702 [344] R. S. Taylor, “Question-Negotiation and Information Seeking in Libraries,” *College and Re-*
6703 *search Libraries*, vol. 29, no. 3, 1968, DOI 10.5860/crl_29_03_178.
- 6704 [345] S. W. Thomas, B. Adams, A. E. Hassan, and D. Blostein, “Studying software evolu-
6705 tion using topic models,” *Science of Computer Programming*, vol. 80, pp. 457–479, 2014,
6706 DOI 10.1016/j.scico.2012.08.003. ISSN 0167-6423
- 6707 [346] S. Thrun, “Is Learning The n-th Thing Any Easier Than Learning The First?” in *Proceedings*
6708 *of the 8th International Conference on Neural Information Processing Systems*. Denver, CO,
6709 USA: MIT Press, November 1996. ISSN 1049-5258 p. 7.
- 6710 [347] C. Treude, O. Barzilay, and M. A. Storey, “How do programmers ask and answer questions
6711 on the web?” in *Proceedings of the 33rd International Conference on Software Engineering*.
6712 Honolulu, HI, USA: ACM, May 2011. DOI 10.1145/1985793.1985907. ISBN 978-1-45-
6713 030445-0. ISSN 0270-5257 pp. 804–807.
- 6714 [348] B. Turhan, M. Shepperd, and T. Menzies, “On the dataset shift problem in software en-
6715 gineering prediction models,” *Empirical Software Engineering*, vol. 17, pp. 62–74, 2012,
6716 DOI 10.1007/s10664-011-9182-8.
- 6717

- 6718 [349] G. Uddin and F. Khomh, "Automatic Mining of Opinions Expressed About APIs in
6719 Stack Overflow," *IEEE Transactions on Software Engineering*, February 2019, In Press,
6720 DOI 10.1109/TSE.2019.2900245. ISSN 1939-3520
- 6721 [350] G. Uddin and M. P. Robillard, "How API Documentation Fails," *IEEE Software*, vol. 32, no. 4,
6722 pp. 68–75, June 2015, DOI 10.1109/MS.2014.80. ISSN 0740-7459
- 6723 [351] M. Usman, R. Britto, J. Börstler, and E. Mendes, "Taxonomies in software engineering: A
6724 Systematic mapping study and a revised taxonomy development method," *Information and
6725 Software Technology*, vol. 85, pp. 43–59, May 2017, DOI 10.1016/j.infsof.2017.01.006. ISSN
6726 0950-5849
- 6727 [352] A. Van Assche and H. Blockeel, "Seeing the forest through the trees learning a comprehensible
6728 model from a first order ensemble," in *Proceedings of the 17th International Conference on
6729 Inductive Logic Programming*. Corvallis, OR, USA: Springer, June 2007. DOI 10.1007/978-
6730 3-540-78469-2_26. ISBN 3-540-078468-3. ISSN 0302-9743 pp. 269–279.
- 6731 [353] R. Vasa, "Growth and Change Dynamics in Open Source Software Systems," Ph.D. dissertation,
6732 Swinburne University of Technology, Hawthorn, VIC, Australia, 2010.
- 6733 [354] B. Venners, "Design by Contract: A Conversation with Bertrand Meyer," *Artima Developer*,
6734 2003.
- 6735 [355] W. Verbeke, D. Martens, C. Mues, and B. Baesens, "Building comprehensible customer churn
6736 prediction models with advanced rule induction techniques," *Expert Systems with Applications*,
6737 vol. 38, no. 3, pp. 2354–2364, 2011, DOI 10.1016/j.eswa.2010.08.023. ISSN 0957-4174
- 6738 [356] F. Wachter, Mitterstadt, "EU regulations on algorithmic decision-making and a "right to explanation"," in *Proceedings of the 2016 ICML Workshop on Human Interpretability in Machine
6739 Learning*, New York, NY, USA, June 2016, pp. 26–30.
- 6740 [357] B. Wang, Y. Yao, B. Viswanath, H. Zheng, and B. Y. Zhao, "With great training comes great
6741 vulnerability: Practical attacks against transfer learning," in *Proceedings of the 27th USENIX
6742 Security Symposium*. Baltimore, MD, USA: USENIX Association, July 2018. ISBN 978-1-
6743 93-913304-5 pp. 1281–1297.
- 6744 [358] K. Wang, *Cloud Computing for Machine Learning and Cognitive Applications: A Machine
6745 Learning Approach*. Cambridge, MA, USA: MIT Press, 2017. ISBN 978-0-26-203641-2
- 6746 [359] S. Wang, D. Lo, and L. Jiang, "An empirical study on developer interactions in StackOverflow,"
6747 in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. Coimbra, Portugal:
6748 ACM, March 2013. DOI 10.1145/2480362.2480557, pp. 1019–1024.
- 6749 [360] W. Wang and M. W. Godfrey, "Detecting API usage obstacles: A study of iOS and android
6750 developer questions," in *Proceedings of the 10th Working Conference on Mining Software
6751 Repositories*. San Francisco, CA, USA: IEEE, May 2013. DOI 10.1109/MSR.2013.6624006.
6752 ISBN 978-1-46-732936-1. ISSN 2160-1852 pp. 61–64.
- 6753 [361] W. Wang, H. Malik, and M. W. Godfrey, "Recommending Posts concerning API Issues in
6754 Developer Q&A Sites," in *Proceedings of the 12th Working Conference on Mining Software
6755 Repositories*. Florence, Italy: IEEE, May 2015. DOI 10.1109/MSR.2015.28. ISBN 978-0-
6756 7695-5594-2. ISSN 2160-1860 pp. 224–234.
- 6757 [362] R. Watson, "Development and application of a heuristic to assess trends in API documentation," in *Proceedings of the 30th ACM International Conference on Design of Communication*.
6758 Seattle, WA, USA: ACM, October 2012. DOI 10.1145/2379057.2379112. ISBN 978-1-45-
6759 031497-8 pp. 295–302.
- 6760 [363] R. Watson, M. Mark Stammes, J. Jeannot-Schroeder, and J. H. Spyridakis, "API documentation
6761 and software community values: A survey of open-source API documentation," in *Proceedings
6762 of the 31st ACM International Conference on Design of Communication*. Greenville, SC,
6763 USA: ACM, September 2013. DOI 10.1145/2507065.2507076, pp. 165–174.
- 6764 [364] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, and D. F. Ferguson, *Web Services Platform
6765 Architecture*. Crawfordsville, IN, USA: Prentice-Hall, 2005. ISBN 0-13-148874-0
- 6766 [365] G. M. Weiss, "Mining with rarity," *ACM SIGKDD Explorations Newsletter*, vol. 6, no. 1, pp.
6767 7–19, 2004, DOI 10.1145/1007730.1007734. ISSN 1931-0145
- 6768 [366] D. Wettschereck, D. W. Aha, and T. Mohri, "A Review and Empirical Evaluation of Feature
6769 Weighting Methods for a Class of Lazy Learning Algorithms," *Artificial Intelligence Review*,
6770 vol. 11, no. 1-5, pp. 273–314, 1997, DOI 10.1007/978-94-017-2053-3_11. ISSN 0269-2821
- 6771

- 6773 [367] J. Wexler, M. Pushkarna, T. Bolukbasi, M. Wattenberg, F. Viegas, and J. Wilson, “The What-If
6774 Tool: Interactive Probing of Machine Learning Models,” *IEEE Transactions on Visualization*
6775 and *Computer Graphics*, vol. 26, no. 1, pp. 56–65, 2019, DOI 10.1109/tvcg.2019.2934619.
6776 ISSN 1077-2626
- 6777 [368] H. Wickham, “A Layered grammar of graphics,” *Journal of Computational and Graphical*
6778 *Statistics*, vol. 19, no. 1, pp. 3–28, January 2010, DOI 10.1198/jcgs.2009.07098. ISSN 1061-
6779 8600
- 6780 [369] R. Wieringa, N. Maiden, N. Mead, and C. Rolland, “Requirements engineering paper classifi-
6781 cation and evaluation criteria: a proposal and a discussion,” *Requirements Engineering*, vol. 11,
6782 no. 1, pp. 102–107, March 2006, DOI 10.1007/s00766-005-0021-6. ISSN 0947-3602
- 6783 [370] Wikipedia Contributors, “List of datasets for machine-learning research — Wikipedia, The
6784 Free Encyclopedia,” [Online] Available: <https://bit.ly/3cZgwLb>, 2020.
- 6785 [371] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical Machine Learning*
6786 *Tools and Techniques*. Morgan Kaufmann, 2016. DOI 10.1016/c2009-0-19715-5. ISBN
6787 978-0-12-804291-5
- 6788 [372] C. Wohlin and A. Aurum, “Towards a decision-making structure for selecting a research design
6789 in empirical software engineering,” *Empirical Software Engineering*, vol. 20, no. 6, pp. 1427–
6790 1455, May 2015, DOI 10.1007/s10664-014-9319-7. ISSN 1573-7616
- 6791 [373] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation*
6792 in *Software Engineering*. Berlin, Heidelberg: Springer, 2012. DOI 10.1007/978-3-642-
6793 29044-2. ISBN 978-3-64-229044-2
- 6794 [374] M. L. Wong and K. S. Leung, *Data Mining Using Grammar Based Genetic Programming and*
6795 *Applications*. Springer, 2002. DOI 10.1007/b116131. ISBN 978-0-79-237746-7
- 6796 [375] M. R. Wrobel, “Emotions in the software development process,” in *Proceedings of 6th In-
6797 ternational Conference on Human System Interactions*. Sopot, Poland: IEEE, June 2013.
6798 DOI 10.1109/HSI.2013.6577875, pp. 518–523.
- 6799 [376] X. Yi and K. J. Kochut, “A CP-nets-based design and verification framework for web services
6800 composition,” in *Proceedings of the 2004 IEEE International Conference on Web Services*. San
6801 Diego, CA, USA: IEEE, July 2004. DOI 10.1109/icws.2004.1314810. ISBN 0-76-952167-3
6802 pp. 756–760.
- 6803 [377] R. K. Yin, *Case study research and applications: Design and methods*, 6th ed. Los Angeles,
6804 CA, USA: SAGE, 2017. ISBN 978-1-50-633616-9
- 6805 [378] J. Zahálka and F. Železný, “An experimental test of Occam’s razor in classification,” *Machine*
6806 *Learning*, vol. 82, no. 3, pp. 475–481, 2011, DOI 10.1007/s10994-010-5227-2. ISSN 0885-6125
- 6807 [379] J. Zhang and R. Kasturi, “Extraction of Text Objects in Video Documents: Recent Progress,” in
6808 *Proceedings of the 8th International Workshop on Document Analysis Systems*. Nara, Japan:
6809 IEEE, September 2008. DOI 10.1109/das.2008.49, pp. 5–17.
- 6810 [380] X. Zhang, A. S. Ross, A. Caspi, J. Fogarty, and J. O. Wobbrock, “Interaction Proxies for Runtime
6811 Repair and Enhancement of Mobile Application Accessibility,” in *Proceedings of the 2017 CHI*
6812 *Conference on Human Factors in Computing Systems*, ser. CHI ’17. Denver, CO, USA: ACM,
6813 May 2017. DOI 10.1145/3025453.3025846. ISBN 978-1-4503-4655-9 pp. 6024–6037.
- 6814 [381] B. Zupan, J. Demšar, M. W. Kattan, J. R. Beck, and I. Bratko, “Machine learning for survival
6815 analysis: a case study on recurrence of prostate cancer,” *Artificial intelligence in medicine*,
6816 vol. 20, no. 1, pp. 59–75, 2000.
- 6817 [382] M. Zur Muehlen, J. V. Nickerson, and K. D. Swenson, “Developing web services choreography
6818 standards - The case of REST vs. SOAP,” *Decision Support Systems*, vol. 40, no. 1, pp. 9–29,
6819 July 2005, DOI 10.1016/j.dss.2004.04.008. ISSN 0167-9236

6820

6821

List of Online Artefacts

6822

6823 The online artefacts listed below have been downloaded and stored on the Deakin
6824 Research Data Store (RDS) for archival purposes at the following location:

6825 RDS29448-Alex-Cummaudo-PhD/datasets/webrefs

- 6826 [383] Affectiva, Inc., “Home - Affectiva : Affectiva,” <http://bit.ly/36sgbMM>, 2018, accessed: 15
6827 October 2018.
- 6828 [384] Amazon Web Services, Inc., “Detecting Labels in an Image,” <https://amzn.to/2TBNTa>, 2018,
6829 accessed: 28 August 2018.
- 6830 [385] ——, “Detecting Objects and Scenes,” <https://amzn.to/2TDed5V>, 2018, accessed: 28 August
6831 2018.
- 6832 [386] ——, “Amazon Rekognition,” <https://amzn.to/2TyT2BL>, 2018, accessed: 13 September 2018.
- 6833 [387] ——, “Aws release notes,” <https://go.aws/2v0RYjr>, 2019, accessed: 18 March 2019.
- 6834 [388] ——, “Actions - amazon rekognition,” <https://amzn.to/392p3dH>, 2019, accessed: 18 March
6835 2019.
- 6836 [389] ——, “Amazon rekognition | aws machine learning blog,” <https://go.aws/37Q7lKc>, 2019, ac-
6837 cessed: 18 March 2019.
- 6838 [390] ——, “Amazon rekognition image,” <https://go.aws/2ubB6qc>, 2019, accessed: 18 March 2019.
- 6839 [391] ——, “Best practices for sensors, input images, and videos - amazon rekognition,” <https://amzn.to/2uZIW00>, 2019, accessed: 18 March 2019.
- 6840 [392] ——, “Exercise 1: Detect objects and scenes in an image (console) - amazon rekognition,” <https://amzn.to/36TkLnm>, 2019, accessed: 18 March 2019.
- 6841 [393] ——, “Java (sdk v1) code samples for amazon rekognition - aws code sample,” <https://amzn.to/2ugTle3>, 2019, accessed: 18 March 2019.
- 6842 [394] ——, “Limits in amazon rekognition - amazon rekognition,” <https://amzn.to/2On6n0h>, 2019,
6843 accessed: 18 March 2019.
- 6844 [395] ——, “Step 1: Set up an aws account and create an iam user - amazon rekognition,” <https://amzn.to/2tqW4kI>, 2019, accessed: 18 March 2019.
- 6845 [396] ——, “Troubleshooting amazon rekognition video - amazon rekognition,” <https://amzn.to/3b763fS>, 2019, accessed: 18 March 2019.
- 6846 [397] Beijing Geling Shentong Information Technology Co., Ltd., “DeepGlint,” <http://bit.ly/2uHHdPS>, 2018, accessed: 3 April 2019.
- 6847 [398] Beijing Kuangshi Technology Co., Ltd., “Megvii,” <http://bit.ly/2WJYFzk>, 2018, accessed: 3
6848 April 2019.

- 6855 [399] Clarifai, Inc., “Enterprise AI Powered Computer Vision Solutions | Clarifai,” <http://bit.ly/2TB3kSa>, 2018, accessed: 13 September 2018.
- 6856 [400] CloudSight, Inc., “Image Recognition API & Visual Search Results | CloudSight AI,” <http://bit.ly/2UmNPCw>, 2018, accessed: 13 September 2018.
- 6857 [401] Cognitec Systems GmbH, “The face recognition company - Cognitec,” <http://bit.ly/38VguBB>, 2018, accessed: 15 October 2018.
- 6858 [402] A. Cummaudo, <http://bit.ly/2KlyhcD>, 2019, accessed: 27 March 2019.
- 6859 [403] ——, <http://bit.ly/2G7saFJ>, 2019, accessed: 27 March 2019.
- 6860 [404] ——, <http://bit.ly/2G5ZEEe>, 2019, accessed: 27 March 2019.
- 6861 [405] ——, “ICSE 2020 Submission #564 Supplementary Materials,” <http://bit.ly/2Z8zOKW>, 2019.
- 6862 [406] ——, <http://bit.ly/2G6ZOeC>, 2019, accessed: 27 March 2019.
- 6863 [407] Deep AI, Inc., “DeepAI: The front page of A.I. | DeepAI,” <http://bit.ly/2TBNYgf>, 2018, accessed: 26 September 2018.
- 6864 [408] Google LLC, “Best practices for enterprise organizations | documentation | google cloud,” <http://bit.ly/2v0RSs5>, 2019, accessed: 18 March 2019.
- 6865 [409] ——, “Detect Labels | Google Cloud Vision API Documentation | Google Cloud,” <http://bit.ly/2TD5kcy>, 2018, accessed: 28 August 2018.
- 6866 [410] ——, “Class EntityAnnotation | Google.Cloud.Vision.V1,” <http://bit.ly/2TD5fpg>, 2018, accessed: 28 August 2018.
- 6867 [411] ——, “Vision API - Image Content Analysis | Cloud Vision API | Google Cloud,” <http://bit.ly/2TD9mBs>, 2018, accessed: 13 September 2018.
- 6868 [412] ——, “Machine learning glossary | google developers,” <http://bit.ly/3b38VdL>, 2019, accessed: 18 March 2019.
- 6869 [413] ——, “Open Images Dataset V4,” <http://bit.ly/2Ry2vvF>, 2019, accessed: 9 November 2018.
- 6870 [414] ——, “Quickstart: Using client libraries | cloud vision api documentation | google cloud,” <http://bit.ly/2RRMQHG>, 2019, accessed: 18 March 2019.
- 6871 [415] ——, “Release notes | cloud vision api documentation | google cloud,” <http://bit.ly/2UipY5J>, 2019, accessed: 18 March 2019.
- 6872 [416] ——, “Sample applications | cloud vision api documentation | google cloud,” <http://bit.ly/2SdoB5r>, 2019, accessed: 18 March 2019.
- 6873 [417] ——, “Tips & tricks | cloud functions documentation | google cloud,” <http://bit.ly/2GZNc8Z>, 2019, accessed: 18 March 2019.
- 6874 [418] ——, “Vision ai | derive image insights via ml | cloud vision api | google cloud,” <http://bit.ly/31nWoNx>, 2019, accessed: 18 March 2019.
- 6875 [419] Guangzhou Tup Network Technology, “TupuTech,” <http://bit.ly/2uF4IsN>, 2018, accessed: 3 April 2019.
- 6876 [420] Imaggia Technologies, “Imaggia - powerful image recognition APIs for automated categorization & tagging in the cloud and on-premises,” <http://bit.ly/2TxsyRe>, 2018, accessed: 13 September 2018.
- 6877 [421] International Business Machines Corporation, “Watson Visual Recognition - Overview | IBM,” <https://ibm.co/2TBNIO4>, 2018, accessed: 13 September 2018.
- 6878 [422] ——, “Watson Tone Analyzer,” <https://ibm.co/37w3y4A>, 2019, accessed: 25 January 2019.
- 6879 [423] Kairos AR, Inc., “Kairos: Serving Businesses with Face Recognition,” <http://bit.ly/30WHGNs>, 2018, accessed: 15 October 2018.
- 6880 [424] Microsoft Corporation, “azure-sdk-for-java/ImageTag.java,” <http://bit.ly/38IDPWU>, 2018, accessed: 28 August 2018.
- 6881 [425] ——, “Image Processing with the Computer Vision API | Microsoft Azure,” <http://bit.ly/2YqhkS6>, 2018, accessed: 13 September 2018.
- 6882 [426] ——, “How to call the Computer Vision API,” <http://bit.ly/2TD5oJk>, 2018, accessed: 28 August 2018.
- 6883 [427] ——, “What is Computer Vision?” <http://bit.ly/2TDgUnU>, 2018, accessed: 28 August 2018.
- 6884 [428] ——, “Call the computer vision api - azure cognitive services | microsoft docs,” <http://bit.ly/2vHSdjT>, 2019, accessed: 18 March 2019.
- 6885 [429] ——, “Content tags - computer vision - azure cognitive services | microsoft docs,” <http://bit.ly/2vESzHX>, 2019, accessed: 18 March 2019.

- 6910 [430] ——, “Github - azure-samples/cognitive-services-java-computer-vision-tutorial: This tutorial
6911 shows the features of the microsoft cognitive services computer vision rest api.” <http://bit.ly/37N1yoN>, 2019, accessed: 18 March 2019.
- 6912
- 6913 [431] ——, “Improving your classifier - custom vision service - azure cognitive services | microsoft
6914 docs,” <http://bit.ly/37SBkRQ>, 2019, accessed: 18 March 2019.
- 6915 [432] ——, “Quickstart: Computer vision client library for .net - azure cognitive services | microsoft
6916 docs,” <http://bit.ly/2vF3wJC>, 2019, accessed: 18 March 2019.
- 6917 [433] ——, “Release notes - custom vision service - azure cognitive services | microsoft docs,”
6918 <http://bit.ly/2UIPiaW>, 2019, accessed: 18 March 2019.
- 6919 [434] ——, “Sample: Explore an image processing app in c# - azure cognitive services | microsoft
6920 docs,” <http://bit.ly/2u4mPMh>, 2019, accessed: 18 March 2019.
- 6921 [435] ——, “Tutorial: Generate metadata for azure images - azure cognitive services | microsoft
6922 docs,” <http://bit.ly/2RRnARK>, 2019, accessed: 18 March 2019.
- 6923 [436] ——, “Tutorial: Use custom logo detector to recognize azure services - custom vision - azure
6924 cognitive services | microsoft docs,” <http://bit.ly/2RUGwPH>, 2019, accessed: 18 March 2019.
- 6925 [437] ——, “What is computer vision? - computer vision - azure cognitive services | microsoft docs,”
6926 <http://bit.ly/37SomDx>, 2019, accessed: 18 March 2019.
- 6927 [438] SenseTime, “SenseTime,” <http://bit.ly/2WH6RjF>, 2018, accessed: 3 April 2019.
- 6928 [439] Shanghai Yitu Technology Co., Ltd., “Yitu Technology,” <http://bit.ly/2uGvxgf>, 2018, accessed:
6929 3 April 2019.
- 6930 [440] Stack Overflow User #1008563 ‘samiles’, “AWS Rekognition PHP SDK gives invalid image
6931 encoding error,” <http://bit.ly/31Sgpec>, 2019, accessed: 22 June 2019.
- 6932 [441] Stack Overflow User #10318601 ‘reza naderii’, “google cloud vision category detecting,”
6933 <http://bit.ly/31Uf32t>, 2019, accessed: 22 June 2019.
- 6934 [442] Stack Overflow User #10729564 ‘gabgob’, “Multiple Google Vision OCR requests at once?”
6935 <http://bit.ly/31P09dU>, 2019, accessed: 22 June 2019.
- 6936 [443] Stack Overflow User #1453704 ‘deoptimancode’, “Human body part detection in Android,”
6937 <http://bit.ly/31T5pxd>, 2019, accessed: 22 June 2019.
- 6938 [444] Stack Overflow User #174602 ‘geekyaleks’, “aws Rekognition not initializing on iOS,” <http://bit.ly/31UeqG9>, 2019, accessed: 22 June 2019.
- 6939
- 6940 [445] Stack Overflow User #2251258 ‘James Dorfman’, “All GoogleVision label possibilities?”
6941 <http://bit.ly/31R4FZi>, 2019, accessed: 22 June 2019.
- 6942 [446] Stack Overflow User #2521469 ‘Hillary Sanders’, “Is there a full list of potential labels that
6943 Google’s Vision API will return?” <http://bit.ly/2KNnJSB>, 2019, accessed: 22 June 2019.
- 6944 [447] Stack Overflow User #2604150 ‘user2604150’, “Google Vision Accent Character Set NodeJs,”
6945 <http://bit.ly/31TsVdp>, 2019, accessed: 22 June 2019.
- 6946 [448] Stack Overflow User #3092947 ‘Mark Bench’, “Google Cloud Vision OCR API returning
6947 incorrect values for bounding box/vertices,” <http://bit.ly/31UeZjf>, 2019, accessed: 22 June
6948 2019.
- 6949 [449] Stack Overflow User #3565255 ‘CSquare’, “Vision API topicality and score always the same,”
6950 <http://bit.ly/2TD5As2>, 2019, accessed: 22 June 2019.
- 6951 [450] Stack Overflow User #4748115 ‘Latifa Al-jiffry’, “similar face recognition using google cloud
6952 vision API in android studio,” <http://bit.ly/31WhMZY>, 2019, accessed: 22 June 2019.
- 6953 [451] Stack Overflow User #4852910 ‘Gaurav Mathur’, “Amazon Rekognition Image caption,” <http://bit.ly/31P08qm>, 2019, accessed: 22 June 2019.
- 6954
- 6955 [452] Stack Overflow User #5294761 ‘Eury Pérez Beltré’, “Specify language for response in Google
6956 Cloud Vision API,” <http://bit.ly/31SsUGG>, 2019, accessed: 22 June 2019.
- 6957 [453] Stack Overflow User #549312 ‘GroovyDotCom’, “Image Selection for Training Visual Recog-
6958 nition,” <http://bit.ly/31W8lcw>, 2019, accessed: 22 June 2019.
- 6959 [454] Stack Overflow User #5809351 ‘J.Doe’, “How to confidently validate object detection results
6960 returned from Google Cloud Vision,” <http://bit.ly/31UcCNy>, 2019, accessed: 22 June 2019.
- 6961 [455] Stack Overflow User #5844927 ‘Amit Pawar’, “Google cloud Vision and Clarifai doesn’t Support
6962 tagging for 360 degree images and videos,” <http://bit.ly/31StuEm>, 2019, accessed: 22 June 2019.
- 6962
- 6963 [456] Stack Overflow User #5924523 ‘Akash Dathan’, “Can i give aspect ratio in Google Vision api?”
6964 <http://bit.ly/2KSJwsp>, 2019, accessed: 22 June 2019.
- 6965

- 6965 [457] Stack Overflow User #6210900 ‘Mike Grommet’, “Are the Cloud Vision API limits in documentation correct?” <http://bit.ly/31SsNLg>, 2019, accessed: 22 June 2019.
- 6966 [458] Stack Overflow User #6649145 ‘I. Sokolyk’, “How to get a position of custom object on image using vision recognition api,” <http://bit.ly/3210Q49>, 2019, accessed: 22 June 2019.
- 6967 [459] Stack Overflow User #6841211 ‘NigelJL’, “Google Cloud Vision - Numbers and Numerals OCR,” <http://bit.ly/31P07mi>, 2019, accessed: 22 June 2019.
- 6968 [460] Stack Overflow User #7064840 ‘Josh’, “Google Cloud Vision fails at batch annotate images. Getting Netty Shaded ClosedChannelException error,” <http://bit.ly/31UrBH9>, 2019, accessed: 22 June 2019.
- 6969 [461] Stack Overflow User #7187987 ‘tuanars10’, “Adding a local path to Microsoft Face API by Python,” <http://bit.ly/2KLeMt3>, 2019, accessed: 22 June 2019.
- 6970 [462] Stack Overflow User #7219743 ‘Davide Biraghi’, “Google Vision API does not recognize single digits,” <http://bit.ly/31Ws1Nj>, 2019, accessed: 22 June 2019.
- 6971 [463] Stack Overflow User #7738248 ‘lavuy’, “Meaning of score in Microsoft Cognitive Service’s Entity Linking API,” <http://bit.ly/2TD9vVw>, 2019, accessed: 22 June 2019.
- 6972 [464] Stack Overflow User #7604576 ‘Alagappan Narayanan’, “Text extraction - line-by-line,” <http://bit.ly/31Yc21s>, 2019, accessed: 22 June 2019.
- 6973 [465] Stack Overflow User #7692297 ‘lucas’, “Can Google Cloud Vision generate labels in Spanish via its API?” <http://bit.ly/31UcBsY>, 2019, accessed: 22 June 2019.
- 6974 [466] Stack Overflow User #7896427 ‘David mark’, “Google Api Vision, ““before_request”” error,” <http://bit.ly/31Z27Zt>, 2019, accessed: 22 June 2019.
- 6975 [467] Stack Overflow User #8210103 ‘Cosmin-Ioan Leferman’, “Google Vision API text detection strange behaviour - Javascript,” <http://bit.ly/31Ucyxi>, 2019, accessed: 22 June 2019.
- 6976 [468] Stack Overflow User #8411506 ‘AsSportac’, “How can we find an exhaustive list (or graph) of all logos which are effectively recognized using Google Vision logo detection feature?” <http://bit.ly/31Z27IX>, 2019, accessed: 22 June 2019.
- 6977 [469] Stack Overflow User #8594124 ‘God Himself’, “How to set up AWS mobile SDK in iOS project in Xcode,” <http://bit.ly/31St2pE>, 2019, accessed: 22 June 2019.
- 6978 [470] Stack Overflow User #9006896 ‘Dexter Intelligence’, “Getting wrong text sequence when image scanned by offline google mobile vision API,” <http://bit.ly/31Sgr5O>, 2019, accessed: 22 June 2019.
- 6979 [471] Stack Overflow User #9913535 ‘Sahil Mehra’, “Google Vision API: ModuleNotFoundError: module not found ‘google.oauth2’,” <http://bit.ly/31VIZfU>, 2019, accessed: 22 June 2019.
- 6980 [472] Symisc Systems, S.U.A.R.L, “Computer Vision & Media Processing APIs | PixLab,” <http://bit.ly/2Ulkw9K>, 2018, accessed: 13 September 2018.
- 6981 [473] Talkwalker Inc., “Image Recognition - Talkwalker,” <http://bit.ly/2TyT7W5>, 2018, accessed: 13 September 2018.
- 6982 [474] TheySay Limited, “Sentiment Analysis API | TheySay,” <http://bit.ly/37AzTHI>, 2019, accessed: 25 January 2019.

Part IV

Appendices

APPENDIX A

Additional Materials

A.1 Development, Documentation and Usage of Web APIs

The development of web APIs (commonly referred to as a *web service*) traces its roots back to the early 1990s, where the Open Software Foundation’s distributed computing environment (DCE) introduced a collection of services and tools for developing and maintaining distributed systems using a client/server architecture [301]. This framework used the synchronous communication paradigm remote procedure calls (RPCs) first introduced by Nelson [253] that allows procedures to be called in a remote address space as if it were local. Its communication paradigm, DCE/RPC [261], enables developers to write distributed software with underlying network code abstracted away. To bridge remote DCE/RPCs over components of different operating systems and languages, an interface definition language (IDL) document served as the common service contract or *service interface* for software components.

This important leap toward language-agnostic distributed programming paved way for XML-RPC, enabling RPCs over HTTP (and thus the Web) encoded using XML (instead of octet streams [261]). As new functionality was introduced, this lead to the natural development of the Simple Object Access Protocol (SOAP), the backbone messaging connector for web service applications, a realisation of the service-oriented architecture (SOA) [72] pattern. The SOA pattern prescribes that services are offered by service providers and consumed by service consumers in a platform- and language-agnostic manner and are used in large-scale enterprise systems (e.g., banking, health). Key to the SOA pattern is that a service’s quality attributes (see Section 2.1) can be specified and guaranteed using a service-level agreement (SLA) whereby the consumer and provider agree upon a set level of service, which in some cases are legally binding [31]. This agreement can be measured using quality of service (QoS) parameters met by the service provider during the transportation layer (e.g., response time, cost of leasing resources, reliability guarantees, system availability and trust/security assurance [358, 364]). These attributes are included within SOAP headers; thus, QoS aspects are independent from the transport layer and instead exist at the application layer [271]. The IDL of SOAP is Web Services Description Language (WSDL), providing a description of how the web service is invoked, what parameters to expect, and what data structures are returned.

While it is rich in metadata and verbosity, discussions on whether this was a benefit or drawback came about the mid-2000s [271, 382] whether the amount of data transfer paid off (especially for mobile clients where data usage was scarce). Developer usability for debugging the SOAP ‘envelopes’ (messages POSTed over HTTP to the service provider component) was difficult, both due to the nature of XML’s wordiness and difficulty to test (by sending POST requests) in-browser. As a simple example, 25 lines (794 bytes) of HTTP communication is transferred to request a customer’s name from a record using SOAP (Listings A.1 and A.2).

Listing A.1: A SOAP HTTP POST consumer request to retrieve customer record #43456 from a web service provider. Source: [22].

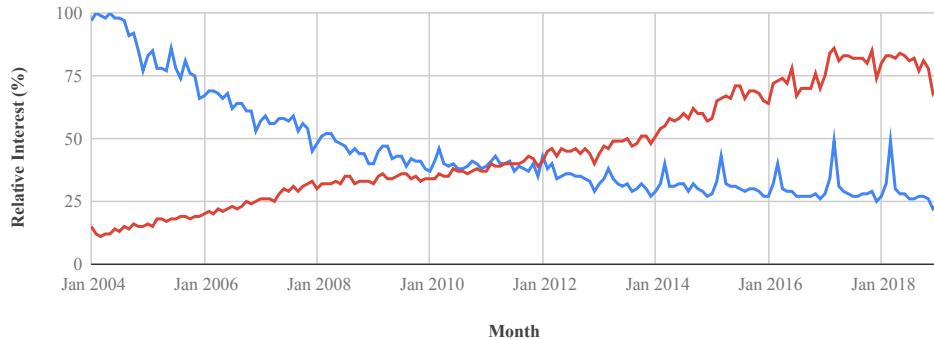


Figure A.1: Worldwide search interest for SOAP (blue) and REST (red) since 2004. Source: Google Trends.

```

1 POST /customers HTTP/1.1
2 Host: www.example.org
3 Content-Type: application/soap+xml; charset=utf-8
4
5 <?xml version="1.0"?>
6 <soap:Envelope
7   xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
8     <soap:Body>
9       <m:GetCustomer
10         xmlns:m="http://www.example.org/customers">
11           <m:CustomerId>43456</m:CustomerId>
12         </m:GetCustomer>
13       </soap:Body>
14     </soap:Envelope>
```

Listing A.2: The SOAP HTTP service provider response for Listing A.1. Source: [22].

```

1 HTTP/1.1 200 OK
2 Content-Type: application/soap+xml; charset=utf-8
3
4 <?xml version='1.0' ?>
5 <env:Envelope
6   xmlns:env="http://www.w3.org/2003/05/soap-envelope" >
7     <env:Body>
8       <m:GetCustomerResponse
9         xmlns:m="http://www.example.org/customers">
10           <m:Customer>Foobar Quux, inc</m:Customer>
11         </m:GetCustomerResponse>
12       </env:Body>
13     </env:Envelope>
```

SOAP uses the architectural principle that web services (or the applications they provide) should remain *outside* the web, using HTTP only as a tunnelling protocol to enable remote communication [271]. That is, the HTTP is considered as a transport

protocol solely. In 2000, Fielding [116] introduced REpresentational State Transfer (REST), which instead approaches the web as a medium to publish data (i.e., HTTP is part of the *application* layer instead). Hence, applications become amalgamated into of the Web. Fielding bases REST on four key principles:

- **URIs identify resources.** Resources and services have a consistent global address space that aides in their discovery via URIs [35].
- **HTTP verbs manipulate those resources.** Resources are manipulated using the four consistent CRUD verbs provided by HTTP: POST, GET, PUT, DELETE.
- **Self-descriptive messages.** Each request provides enough description and context for the server to process that message.
- **Resources are stateless.** Every interaction with a resource is stateless.

Consider the equivalent example of Listings A.1 and A.2 but in a RESTful architecture (Listings A.3 and A.4) and it is clear why this style has grown more popular with developers (as we highlight in Figure A.1). Developers have since embraced RESTful application programming interface (API) development, though the major drawback of RESTful services is its lack of a uniform IDL to facilitate development (though it is possible to achieve this using Web Application Description Language (WADL) [225]). Therefore, no RESTful service uses a standardised response document or invocation syntax. While there are proposals, such as WADL [146], RAML¹, API Blueprint², and the OpenAPI³ specification (initially based on Swagger⁴), there is still no consensus as there was for SOAP and convergence of these IDLs is still underway.

Listing A.3: An equivalent HTTP consumer request to that of Listing A.1, but using REST.
Source: [22].

```
1 | GET /customers/43456 HTTP/1.1
2 | Host: www.example.org
```

Listing A.4: The REST HTTP service provider response for Listing A.3.

```
1 | HTTP/1.1 200 OK
2 | Content-Type: application/json; charset=utf-8
3 |
4 | {"Customer": "Foobar Quux, inc"}
```

¹<https://raml.org> last accessed 25 January 2019.

²<https://apiblueprint.org> last accessed 25 January 2019.

³<https://www.openapis.org> last accessed 25 January 2019.

⁴<https://swagger.io> last accessed 25 January 2019.

A.2 Additional Figures

The following figures are listed in this section:

- **Figure A.2 (p240)** is a reproduction of Lo Giudice et al. [219]’s report on how AI will re-shape applications. The authors produce four primary categories and list sample products, vendors and use cases. This image was originally included within Chapter 2.
- **Figure A.3 (p241)** highlights an increasing trend of CVS usage measured as discussion of posts that mention a product name. This graph was originally included within Chapter 5 based on the posts extracted from this study.
- **Figure A.4 (p242)** highlights potential causal factors that may influence a developer’s understanding of the documentation and response of IWSs. It was intended to be used as the basis of a survey study in Chapter 8, and can be used for future avenues of research.
- **Figure A.5 (p243)** was intended for the discussion in Chapter 5, where we propose that developers have a misaligned of the technical domain models within IWSs and more specifically CVSs. We designed a draft technical domain model to describe the various aspects developers must consider when using these services, based on the work by Barnett [25].
- **Figure A.6 (p244)** describes potential questions that may arise to analyse and test the causal factors of the technical domain model proposed in Figure A.5. This lies an open avenue of future research.
- **Figure A.7 (p244)** emphasises dichotomy between an application using an IWS and the IWS’ training data (which is sourced from an unknown context) and the context of an application, which is known. This is to emphasise how the model produced from these services need to be calibrated to the application domain being used in order for the decision boundary of a single inference to be properly assessed by the developer. This image was originally included within the Threshy publication (Chapter 10) but was removed due to space limitations.
- **Figure A.8 (p245)** illustrates the domain model of Threshy (Chapter 10).
- **Figure A.9 (p245)** illustrates the dynamic model of using Threshy and its interactions between the application, front-end of Threshy and back-end of Threshy (Chapter 10).
- **Figure A.10 (p246)** was originally included within the publication Chapter 5 but was removed due to space limitations. It provides a high-level overview of the main steps we performed within this study.
- **Figure A.11 (p247)** is a class diagram of the reference architecture of the proposed architecture in Chapter 11. The implementation is provided in Appendix B. See Chapter 11 for more.
- **Figure A.12 (p248)** is a sequence diagram illustrating how the reference architecture can be used to create a new benchmark as per the implementation provided in Appendix B. See Chapter 11 for more.

- **Figure A.13 (p249)** is a sequence diagram illustrating how applications can make requests to the proxy server ‘facade’ as per the implementation provided in Appendix B. See Chapter 11 for more.
- **Figure A.14 (p250)** is a state diagram that illustrates the overall states that exist within the architecture tactic’s workflows. See Chapter 11 for more.
- **Figure A.15 (p251)** is a sequence diagram illustrating how the reference architecture handles evolution in an external service per the implementation provided in Appendix B. See Chapter 11 for more.
- **Figure A.16 (p252)** illustrates how the reference architecture is able to capture and handle three requests (two valid, one invalid) when sent to the proxy server. See Chapter 11 for more.

Figure A.2: A Broad Range of AI-Based Products And Services Is Already Visible. (From [219].)

Category	Sample vendors and products	Typical use cases
Embedded AI Expert assistants leverage AI technology embedded in platforms and solutions.	<ul style="list-style-type: none"> Amazon: Alexa Apple: Siri Facebook: Messenger Google: Google Assistant (and more) Microsoft: Cortana Salesforce: MetaMind (acquisition) 	<ul style="list-style-type: none"> Personal assistants for search, simple inquiry, and growing as expert assistance (composed problems, not just search) Available on mobile platforms, devices, the internet of things Voice, image recognition, various levels of NLP sophistication Bots, agents
AI point solutions Point solutions provide specialized capabilities for NLP, vision, speech, and reasoning.	<ul style="list-style-type: none"> 24[7]: 24[7] Admantx: Admantx Affectiva: Affdex Assist: AssistDigital Automated Insights: Wordsmith Beyond Verbal: Beyond Verbal Expert System: Cogito HPE: Haven OnDemand IBM: Watson Analytics, Explorer, Advisor Narrative Science: Quill Nuance: Dragon Salesforce: MetaMind (acquisition) Wise.io: Wise Support 	<ul style="list-style-type: none"> Semantic text, facial/visual recognition, voice intonation, intelligent narratives Various levels of NLP from brief text messaging, chat/conversational messaging, full complex text understanding Machine learning, predictive analytics, text analytics/mining Knowledge management and search Expert advisors, reasoning tools Customer service, support APIs
AI platforms Platforms that offer various AI tech, including (deep) machine learning, as tools, APIs, or services to build solutions.	<ul style="list-style-type: none"> CognitiveScale: Engage, Amplify Digital Reasoning: Synthesys Google: Google Cloud Machine Learning IBM: Watson Developers, Watson Knowledge Studio Intel: Saffron Natural Intelligence IPsoft: Amelia, Apollo, IP Center Microsoft: Cortana Intelligence Suite Nuance: 360 platform Salesforce: Einstein Wipro: Holmes 	<ul style="list-style-type: none"> APIs, cloud services, on-premises for developers to build AI solutions Insights/advice building Rule-based reasoning Vertical domain advisors (e.g., fraud detection in banking, financial advisors, healthcare) Cognitive services and bots
Deep learning Platforms, advanced projects, and algorithms for deep learning.	<ul style="list-style-type: none"> Amazon: FireFly Google: TensorFlow/DeepMind LoopAI Labs: LoopAI Numenta: Grok Vicarious: Vicarious 	<ul style="list-style-type: none"> Deep learning neural networks for categorization, clustering, search, image recognition, NLP, and more Location pattern recognition Brain neocortex simulation

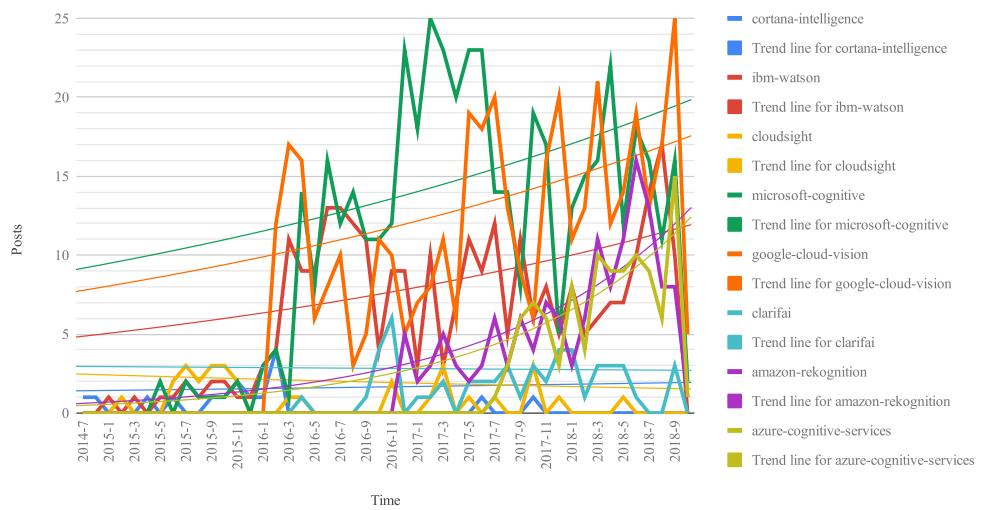
Figure A.3: Increasing interest on Stack Overflow for CVSSs.

Figure A.4: Causal factors that may influence understanding of intelligent web services.

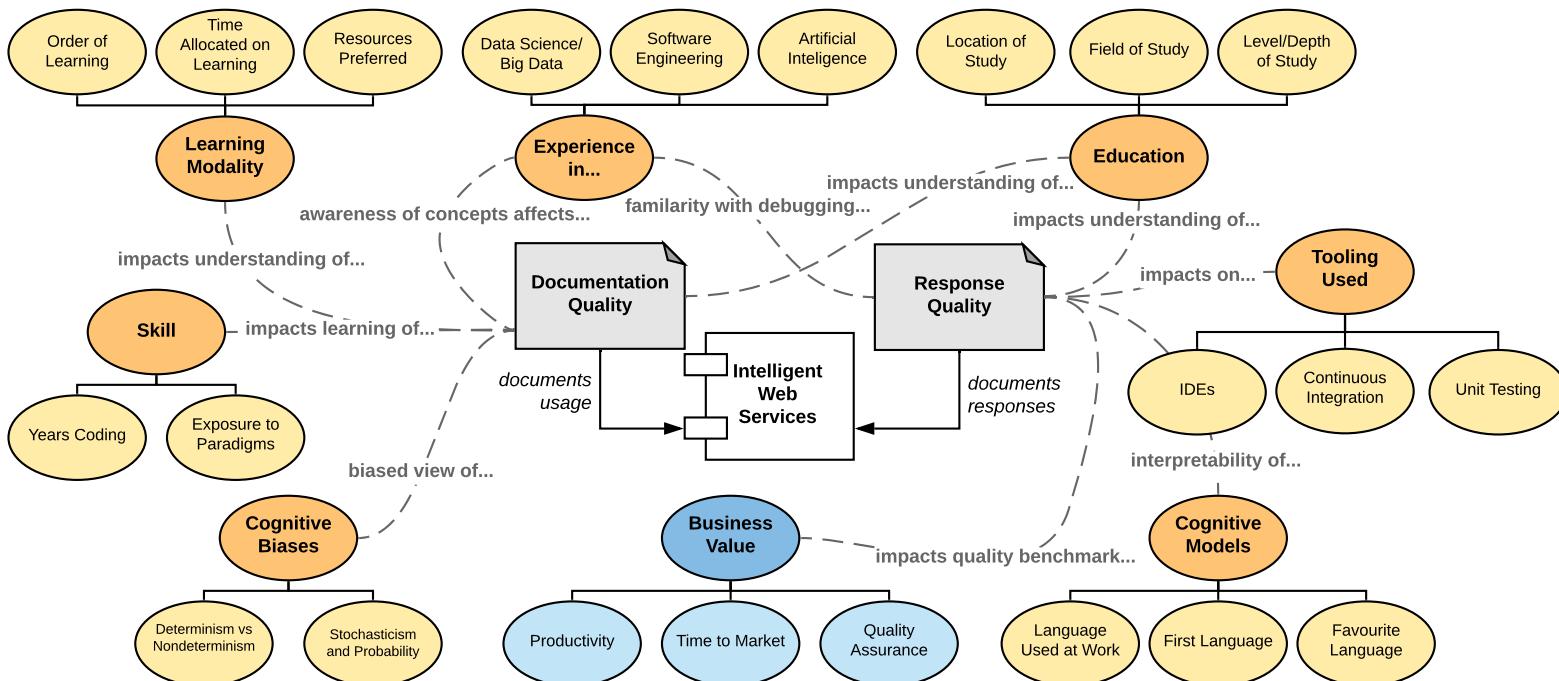


Figure A.5: A proposal technical domain model for intelligent services. (The ⓘ symbol indicates computer vision related services only.)

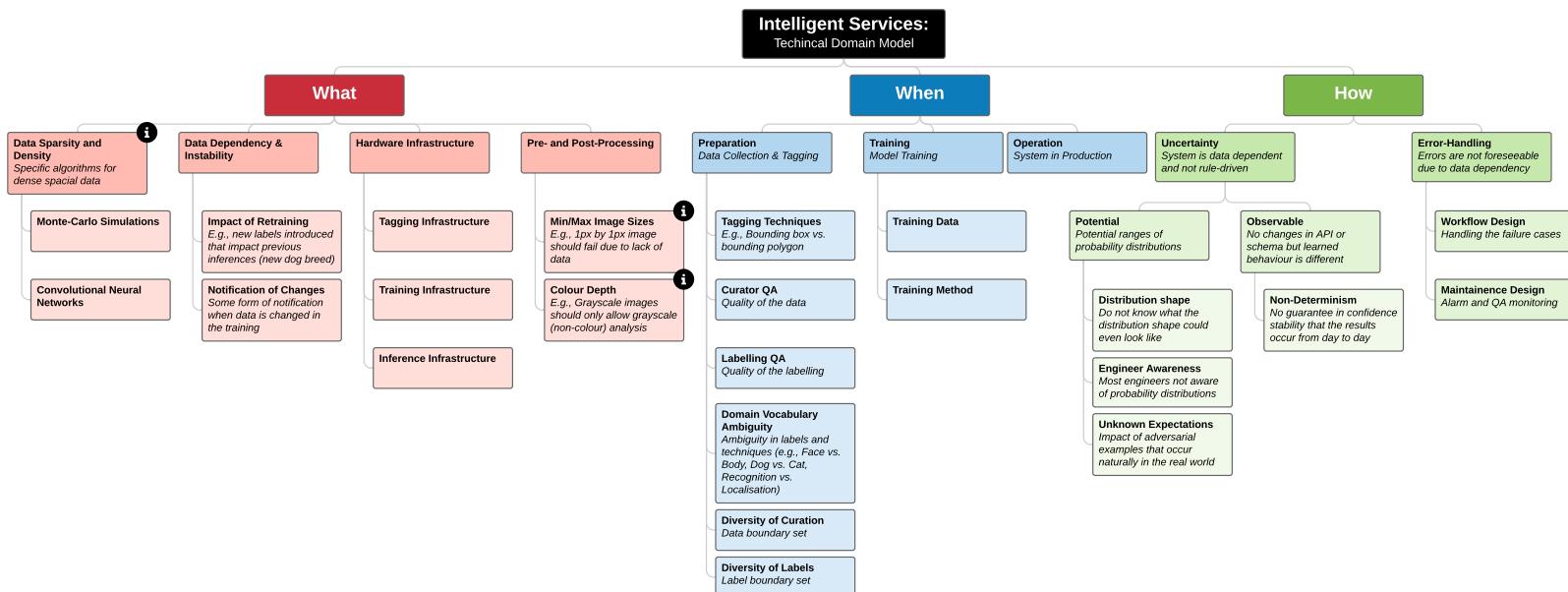


Figure A.6: Potential questions that can be asked around causal factors of a developer's understanding of an intelligent service.

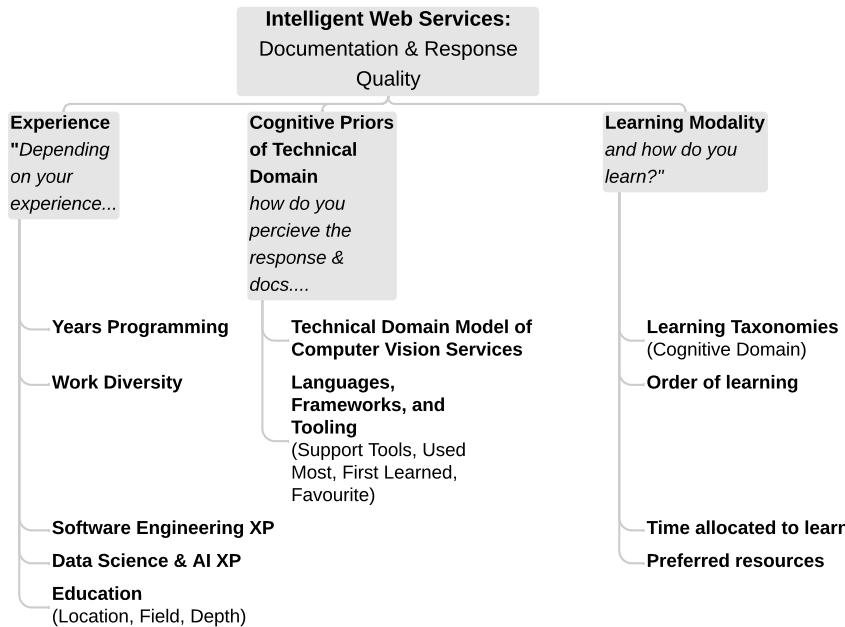


Figure A.7: Threshy assists with making appropriate decision boundaries in the application context by calibrating model (train on an unknown context) to your domain.

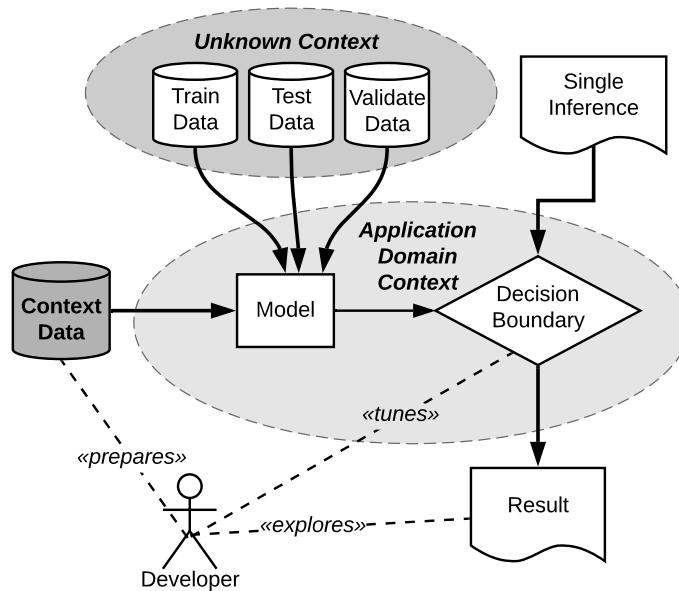


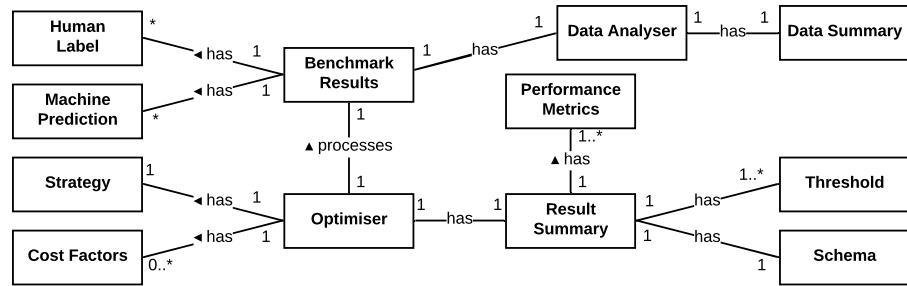
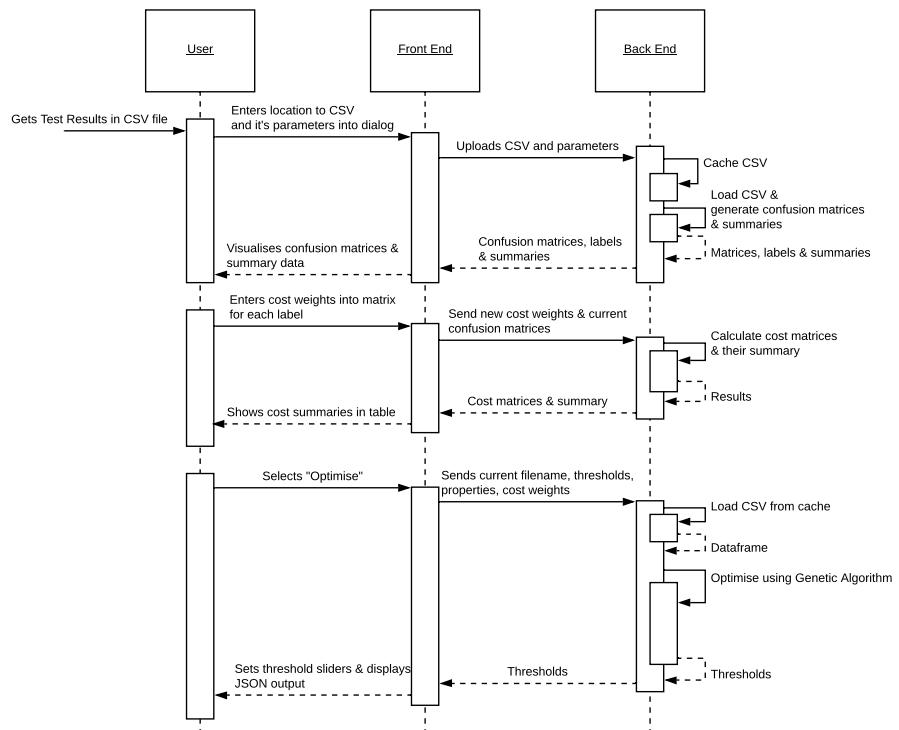
Figure A.8: Threshy domain model.**Figure A.9:** High level overview of Threshy's interaction between the front- and back-end.

Figure A.10: High-level overview of the methodology within Chapter 5.

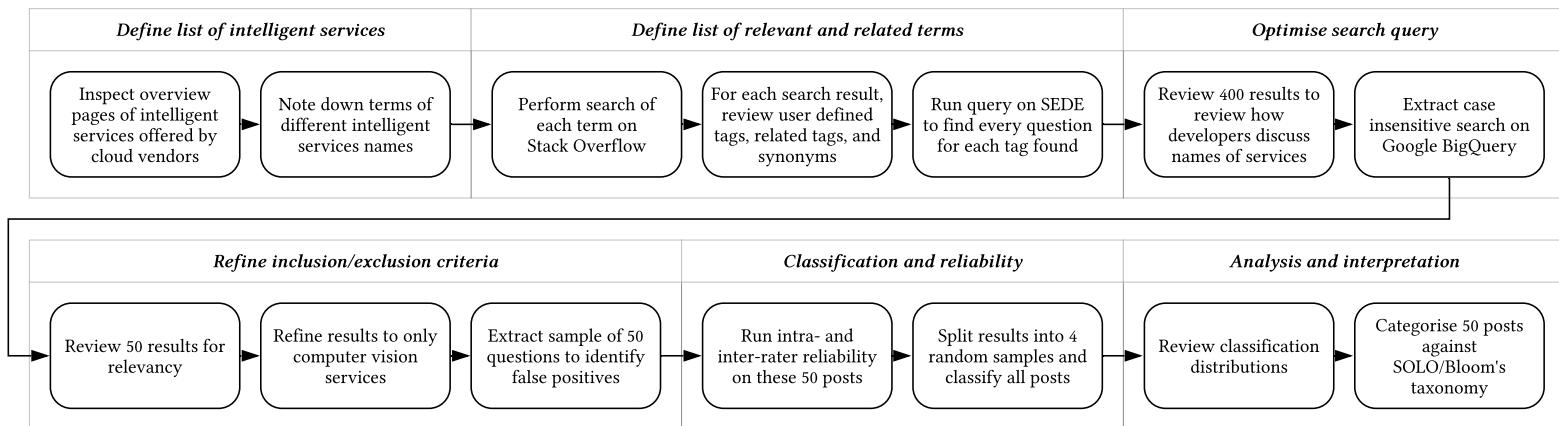


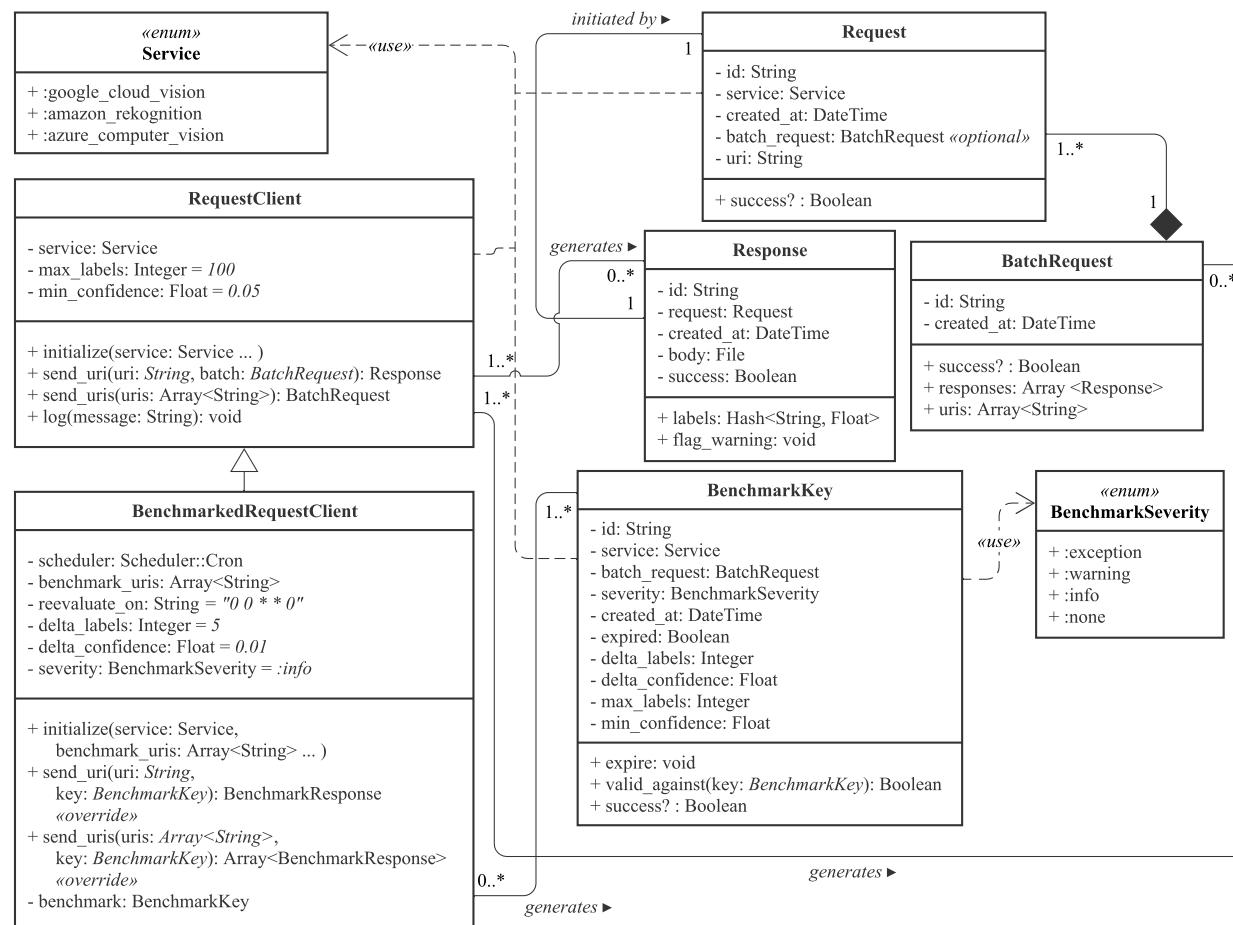
Figure A.11: Class diagram of the implementation of our architecture.

Figure A.12: Creation of a new benchmark proxy server using the architecture tactic.

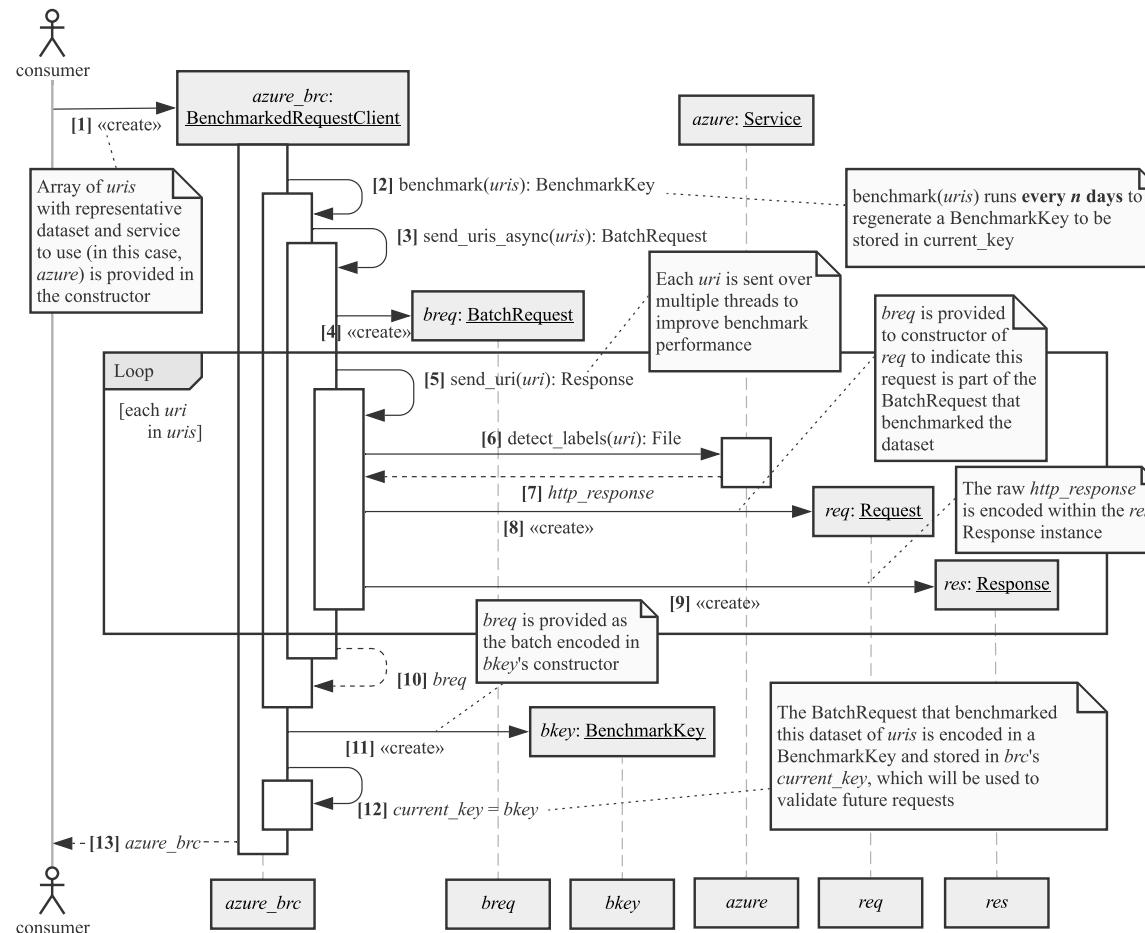


Figure A.13: Making a request through the proxy server ‘facade’.

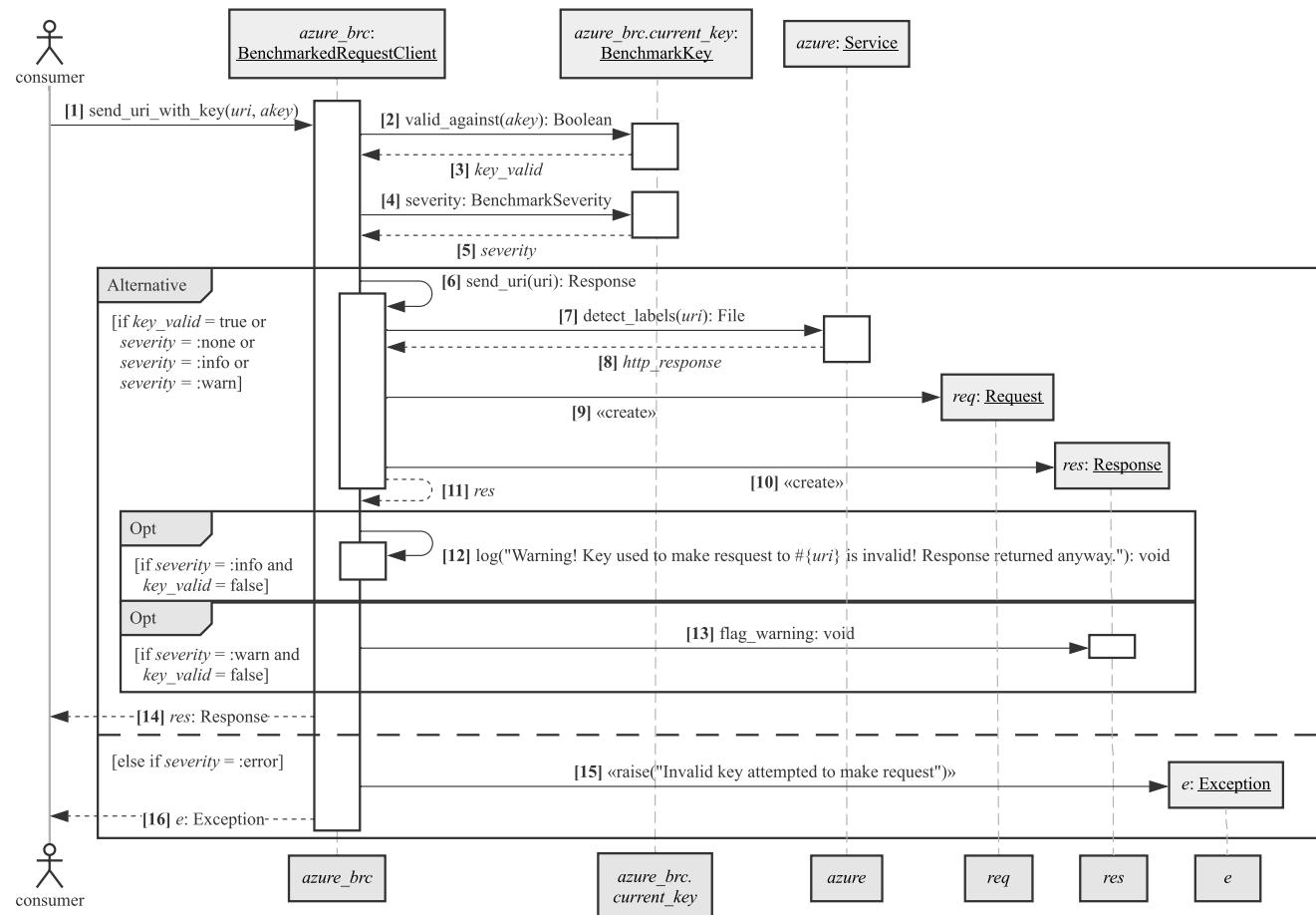


Figure A.14: State diagram of high-level workflows in the architectural tactic.

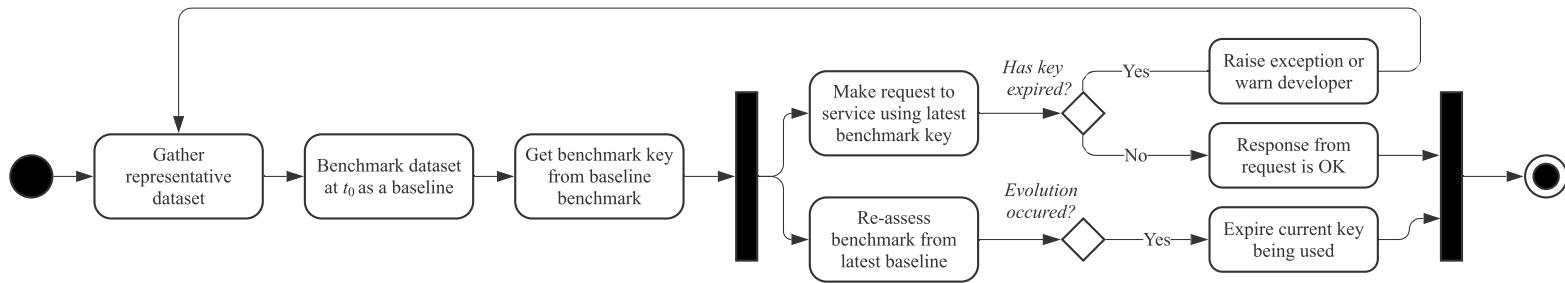


Figure A.15: Evolution occurring in the benchmark and how the architectural tactic notifies the consumer.

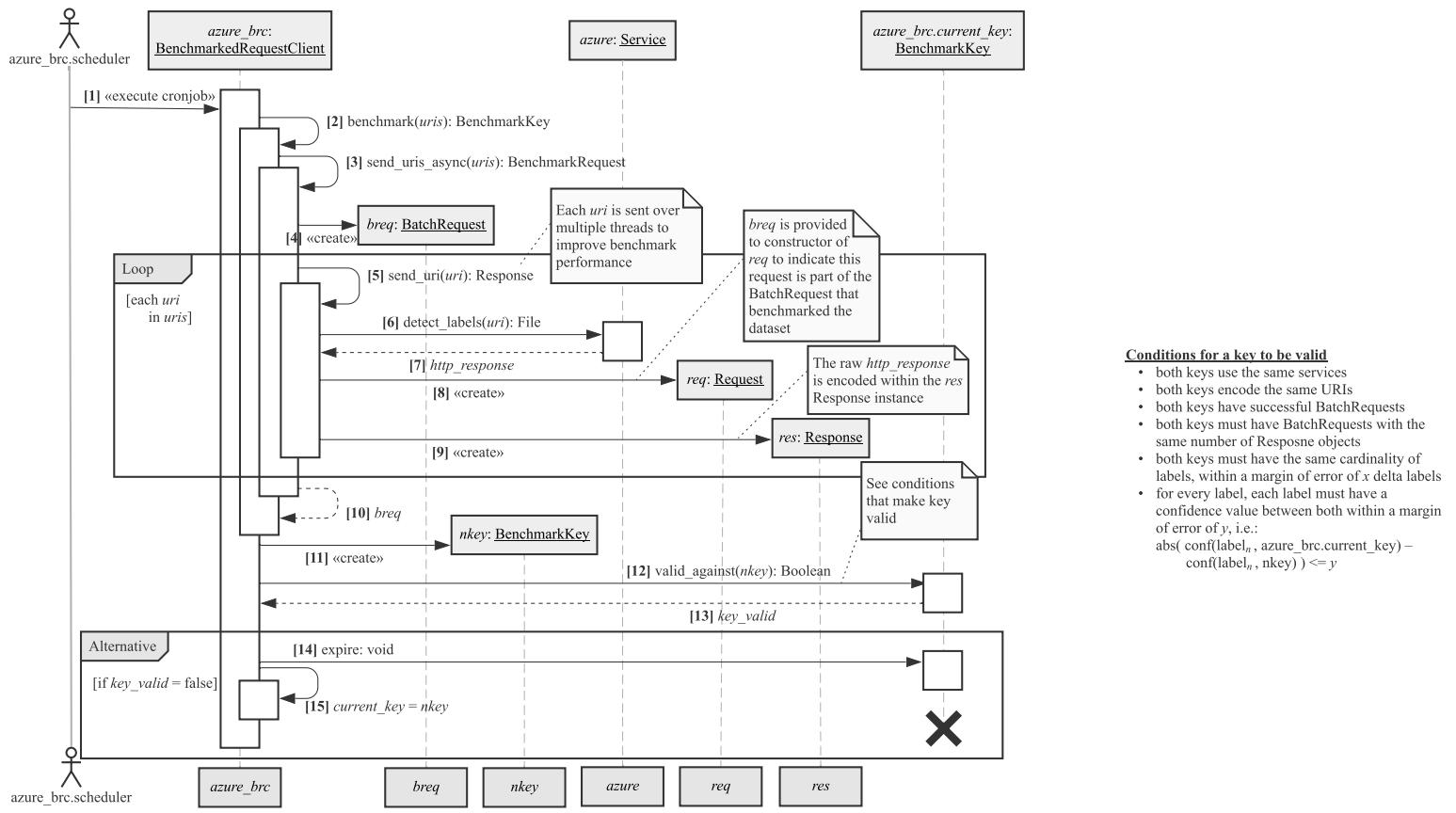
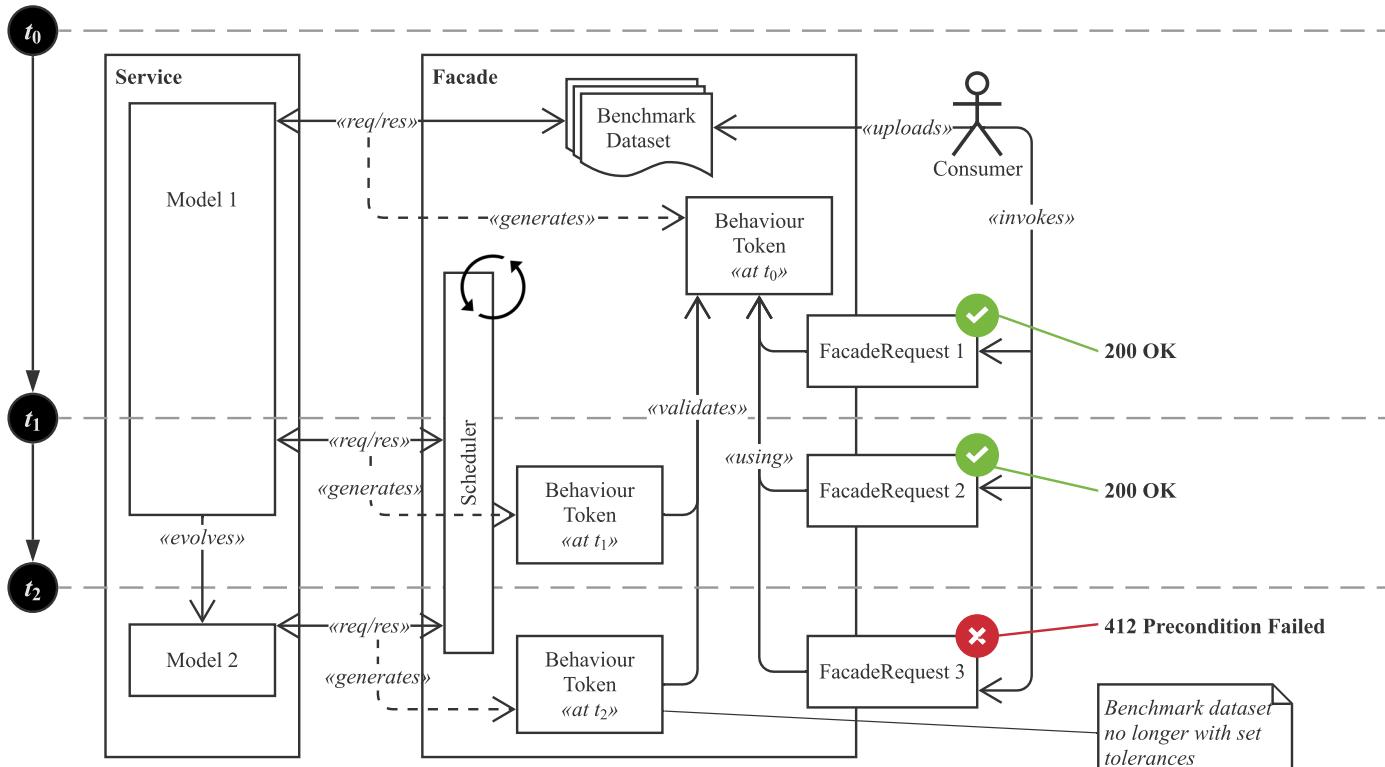


Figure A.16: Evolution occurring in an intelligent service and how the architectural tactic handles it.



APPENDIX B

Reference Architecture Source Code

Listing B.1: Implementation of architecture module components.

```
1 # frozen_string_literal: true
2
3 # Author:: Alex Cummaudo (mailto:ca@deakin.edu.au)
4 # Copyright:: Copyright (c) 2019 Alex Cummaudo
5 # License:: MIT License
6
7 require 'sequel'
8 require 'logger'
9 require 'stringio'
10 require 'binding_of_caller'
11 require 'dotenv/load'
12 require 'google/cloud/vision'
13 require 'aws-sdk-rekognition'
14 require 'net/http/post/multipart'
15 require 'down'
16 require 'uri'
17 require 'json'
18 require 'tempfile'
19 require 'rufus-scheduler'
20
21 # Intelligent Computer Vision Service Benchmarker (ICVSB) module. This module
22 # implements an architectural pattern that helps overcome evolution issues
23 # within intelligent computer vision services.
24 module ICVSB
25   Thread.abort_on_exception = true
26   # The valid services this version of the ICVSB module supports. At present the
27   # only services supported are Google Cloud Vision, Amazon Rekognition, and
28   # Azure Computer Vision and their respective labelling/tagging endpoints. You
29   # can also request the demo.
30   # @see https://cloud.google.com/vision/docs/labels
31   # Google Cloud Vision labelling endpoint.
32   # @see https://docs.aws.amazon.com/rekognition/latest/dg/API_DetectLabels.html
33   # Amazon Rekognition's labelling endpoint.
34   # @see https://docs.microsoft.com/en-us/rest/api/cognitiveservices/
35   # computervision/tagimage/tagimage
```

```

35  # Azure Computer Vision's tagging endpoint.
36  VALID_SERVICES = %i[google_cloud_vision amazon_rekognition
37      ↪ azure_computer_vision demo].freeze
38
39  # A list of the valid severities that the ICVSB module supports. Exception
40  # prevents the response from being accessed; warning will still produce a
41  # response but the +error+ field will be filled in; info will only log
42  # errors to the ICVSB log file and keep +error+ empty and none ignores the
43  # errors entirely.
44  VALID_SEVERITIES = %i[exception warning info none].freeze
45
46  # Logs a message to the global ICVSB logger. If called from within the
47  # stack trace of a RequestClient, it will also add the message provided
48  # the RequestClient's log associated with the RequestClient's object id.
49  # @param [Logger::Severity] severity The type of severity to log.
50  # @param [String] message The message to log.
51  def self.lmessage(severity, message)
52      unless [Logger::DEBUG, Logger::INFO, Logger::WARN, Logger::ERROR, Logger::
53          ↪ FATAL, Logger::UNKNOWN].include?(severity)
54          raise ArgumentError, 'Severity must be a Logger::Severity type'
55      end
56      raise ArgumentError, 'Message must be a string' unless message.is_a?(String)
57
58      @log ||= Logger.new(ENV['ICVSB_LOGGER_FILE'] || STDOUT)
59
60      # Add message to global ICVSB logger
61      @log.add(severity, message)
62
63      # Find object_id within request_clients... when found add this message w/
64      # severity to that RC's log too
65      binding.frame_count.times do |n|
66          caller_obj_id = binding.of_caller(n).eval('object_id')
67          if @request_clients.keys.include?(caller_obj_id)
68              @request_clients[caller_obj_id].log(severity, "[RequestClient=#{
69                  ↪ caller_obj_id}] #{message}")
70          end
71      end
72
73      # Logs an error to the global ICVSB logger.
74      # @param [String] message The message to log.
75      def self.lerror(message)
76          lmessage(Logger::ERROR, message)
77
78      # Logs a warning to the global ICVSB logger.
79      # @param [String] message The message to log.
80      def self.lwarn(message)
81          lmessage(Logger::WARN, message)
82
83      # Logs an info message to the global ICVSB logger.
84      # @param [String] message The message to log.
85      def self.linfo(message)
86          lmessage(Logger::INFO, message)
87
88      # Logs a debug message to the global ICVSB logger.
89      # @param [String] message The message to log.
90      def self.ldebug(message)
91          lmessage(Logger::DEBUG, message)
92
93      # Register's a request client to the ICVSB's register of request clients.

```

```

96  # @param [RequestClient] request_client The request client to register.
97  def self.register_request_client(request_client)
98      raise ArgumentError, 'request_client must be a RequestClient' unless
99          ↪ request_client.is_a?(RequestClient)
100
101     @request_clients ||= {}
102     @request_clients[request_client.object_id] = request_client
103 end
104 #####
105 # Database schema creation seed #
106 #####
107 url = ENV['ICVSB_DATABASE_CONNECTION_URL'] || 'sqlite://icvsb.db'
108 log = ENV['ICVSB_DATABASE_LOG_FILE'] || 'icvsb.db.log'
109 dbc = Sequel.connect(url, logger: Logger.new(log))
110 # Create Services and Severity enums...
111 dbc.create_table?(:services) do
112     primary_key :id
113     column :name, String, null: false, unique: true
114 end
115 dbc.create_table?(:benchmark_severities) do
116     primary_key :id
117     column :name, String, null: false, unique: true
118 end
119 if dbc[:services].first.nil?
120     VALID_SERVICES.each { |s| dbc[:services].insert(name: s.to_s) }
121     VALID_SEVERITIES.each { |s| dbc[:benchmark_severities].insert(name: s.to_s) }
122 end
123 # Create Objects...
124 dbc.create_table?(:batch_requests) do
125     primary_key :id
126     column :created_at, DateTime, null: false
127 end
128 dbc.create_table?(:requests) do
129     primary_key :id
130     foreign_key :service_id, :services, null: false
131     foreign_key :batch_request_id, :batch_requests, null: true
132     foreign_key :benchmark_key_id, :benchmark_keys, null: true
133
134     column :created_at, DateTime, null: false
135     column :uri, String, null: false
136
137     index %i[service_id batch_request_id]
138 end
139 dbc.create_table?(:responses) do
140     primary_key :id
141     foreign_key :request_id, :requests, null: false
142
143     column :created_at, DateTime, null: false
144     column :body, File, null: true
145     column :success, TrueClass, null: false
146
147     index :request_id
148 end
149 dbc.create_table?(:benchmark_keys) do
150     primary_key :id
151     foreign_key :service_id, :services, null: false
152     foreign_key :batch_request_id, :batch_requests, null: false
153     foreign_key :benchmark_severity_id, :benchmark_severities, null: false
154
155     column :created_at, DateTime, null: false
156     column :expired, TrueClass, null: false
157     column :delta_labels, Integer, null: false
158     column :delta_confidence, Float, null: false

```

```

159   column :max_labels, Integer, null: false
160   column :min_confidence, Float, null: false
161   column :expected_labels, String, null: true
162
163   index %i[service_id batch_request_id]
164 end
165
166 # Service representing the list of VALID_SERVICES the ICVSB module supports.
167 class Service < Sequel::Model(dbc)
168   # The Service representing Google Cloud Vision's labelling endpoint.
169   # @see https://cloud.google.com/vision/docs/labels
170   # Google Cloud Vision labelling endpoint.
171   GOOGLE = Service[name: VALID_SERVICES[0].to_s]
172
173   # The Service representing Amazon Rekognition's labelling endpoint.
174   # @see https://docs.aws.amazon.com/rekognition/latest/dg/API_DetectLabels.html
175   # Amazon Rekognition's labelling endpoint.
176   AMAZON = Service[name: VALID_SERVICES[1].to_s]
177
178   # The Service representing Azure Computer Vision's tagging endpoint.
179   # @see https://docs.microsoft.com/en-us/rest/api/cognitiveservices/
180       → computervision/tagimage/tagimage
181   # Azure Computer Vision's tagging endpoint.
182   AZURE = Service[name: VALID_SERVICES[2].to_s]
183
184   # The Service representing a demonstration of the facade.
185   DEMO = Service[name: VALID_SERVICES[3].to_s]
186 end
187
188 # Severity representing the list of VALID_SEVERITIES the ICVSB module
189 # supports. The severity is encoded within a BenchmarkKey.
190 class BenchmarkSeverity < Sequel::Model(dbc[:benchmark_severities])
191   # Exception severities will prevent responses from being accessed. This
192   # disallows access to the Response object encoded within a
193   # BenchmarkedRequestClient#send_uri_with_key or
194   # BenchmarkedRequestClient#send_uris_with_key result.
195   EXCEPTION = BenchmarkSeverity[name: VALID_SEVERITIES[0].to_s]
196
197   # Warning severities will allow the Response from being accessed but will
198   # additionally populate the +error+ value encoded within a
199   # BenchmarkedRequestClient#send_uri_with_key or
200   # BenchmarkedRequestClient#send_uris_with_key result.
201   WARNING = BenchmarkSeverity[name: VALID_SEVERITIES[1].to_s]
202
203   # Info severities will allow the Response from being accessed encoded within
204   # the result of a BenchmarkedRequestClient#send_uri_with_key or
205   # BenchmarkedRequestClient#send_uris_with_key call, however, information
206   # pertaining to issues with the request will be logged to the ICVSB log
207   # file.
208   INFO = BenchmarkSeverity[name: VALID_SEVERITIES[2].to_s]
209
210   # None severities will essentially ignore all benchmarking capabilities and
211   # 'switches off' the benchmarking.
212   NONE = BenchmarkSeverity[name: VALID_SEVERITIES[3].to_s]
213
214   # Overrides the to_s method to return the name.
215   # @return [String] The name of the severity type.
216   def to_s
217     name
218   end
219 end
220
221   # This class represents a single request made to a Service. It encodes the
222   # service, batch of requests (if applicable) and respective response.

```

```

222  class Request < Sequel::Modeldbc)
223    many_to_one :service
224    many_to_one :batch
225    many_to_one :benchmark_key
226    one_to_one :response
227
228    # @see Response#success.
229    def success?
230      response.success?
231    end
232  end
233
234  # This class represents a single response returned back from a Service. It
235  # encodes the request that was made to invoke the response.
236  class Response < Sequel::Modeldbc)
237    many_to_one :request
238
239    # Indicates if the response from the request was successful.
240    # @return [Boolean] True if the response was successful or false if the
241    # response contained some issue.
242    def success?
243      success
244    end
245
246    # Returns a hash of the entire response object, decoded from its
247    # Service-specific response Ruby type and into a simple hash object.
248    # @return [Hash] A hash representing the entire Service response object
249    # within a Hash type.
250    def hash
251      return nil if body.nil?
252
253      JSON.parse(body.lit.downcase.to_s, symbolize_names: true).to_h
254    end
255
256    # Returns hash of labels paired with their respective confidence values.
257    # Decodes each Service's individual response syntax into a simple
258    # key-value-pair that can be used for generalised use, regardless of which
259    # Service actually generated the response.
260    # @return [Hash] A hash with key-value-pairs representing the label (key)
261    # and value (confidence) of the response.
262    def labels
263      if success?
264        case request.service
265        when Service::GOOGLE
266          _google_cloud_vision_labels
267        when Service::AMAZON
268          _amazon_rekognition_labels
269        when Service::AZURE
270          _azure_computer_vision_labels
271        when Service::DEMO
272          _demo_service_labels
273        end
274      else
275        {}
276      end
277    end
278
279    # Returns the benchmark key ID of the request.
280    # @return [Integer] The benchmark key id of this response's request.
281    def benchmark_key_id
282      request.benchmark_key.id
283    end
284
285    # Returns the benchmark key of the request.

```

```

286  # @return [BenchmarkKey] The benchmark key of this response's request.
287  def benchmark_key
288    request.benchmark_key
289  end
290
291  # Sets the benchmark key of the request.
292  # @param [BenchmarkKey] value The new benchmark key to set.
293  # @return [void]
294  def benchmark_key=(value)
295    request.benchmark_key = value
296    request.save
297  end
298
299  # Sets the benchmark key id of the request.
300  # @param [Integer] value The new benchmark key id to set.
301  # @return [void]
302  def benchmark_key_id=(value)
303    request.benchmark_key_id = value
304    request.save
305  end
306
307  private
308
309  # Decodes a Google Cloud Vision label endpoint response into a simple hash.
310  # @return [Hash] A key-value-pair representing label => confidence.
311  def _google_cloud_vision_labels
312    hash[:responses][0][:label_annotations].map do |label|
313      [label[:description].downcase, label[:score]]
314    end.to_h
315  end
316
317  # Decodes an Amazon Rekognition label endpoint response into a simple hash.
318  # @return [Hash] See #{#_google_cloud_vision_labels}.
319  def _amazon_rekognition_labels
320    hash[:labels].map do |label|
321      [label[:name].downcase, label[:confidence] * 0.01]
322    end.to_h
323  end
324
325  # Decodes an Azure Computer Vision tagging endpoint into a simple hash.
326  # @return [Hash] See #{#_google_cloud_vision_labels}.
327  def _azure_computer_vision_labels
328    hash[:tags].map do |label|
329      [label[:name].downcase, label[:confidence]]
330    end.to_h
331  end
332
333  # Decodes the mock demo service response into a simple hash. This is simply
334  # a relay of Google's as the data is from Google Cloud Vision.
335  # @return [Hash] A key-value-pair representing label => confidence.
336  def _demo_service_labels
337    _google_cloud_vision_labels
338  end
339  end
340
341  # The batch request class collates multiple requests (URIs) invoked to a
342  # single Service's endpoint in a single request. It encodes all requests
343  # made to the service and can produce all responses back.
344  class BatchRequest < Sequel::Model(:dbc)
345    one_to_many :requests
346
347    # Indicates if every request in the batch of requests made were successful.
348    # @return [Boolean] True if every response was successful, false
349    # otherwise.

```

```

350
351     def success?
352         requests.map(&:success?).reduce(:&)
353     end
354
355     # Maps all Response objects that were returned back from this batch to an
356     # array.
357     # @return [Array<Response>] An array of Response objects from every Request
358     # made in this batch.
359     def responses
360         requests.map(&:response)
361     end
362
363     # Maps all URIs that were requested back within this batch.
364     # @return [Array<String>] An array of URI strings from every Request
365     # made in this batch.
366     def uris
367         requests.map(&:uri)
368     end
369 end
370
371 # The Benchmark Key encodes all information pertaining to the evolution of a
372 # specific service and is used to validate if a benchmark dataset has evolved
373 # with time. This key must be used in conjunction with the
374 # BenchmarkedRequestClient to ensure that responses made are still reasonable
375     # to
376 # use or if the service should be re-benchmarked against a new dataset.
377 class BenchmarkKey < Sequel::Model(dbc)
378     many_to_one :service
379     many_to_one :benchmark_severity
380     many_to_one :batch_request
381
382     # Class that encapsulates reasons why a benchmark key can be invalidated.
383     class InvalidKeyError
384         module InvalidKeyErrorType
385             NO_KEY_YET = 'No key yet exists. It is likely key is still benchmarking
386             # its first results.'
387             SERVICE_MISMATCH = 'Keys use different services'
388             DATASET_MISMATCH = 'Keys have different benchmark datasets'
389             SUCCESS_MISMATCH = 'One or both keys do not have successful service
390             # responses'
391             MIN_CONFIDENCE_MISMATCH = 'Keys have different min confidence values'
392             MAX_LABELS_MISMATCH = 'Keys have different max label values'
393             RESPONSE_LENGTH_MISMATCH = 'Keys have different number of responses'
394             LABEL_DELTA_MISMATCH = 'Number of labels in one key exceeds the label
395             # delta threshold'
396             CONFIDENCE_DELTA_MISMATCH = 'Confidence value for a label in one key
397             # exceeds the confidence delta threshold'
398             EXPECTED_LABELS_MISMATCH = 'Expected labels missing from response'
399         end
400
401         include InvalidKeyErrorType
402         attr_reader :errorname, :errorcode, :data
403
404         def initialize(errorrtype, data = '')
405             @errorname = InvalidKeyErrorType.constants.find { |c| InvalidKeyErrorType.
406                 # const_get(c) == errorrtype }
407             @errorcode = InvalidKeyErrorType.constants.index(@errorname)
408             @data = data
409         end
410
411         def to_s
412             "[#{@errorcode}]:#{@errorname}] #{@data}"
413         end
414     end
415 end

```

```

408     def to_h
409     {
410       error_code: @errorcode,
411       error_type: @errorname,
412       error_data: @data
413     }
414   end
415 end
416
417 # @see BatchRequest#success?
418 def success?
419   batch_request.success?
420 end
421
422 # An alias for the +expired+ field on the key, adding a question mark at the
423 # end to make the field more 'Ruby-esque'.
424 # @return [Boolean] True if the key has expired and thus should not be used
425 # for future requests as it is no longer valid.
426 def expired?
427   expired
428 end
429
430 # Expires this key by writing over its +expired+ field and marking it
431 # true.
432 # @return [void]
433 def expire
434   self.expired = true
435   save
436 end
437
438 # Un-expires this key by writing over its +expired+ field and marking it
439 # true.
440 # @return [void]
441 def unexpire
442   self.expired = false
443   save
444 end
445
446 # Returns the comma-separated mandatory labels list as an set of values
447 # @return [Set<String>] The set of mandatory labels required by this key.
448 def expected_labels_set
449   Set[*expected_labels.split(',').map(&:downcase)]
450 end
451
452 # Validates another key against this key to ensure if the two keys are
453 # compatible or if evolution has occurred iff BenchmarkKey is provided to
454 # +key_or_response+. If a Response is provided instead, then validates that
455 # the response is okay against this key's encoded parameters.
456 # @param [BenchmarkKey,Response] key_or_response A key or response to
457 # validate against.
458 # @return [Array<Boolean,Array<BenchmarkKey::InvalidKeyError>>] Returns +true+
459 # if
460 # this key is valid against the other key OR a tuple with +false+ and
461 # BenchmarkKey::InvalidKeyError to explain why the key is invalid.
462 def valid_against?(key_or_response)
463   if key_or_response.is_a?(BenchmarkKey)
464     _validate_against_key(key_or_response)
465   elsif key_or_response.is_a?(Response)
466     _validate_against_response(key_or_response)
467   else
468     raise ArgumentError, 'key_or_response must be a BenchmarkKey or Response
469     ↪ type'
470   end
471 end

```

```

470
471     private
472
473     # Validates a key against this key as per rules encoded within this key.
474     # @param [BenchmarkKey] key The key to validate.
475     # @return See #valid_against?
476     def _validate_against_key(key)
477       ICSVSB.linfo("Validating key id=#{id} with other key id=#{key.id}")
478
479       # True if same key id...
480       return true if key == self
481
482       invalid_key_errors = []
483
484       # 1. Ensure same services!
485       if key.service == service
486         ICSVSB.ldebug('Services both match')
487       else
488         ICSVSB.lwarn("Service mismatch in validation: #{key.service.name} != #{service.name}")
489         invalid_key_errors << BenchmarkKey::InvalidKeyError.new(
490           BenchmarkKey::InvalidKeyError::SERVICE_MISMATCH, {
491             source_key: {
492               id: id,
493               created_at: created_at,
494               service_name: service.name
495             },
496             violating_key: {
497               id: key.id,
498               created_at: key.created_at,
499               service_name: key.service.name
500             },
501             message: "Source key (id=#{id}) service=#{service.name} but \"\n"
502               "validation key (id=#{key.id}) service=#{key.service.name}.\n"
503           }
504         )
505       end
506
507       # 2. Ensure same benchmark dataset
508       symm_diff_uris = Set[*batch_request.uris] ^ Set[*key.batch_request.uris]
509       if symm_diff_uris.empty?
510         ICSVSB.ldebug('Same benchmark dataset has been used')
511       else
512         ICSVSB.lwarn('Benchmark dataset mismatch in key validation: '\
513           "Symm difference contains #{symm_diff_uris.count} different URIs")
514         invalid_key_errors << BenchmarkKey::InvalidKeyError.new(
515           BenchmarkKey::InvalidKeyError::DATASET_MISMATCH, {
516             source_key: {
517               id: id,
518               created_at: created_at,
519               dataset: batch_request.uris
520             },
521             violating_key: {
522               id: key.id,
523               created_at: key.created_at,
524               dataset: key.batch_request.uris
525             },
526             dataset_symmetric_difference: symm_diff_uris.to_a,
527             message: "Source key (id=#{id}) and validation key (id=#{key.id}) have\n"
528               "different \"\n"
529               "benchmark dataset URIS. The symmetric difference is: #{symm_diff_uris.\n"
530               "to_a}.\n"
531           }
532         )

```

```

531     end
532
533     # 3. Ensure successful request made in BOTH instances
534     our_key_success = success?
535     their_key_success = key.success?
536     if our_key_success && their_key_success
537         ICSVSB.ldebug('Both keys were successful')
538     else
539         ICSVSB.lwarn('Sucesss mismatch in key validation')
540         invalid_key_errors << BenchmarkKey::InvalidKeyError.new(
541             BenchmarkKey::InvalidKeyError::SUCCESS_MISMATCH, {
542                 source_key: {
543                     id: id,
544                     created_at: created_at,
545                     successful_response: our_key_success
546                 },
547                 violating_key: {
548                     id: key.id,
549                     created_at: key.created_at,
550                     successful_response: their_key_success
551                 },
552                 message: "Source key (id=#{id}) success=#{our_key_success} but \"\
553                 \"validation key (id=#{key.id}) success=#{their_key_success}."
554             }
555         )
556     end
557
558     # 4. Ensure the same max labels
559     if key.max_labels == max_labels
560         ICSVSB.ldebug('Both keys have same max labels')
561     else
562         ICSVSB.lwarn('Max labels mismatch in key validation')
563         invalid_key_errors << BenchmarkKey::InvalidKeyError.new(
564             BenchmarkKey::InvalidKeyError::MAX_LABELS_MISMATCH, {
565                 source_key: {
566                     id: id,
567                     created_at: created_at,
568                     max_labels: max_labels
569                 },
570                 violating_key: {
571                     id: key.id,
572                     created_at: key.created_at,
573                     max_labels: key.max_labels
574                 },
575                 message: "Source key (id=#{id}) max_labels=#{max_labels} but \"\
576                 \"validation key (id=#{key.id}) max_labels=#{key.max_labels}."
577             }
578         )
579     end
580
581     # 5. Ensure the same min confs
582     if key.min_confidence == min_confidence
583         ICSVSB.ldebug('Both keys have same min confidence')
584     else
585         ICSVSB.lwarn('Minimum confidence or max labels mismatch in key validation')
586         invalid_key_errors << BenchmarkKey::InvalidKeyError.new(
587             BenchmarkKey::InvalidKeyError::MIN_CONFIDENCE_MISMATCH, {
588                 source_key: {
589                     id: id,
590                     created_at: created_at,
591                     min_confidence: min_confidence
592                 },
593                 violating_key: {
594                     id: key.id,

```

```

595         created_at: key.created_at,
596         min_confidence: key.min_confidence
597     },
598     message: "Source key (id=#{id}) min_confidence=#{min_confidence} but \"\
599     validation key (id=#{key.id}) min_confidence=#{key.min_confidence}.”
600   }
601 )
602 end
603
604 # 6. Ensure same number of results... (responses... not labels!)
605 our_response_length = batch_request.responses.length
606 their_response_length = key.batch_request.responses.length
607 if our_response_length == their_response_length
608   ICVSB.ldebug('Both keys have same number of encoded responses')
609 else
610   ICVSB.lwarn('Number of responses mismatch in key validation')
611   invalid_key_errors << BenchmarkKey::InvalidKeyError.new(
612     BenchmarkKey::InvalidKeyError::RESPONSE_LENGTH_MISMATCH, {
613       source_key: {
614         id: id,
615         created_at: created_at,
616         num_responses: our_response_length
617       },
618       violating_key: {
619         id: key.id,
620         created_at: key.created_at,
621         num_responses: their_response_length
622       },
623       message: "Source key (id=#{id}) responses=#{our_response_length} but "
624         ↪ \
625       "validation key (id=#{key.id}) responses=#{their_response_length}.”
626     }
627   )
628 end
629
630 # 7. Validate every label delta and confidence delta
631 our_requests = batch_request.requests
632 their_requests = key.batch_request.requests
633 our_requests.each do |our_request|
634   this_uri = our_request.uri
635   their_request = their_requests.find { |r| r.uri == this_uri }
636
637   our_labels = Set[*our_request.response.labels.keys]
638   their_labels = Set[*their_request.response.labels.keys]
639
640   # 7a. Label delta
641   symmm_diff_labels = our_labels ^ their_labels
642
643   msg_suffix = "URI = #{this_uri} from #{their_request.created_at} (req_id
644     ↪ =#{their_request.id})"\
645   " to #{our_request.created_at} (req_id=#{our_request.id})"
646
647   ICVSB.ldebug("Request id=#{our_request.id} #{our_labels.to_a} against "\
648     "id=#{their_request.id} #{their_labels.to_a} - symmm diff "\
649     "= #{symmm_diff_labels.to_a}")
650   if symmm_diff_labels.length > delta_labels
651     ICVSB.lwarn("Number of labels mismatch in key validation (margin of error
652       ↪ =#{delta_labels}): "\
653       "New/dropped labels = '#{(our_labels - their_labels).to_a.map { |l| "+#
654         ↪ {l}" }.join(',')}'"
655       "#{(their_labels - our_labels).to_a.map { |l| "-#{l}" }.join(',')}")
656   invalid_key_errors << BenchmarkKey::InvalidKeyError.new(
657     BenchmarkKey::InvalidKeyError::LABEL_DELTA_MISMATCH, {
658       source_key: {

```

```

655         id: id,
656         created_at: created_at
657     },
658     source_response: {
659         id: our_request.id,
660         created_at: our_request.created_at,
661         body: our_request.response.hash
662     },
663     violating_key: {
664         id: key.id,
665         created_at: key.created_at
666     },
667     violating_response: {
668         id: their_request.id,
669         created_at: their_request.created_at,
670         body: their_request.response.hash
671     },
672     uri: this_uri,
673     delta_labels_threshold: delta_labels,
674     delta_labels_detected: symm_diff_labels.length,
675     new_labels: (our_labels - their_labels).to_a,
676     dropped_labels: (their_labels - our_labels).to_a,
677     message: "Source key (id=#{id}) and validation key (id=#{key.id})\n"
678     ↪ have #{symm_diff_labels.length} "\n"
679     "differing labels, which exceeds the delta label value of #{
680     ↪ delta_labels}. "\n"
681     "New/dropped labels = '#{(our_labels - their_labels).to_a.map { |l| "
682     ↪ "#[l]" }.join(',')}'\n"
683     "#{(their_labels - our_labels).to_a.map { |l| "-#[l]" }.join(',')}"\n"
684     ". #{msg_suffix}.\n"
685   }
686   )
687 else
688   ICSVB.ldebug("Number of labels match both keys (within margin of error #{
689   ↪ delta_labels})")
690 end
691
692 # 7b. Confidence delta
693 delta_confs_exceeded = {}
694 our_request.response.labels.each do |label, conf|
695   our_conf = conf
696   their_conf = their_request.response.labels[label]
697
698   if their_conf.nil?
699     ICSVB.ldebug("The label #{label} does not exist in the response id=#{
700     ↪ their_request.response.id}. "\n"
701     "Skipping confidence comparison...")
702   next
703 end
704
705 delta = our_conf - their_conf
706 ICSVB.ldebug("Request id=#{our_request.id} against id=#{their_request.id}\n"
707   ↪ "\n"
708   "for label '#{label}' confidence: #{our_conf}, #{their_conf} (delta=#{
709     ↪ delta})")
710 if delta > delta_confidence
711   ICSVB.lwarn(
712     "Maximum confidence delta breached in key validation (margin of error\n"
713     ↪ =#{delta_confidence}). "\n"
714     "#{msg_suffix}.\n"
715   )
716   delta_confs_exceeded[label] = delta
717 end
718
719 end
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2
```

```

711     if delta_confs_exceeded.empty?
712       ICSVB.ldebug("Both keys have confidence within margin of error #{
713         ↪ delta_confidence}")
714     else
715       invalid_key_errors << BenchmarkKey::InvalidKeyError.new(
716         BenchmarkKey::InvalidKeyError::CONFIDENCE_DELTA_MISMATCH, {
717           source_key: {
718             id: id,
719             created_at: created_at
720           },
721           source_response: {
722             id: our_request.id,
723             created_at: our_request.created_at,
724             body: our_request.response.hash
725           },
726           violating_key: {
727             id: key.id,
728             created_at: key.created_at
729           },
730           violating_response: {
731             id: their_request.id,
732             created_at: their_request.created_at,
733             body: their_request.response.hash
734           },
735           uri: this_uri,
736           delta_confidence_threshold: delta_confidence,
737           delta_confidences_detected: delta_confs_exceeded,
738           message: "Source key (id=#{id}) has exceeded confidence delta of \"\n
739             validation key (id=#{key.id}): #{delta_confs_exceeded}. #{
740               ↪ msg_suffix}.\n
741           "
742         }
743       )
744     end
745   end
746
747   # Check if the responses are valid against this key
748   valid_response, invalid_reasons = valid_against?(our_request.response)
749   if valid_response
750     ICSVB.ldebug('Our response is valid against this key')
751   else
752     invalid_key_errors += invalid_reasons
753   end
754
755   [invalid_key_errors.empty?, invalid_key_errors.sort_by(&:errorcode)]
756 end
757
758 # Validates a response against this key as per rules encoded within this key.
759 # @param [Response] key The response to validate.
760 # @return See #valid_against?
761 def _validate_against_response(response)
762   invalid_key_errors = []
763
764   missing_expected_labels = expected_labels_set - Set[*response.labels.keys]
765   unless missing_expected_labels.empty?
766     invalid_key_errors << BenchmarkKey::InvalidKeyError.new(
767       BenchmarkKey::InvalidKeyError::EXPECTED_LABELS_MISMATCH, {
768         source_key: {
769           id: id,
770           created_at: created_at
771         },
772         violating_response: {
773           id: response.id,
774           created_at: response.created_at,
775           body: response.hash
776         }
777       }
778     )
779   end
780
781   response
782 end

```

```

773     },
774     uri: response.request.uri,
775     expected_labels: expected_labels.split(','),
776     labels_detected: response.labels.keys,
777     labels_missing: missing_expected_labels.to_a,
778     message: "Expected key (id=#{id}) expects the following mandatory
779       ↪ labels: '#{expected_labels}'. \"\
790 \"However, response (id=#{response.id}) has the following labels: '#{
791       ↪ response.labels.keys.join(',')}'. \"\
792 \"The following labels are missing: '#{missing_expected_labels.to_a.join
793       ↪ (',')}'."
794   }
795 end
796
797 [invalid_key_errors.empty?, invalid_key_errors]
798 end
799 end
800
801 # The Request Client class is used to make non-benchmarked requests to the
802 # provided service's labelling endpoints. It handles creating respective
803 # +Request+ and +Response+ records to be committed to the benchmarker database.
804 # Requests made with the +RequestClient+ do *not* ensure that evolution risk
805 # has occurred (see BenchmarkRequestClient).
806 class RequestClient
807   # Initialises a new instance of the requester to label endpoints.
808   # @param [Service] service The service to request from.
809   # @param [Fixnum] max_labels The maximum labels that the requester returns.
810   # Only supported if the service supports this parameter. Default is 100
811   # labels.
812   # @param [Float] min_confidence The confidence threshold by which labels
813   # are returned. Only supported if the service supports this parameter.
814   # Default is 0.50.
815   def initialize(service, max_labels: 100, min_confidence: 0.50)
816     unless service.is_a?(Service) && [Service::GOOGLE, Service::AMAZON, Service
817       ↪ ::AZURE, Service::DEMO].include?(service)
818       raise ArgumentError, "Service with name #{service.name} not supported."
819     end
820
821   # Registers logging for this client
822   ICVSB.register_request_client(self)
823   @logstrio = StringIO.new
824   @log = Logger.new(@logstrio)
825
826   @service = service
827   @service_client =
828     case @service
829     when Service::GOOGLE
830       Google::Cloud::Vision::ImageAnnotator.new
831     when Service::AMAZON
832       Aws::Rekognition::Client.new
833     when Service::AZURE
834       URI('https://australiaeast.api.cognitive.microsoft.com/vision/v2.0/tag')
835     when Service::DEMO
836       nil # Not client needed for mock...
837     end
838
839   @config = {
840     max_labels: max_labels,
841     min_confidence: min_confidence
842   }
843   @max_labels = max_labels
844   @min_confidence = min_confidence
845 end
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2
```

```

833     attr_reader :max_labels, :min_confidence
834
835     # Sends a request to the client's respective service endpoint. Does *not*
836     # validate a response against a key (see BenchmarkedRequestClient).
837     # Params:
838     # @param [String] uri A URI to an image to detect labels.
839     # @param [BatchRequest] batch The batch that the request is being made
840     # under. Defaults to nil.
841     # @return [Response] The response record committed to the benchmark
842     # database.
843   def send_uri(uri, batch: nil)
844     raise ArgumentError, 'URI must be a string.' unless uri.is_a?(String)
845     raise ArgumentError, 'Batch must be a BatchRequest.' if !batch.nil? && !
846       ↪ batch.is_a?(BatchRequest)
847
848     batch_id = batch.nil? ? nil : batch.id
849     ICVSB.ldebug("Sending URI #{uri} to #{@service.name} - batch_id: #{batch_id}
850       ↪ ")
851
852     begin
853       request_start = DateTime.now
854       exception = nil
855       case @service
856       when Service::GOOGLE
857         response = _request_google_cloud_vision(uri)
858       when Service::AMAZON
859         response = _request_amazon_rekognition(uri)
860       when Service::AZURE
861         response = _request_azure_computer_vision(uri)
862       when Service::DEMO
863         response = _request_demo_service(uri)
864       end
865       ICVSB.ldebug("Successful response for URI #{uri} to #{@service.name} (
866         ↪ batch_id=#{batch_id})")
867     rescue StandardError => e
868       ICVSB.lwarn("Exception caught in send_uri: #{e.class} - #{e.message}")
869       exception = e
870     end
871     request = Request.create(
872       service_id: @service.id,
873       created_at: request_start,
874       uri: uri,
875       batch_request_id: batch_id
876     )
877     response = Response.create(
878       created_at: DateTime.now,
879       body: response[:body],
880       success: exception.nil? && response[:success],
881       request_id: request.id
882     )
883     ICVSB.ldebug("Request saved (id=#{request.id}) with response (id=#{response.
884       ↪ id})")
885   end
886
887   # Sends a batch request with multiple images to client's respective service
888   # endpoint. Does *not* validate a response against a key (see
889   # ICVSB::BenchmarkedRequestClient).
890   # @param [Array<String>] uris An array of URIs to an image to detect labels.
891   # @return [BatchRequest] The batch request that was created.
892   def send_uris(uris)
893     raise ArgumentError, 'URIs must be an array of strings.' unless uris.is_a?(
894       ↪ Array)
895

```

```

892     batch_request = BatchRequest.create(created_at: DateTime.now)
893     ICVSB.linfo("Initiated a batch request for #{uris.count} URIs")
894     uris.each do |uri|
895       send_uri(uri, batch: batch_request)
896     end
897     ICVSB.linfo("Batch is complete (id=#{batch_request.id})")
898     batch_request
899   end
900
901   # Performs the same operation as send_uris but performs sends each URI
902   # asynchronously. Saves a lot of time if you have lots of URIs. This method
903   # should not be used with an SQLite database.
904   # @see #send_uris
905   # @param [Array<String>] uri See #send_uris
906   # @return [Array<BatchRequest, Array<Thread>]] Returns both the array and an
907   # array of threads representing each request. Call +threads.join(&:each)+  

908   # to ensure all requests have finished.
909   def send_uris_async(uris)
910     raise ArgumentError, 'URIs must be an array of strings.' unless uris.is_a?(
911       → Array)
912     if ICVSB::Request.superclass.db.url.start_with?('sqlite')
913       raise StandardError, 'You are using SQLite and thus async operations are
914       → not supported.'
915     end
916
917     threads = []
918     batch_request = BatchRequest.create(created_at: DateTime.now)
919     ICVSB.linfo("Initiated an async batch request for #{uris.count} URIs")
920     uris.each do |uri|
921       threads << Thread.new do
922         send_uri(uri, batch: batch_request)
923       end
924     end
925     ICVSB.linfo("Async batch submitted (id=#{batch_request.id}). Wait for this
926       → batch to be complete!")
927     [batch_request, threads]
928   end
929
930   # Adds a message of a specific severity to this client's logger.
931   # @param [Logger::Severity] severity The type of severity to log.
932   # @param [String] message The message to log.
933   def log(severity, message)
934     unless [Logger::DEBUG, Logger::INFO, Logger::WARN, Logger::ERROR, Logger::
935       → FATAL, Logger::UNKNOWN]
936       .include?(severity)
937       raise ArgumentError, 'Severity must be a Logger::Severity type'
938     end
939     raise ArgumentError, 'Message must be a string' unless message.is_a?(String)
940
941     @log.add(severity, message)
942   end
943
944   # Gets the log of this client as a string.
945   # @return [String] The entire log.
946   def read_log
947     @logstrio.string
948   end
949
950   private
951
952   # Makes a request to Google Cloud Vision's +LABEL_DETECTION+ feature.
953   # @see https://cloud.google.com/vision/docs/labels
954   # @param [String] uri A URI to an image to detect labels. Google Cloud
955   # Vision supports JPEGs, PNGs, GIFs, BMPs, WEBPs, RAWs, ICOs, PDFs and

```

```

952      # TIFFs only.
953      # @return [Hash] A hash containing the response +body+ and whether the
954      # request was +successful+.
955      def _request_google_cloud_vision(uri)
956        begin
957          image = _download_image(
958            uri,
959            %w[
960              image/jpeg
961              image/png
962              image/gif
963              image/webp
964              image/x-dcraw
965              image/vnd.microsoft.icon
966              application/pdf
967              image/tiff
968            ]
969          )
970          exception = nil
971          res = @service_client.label_detection(
972            image: image.open,
973            max_results: @max_labels
974          ).to_h
975          rescue StandardError => e
976            exception = e
977            res = { service_error: "#{exception.class} - #{exception.message}" }
978          end
979        {
980          body: res.to_json,
981          success: exception.nil? && res.key?(:responses)
982        }
983      end
984
985      # Makes a request to Amazon Rekognition's +DetectLabels+ endpoint.
986      # @see https://docs.aws.amazon.com/rekognition/latest/dg/API_DetectLabels.html
987      # @param [String] uri A URI to an image to detect labels. Amazon Rekognition
988      # only supports JPEGs and PNGs.
989      # @return (see #_request_google_cloud_vision)
990      def _request_amazon_rekognition(uri)
991        begin
992          image = _download_image(uri, %w[image/jpeg image/png])
993          exception = nil
994          res = @service_client.detect_labels(
995            image: {
996              bytes: image.read
997            },
998            max_labels: @max_labels,
999            min_confidence: @min_confidence
1000          ).to_h
1001          rescue StandardError => e
1002            exception = e
1003            res = { service_error: "#{e.class} - #{e.message}" }
1004          end
1005        {
1006          body: res.to_json,
1007          success: exception.nil? && res.key?(:labels)
1008        }
1009      end
1010
1011      # Makes a request to Azure's +analyze+ endpoint with +visualFeatures+ of
1012      # +Tags+.
1013      # @see https://docs.microsoft.com/en-us/rest/api/cognitiveservices/
1014      #      computervision/tagimage/tagimage
1015      # @param [String] uri A URI to an image to detect labels. Azure Computer

```

```

1015  # Vision only supports JPEGs, PNGs, GIFs, and BMPs.
1016  # @return (see #_request_google_cloud_vision)
1017  def _request_azure_computer_vision(uri)
1018      image = _download_image(uri, %w[image/jpeg image/png image/gif image/bmp])
1019
1020      http_req = Net::HTTP::Post::Multipart.new(
1021          @service_client,
1022          file: UploadIO.new(image.open, image.content_type, image.original_filename
1023                           ↩ )
1024          )
1025          http_req['Ocp-Apim-Subscription-Key'] = ENV['AZURE_SUBSCRIPTION_KEY']
1026
1027          http_res = Net::HTTP.start(@service_client.host, @service_client.port,
1028                           ↩ use_ssl: true) do |h|
1029              h.request(http_req)
1030          end
1031
1032          tags_present = JSON.parse(http_res.body).key?('tags')
1033          {
1034              body: tags_present ? http_res.body : { service_error: http_res.body },
1035              success: tags_present
1036          }
1037
1038          # Makes a request to the mock demo server, returning JSON data at time 1
1039          # (t1) or time 2 (t2), depending on the timestamp flip (which can be
1040          # triggered by the PATCH /benchmark/:key endpoint).
1041          # @param [String] uri A URI to an image to detect labels.
1042          # @return (see #_request_google_cloud_vision)
1043  def _request_demo_service(uri)
1044      # Get the image id from the URI...
1045      regexp = %r{http://localhost:4567/demo/data/(\d{4,12}).jpe?g}
1046
1047      all_image_ids = JSON.parse(
1048          File.read(File.join('demo', 'categories.json'))
1049          )['all']
1050
1051      invalid_uri = (uri =~ regexp).nil?
1052      image_id = uri.match(regexp)[1] unless invalid_uri
1053      invalid_image_id = !all_image_ids.include?(image_id)
1054
1055      # Mock service can be switched to t1 or t2 at demo endpoint...
1056      body =
1057          if invalid_uri || invalid_image_id
1058              { service_error: 'The URI is not a valid demo URI.' }
1059          else
1060              body = JSON.parse(File.read(File.join('demo', "#{$image_id}.#{demotimestamp}.json")))
1061              { responses: [body] }#[{ label_annotations: body }]
1062          end
1063
1064          {
1065              body: body.to_json,
1066              success: !(invalid_uri || invalid_image_id)
1067          }
1068
1069          # Downloads the image at the specified URI.
1070          # @param [String] uri The URI to download.
1071          # @param [Array<String>] mimes Accepted mime types.
1072          # @return [File] if download was successful.
1073  def _download_image(uri, mimes)
1074      raise ArgumentError, 'URI must be a string.' unless uri.is_a?(String)
1075      raise ArgumentError, 'Mimes must be an array of strings.' unless mimes.is_a

```

```

1076      ↢ ?(Array)
1077      raise ArgumentError, "Invalid URI specified: #{uri}." unless uri =~ URI::
1078      ↢ DEFAULT_PARSER.make_regexp
1079
1080      ICSVB.ldebug("Downloading image at URI: #{uri}")
1081      file = Down.download(uri)
1082      mime = file.content_type
1083
1084      unless mimes.include?(mime)
1085        raise ArgumentError, "Content type of URI #{uri} not accepted. Received #{
1086          ↢ mime}. Valid are: #{mimes}."
1087      end
1088
1089      file
1090    rescue Down::Error => e
1091      raise ArgumentError, "Could not access the URI #{uri} - #{e.class}"
1092    end
1093
1094    # The Benchmarked Request Client class is used to make requests to a service's
1095    # labelling endpoints, ensuring that the response from the endpoint has not
1096    # altered significantly as indicated by the expiration flags. It handles
1097    # creating respective +Request+ and +Response+ records to be committed to the
1098    # benchmarker database. Unlike the +RequestClient+, the
1099    # +BenchmarkedRequestClient+ ensures that, respective to a benchmark dataset,
1100    # evolution has not occurred and thus is safe to use the endpoint without
1101    # re-evaluation. Requires a BenchmarkKey to make any requests.
1102    class BenchmarkedRequestClient < RequestClient
1103      alias send_uri_no_key send_uri
1104      alias send_uris_no_key send_uris
1105      alias send_uris_no_key_async send_uris_async
1106
1107      # Initialises a new instance of the benchmarked requester to label
1108      # endpoints.
1109      # @param [Service] service (see RequestClient#initialize)
1110      # @param [Array<String>] dataset An array of URIs to benchmark
1111      # against.
1112      # @param [Fixnum] max_labels (see RequestClient#initialize)
1113      # @param [Float] min_confidence (see RequestClient#initialize)
1114      # @param [Hash] opts Additional benchmark-related parameters.
1115      # @option opts [String] :trigger_on_schedule A cron-tab string (see
1116      # +man 5 crontab+) that is used for the benchmarker to re-evaluate if the
1117      # current key should be expired. Default is every Sunday at midnight,
1118      # i.e., +0 0 * * 0+.
1119      # @option opts [String] :trigger_on_failcount Number of times the benchmark
1120      # request fails making requests for the benchmark to re-evaluate. Must
1121      # be a positive, non-zero number for the benchmark to trigger on failure,
1122      # else this field is ignored. Default is 0.
1123      # @option opts [BenchmarkSeverity] :severity The severity of warning for
1124      # the #BenchmarkKey to fail. Default is +BenchmarkSeverity::INFO+.
1125      # @option opts [String] :benchmark_callback_uri The URI to call with results
1126      # of a completed benchmark. Optional. If an invalid URI is specified this
1127      # will default to nil.
1128      # @option opts [String] :warning_callback_uri Required when the +:severity:+
1129      # is +BenchmarkSeverity::WARN+. If left blank, the effect of the benchmark
1130      # client is essentially a severity of +BenchmarkSeverity::NONE+, as no
1131      # warning endpoint can be called to notify of issues. If an invalid URI is
1132      # provided, this will default to nil.
1133      # @option opts [Boolean] :autobenchmark Automatically benchmark the client
1134      # as soon as it is initialised. If +false+, then you will need to call
1135      # the #benchmark method immediately (i.e., on your own thread). Defaults
1136      # to true, so will block the current thread before benchmarking is
1137      # complete.
1138      # @option opts [Fixnum] :delta_labels Number of labels that change for a

```

```

1137 # #BenchmarkKey to expire. Default is 5.
1138 # @option opts [Float] :delta_confidences Minimum amount of difference for
1139 # the same label to have changed between the last benchmark for the
1140 # #BenchmarkKey to expire. Default is 0.01.
1141 # @option opts [Array<String>] :expected_labels Array of strings for the
1142 # various expected labels that should be expected in every result. Fails
1143 # otherwise. Encoded within the key.
1144 def initialize(service, dataset, max_labels: 100, min_confidence: 0.50, opts:
1145   ↪ {})
1146   super(service, max_labels: max_labels, min_confidence: min_confidence)
1147   @dataset = dataset
1148   @key_config = {
1149     delta_labels: opts[:delta_labels] || 5,
1150     delta_confidence: opts[:delta_confidence] || 0.01,
1151     severity: opts[:severity] || BenchmarkSeverity::INFO,
1152     expected_labels: opts[:expected_labels] || []
1153   }
1154   @benchmark_config = {
1155     trigger_on_schedule: opts[:trigger_on_schedule] || '0 0 * * 0',
1156     trigger_on_failcount: opts[:trigger_on_failcount] || 0,
1157     autobenchmark: opts[:autobenchmark].nil? ? true : opts[:autobenchmark]
1158   }
1159   # Validate URIs
1160   if !opts[:benchmark_callback_uri].nil? &&
1161     !(opts[:benchmark_callback_uri] =~ URI::DEFAULT_PARSER.make_regexp).nil?
1162     @benchmark_config[:benchmark_callback_uri] = URI(opts[:benchmark_callback_uri])
1163   end
1164   if !opts[:warning_callback_uri].nil? &&
1165     !(opts[:warning_callback_uri] =~ URI::DEFAULT_PARSER.make_regexp).nil?
1166     @benchmark_config[:warning_callback_uri] = URI(opts[:warning_callback_uri])
1167   end
1168   if !opts[:warning_callback_uri].nil? && opts[:severity] != BenchmarkSeverity
1169     ICVSB.lwarn("A warning callback URI #{opts[:warning_callback_uri]} was set
1170     ↪ but \"\n      'the severity is not WARNING. This callback will be ignored...'")
1171   end
1172
1173   @created_at = DateTime.now
1174   @demo_timestamp = 't1' if @service == Service::DEMO
1175   @is_benchmarking = false
1176   @last_benchmark_time = nil
1177   @benchmark_count = 0
1178   @invalid_state_count = 0
1179   trigger_benchmark if @benchmark_config[:autobenchmark]
1180   @scheduler = Rufus::Scheduler.new.schedule(@benchmark_config[:trigger_on_schedule]) do |cronjob|
1181     ICVSB.linfo("Cronjob starting for BenchmarkedRequestClient #{self} - \"\
1182       Scheduled at: #{cronjob.scheduled_at}; Last ran at: #{cronjob.last_time
1183       ↪ }.\")"
1184   trigger_benchmark
1185 end
1186
1187 # Exposes whether or not the client is currently benchmarking.
1188 # @return [Boolean] True if the client is benchmarking, false otherwise.
1189 def benchmarking?
1190   @is_benchmarking
1191 end
1192
1193 # Returns the next time a schedule to trigger a benchmark will run.

```

```

1194  # @return [DateTime] The time the next trigger to benchmark will be run.
1195  def next_scheduled_benchmark_time
1196    DateTime.parse(@scheduler.next_time.to_t.to_s)
1197  end
1198
1199  # Returns the last time a schedule to trigger a benchmark was run.
1200  # @return [DateTime,nil] Time next DateTime the benchmark ran or nil if
1201  # the scheduler has never yet run.
1202  def last_scheduled_benchmark_time
1203    @scheduler.last_time.nil? ? nil : DateTime.parse(@scheduler.last_time.to_t.
1204      ↪ to_s)
1205  end
1206
1207  # Returns the average time taken to complete the last benchmark.
1208  # @return [Float] The time taken.
1209  def mean_scheduled_benchmark_duration
1210    @scheduler.mean_work_time
1211  end
1212
1213  # Returns the time taken to complete the last benchmark.
1214  # @return [Float] The time taken.
1215  def last_scheduled_benchmark_duration
1216    @scheduler.last_work_time
1217  end
1218
1219  attr_reader *%i[
1220    invalid_state_count
1221    current_key
1222    created_at
1223    dataset
1224    benchmark_count
1225    last_benchmark_time
1226    benchmark_config
1227    key_config
1228    service
1229  ]
1230
1231  attr_accessor :demo_timestamp
1232
1233  # Sends an image to this client's respective labelling endpoint, verifying
1234  # the key provided has not expired (and thus substantial evolution in the
1235  # labelling endpoint has not occurred for significant impact to the results).
1236  # Depending on the key's varied severity level, a response will be returned
1237  # with varied fields populated.
1238  # @param [URI] uri (see RequestClient#send_uri)
1239  # @param [BenchmarkKey] key The benchmark key required to make a request
1240  # to the service using this client. This key is verified against this
1241  # client's most recent benchmark, thereby ensuring no evolution has occurred
1242  # in the back-end service.
1243  # @return [Hash] A hash with the following keys: +:response+, the raw
1244  # #Response object returned from the #RequestClient.send_uri method (i.e.,
1245  # a non-benchmarked response) or +nil+ if the #key has expired or invalid
1246  # and the key's severity level is #BenchmarkSeverity::EXCEPTION;
1247  # +:labels+, a shortcut to the #Response.label method of the response or
1248  # +nil+ if the key has expired or was invalid and the key's severity level
1249  # is #BenchmarkSeverity::EXCEPTION; +:key_errors:+ a(n) error(s) response
1250  # indicating if the key has expired (a string value) which is only
1251  # populated if the key has a severity level of
1252  # #BenchmarkSeverity::EXCEPTION or #BenchmarkSeverity::WARNING;
1253  # +:response_errors:+ similar to :key_errors: but for the response;
1254  # +:cached:+ an optional DateTime indicating that there was no need to make
1255  # a request to the service as the benchmarker holds a cached response that
1256  # is still valid; this indicates the time at which the cached response was
# generated.

```

```

1257     def send_uri_with_key(uri, key)
1258       raise ArgumentError, 'URI must be a string.' unless uri.is_a?(String)
1259       raise ArgumentError, 'Key must be a BenchmarkKey.' unless key.is_a?(
1260         BenchmarkKey)
1261 
1262       if @current_key.nil?
1263         return {
1264           key_errors: [
1265             BenchmarkKey::InvalidKeyError.new(BenchmarkKey::InvalidKeyError::
1266               NO_KEY_YET)
1267           ]
1268         }
1269 
1270       result = {
1271         labels: nil,
1272         response: nil,
1273         key_errors: nil,
1274         response_errors: nil,
1275         service_error: nil,
1276         cached: nil
1277       }
1278 
1279       # Check for a cached result w/ this service given provided key...
1280       ICVSB.ldebug("Attempting to use a cached response for #{uri} + #{@service.
1281         name}...")
1282       Request.where(uri: uri, service_id: @service.id, benchmark_key_id: key.id)
1283         .order(Sequel.desc(:created_at)).each do |request|
1284         response = request.response
1285 
1286         # Ignore unsuccessful responses
1287         next if response.nil? || !response.success?
1288 
1289         # Check if the response's benchmark is still valid -- if so, just
1290         # reuse that result... (no need to actually ping service)
1291         key_is_valid, = @current_key.valid_against?(response.benchmark_key)
1292         ICVSB.ldebug("Cached key (id=#{response.benchmark_key.id}) is valid
1293           ↪ against current key \"\n
1294             "(id=#{@current_key.id})? #{key_is_valid}\")")
1295         if !response.benchmark_key.nil? && key_is_valid
1296           return { labels: response.labels, response: response.hash, cached:
1297             DateTime.parse(response.created_at.to_s) }
1298         end
1299       end
1300       ICVSB.ldebug("Cached response failed! Will try to invoke a request to #{@
1301         service.name}")
1302 
1303       # Check for key validity
1304       ICVSB.ldebug("Checking if current key (id=#{@current_key.id}) is valid
1305           ↪ against key provided (id=#{key.id})...")
1306       key_valid, key_invalid_reasons = @current_key.valid_against?(key)
1307       # Invalid state count incrementemnt if key error exists...
1308       unless key_valid
1309         ICVSB.ldebug("Validation of current key (id=#{@current_key.id}) failed
1310           ↪ against key provided (id=#{key.id}). "
1311             "Reasons: #[key_invalid_reasons.join('; ')]")
1312         result[:key_errors] = key_invalid_reasons
1313         @invalid_state_count += 1
1314         ICVSB.linfo("Error has occured in key validation. Invalid state count
1315           ↪ count is now #{@invalid_state_count}.")
1316       end
1317 
1318       # If key is valid, raise request and check if response is valid
1319       ICVSB.ldebug("Key provided #[key.id] is valid against current key #{
1320

```

```

1312           ↪ @current_key.id}!")
1313     if key_valid
1314       ICVSB.ldebug("Invoking a request '#{uri}' to #{@service.name}...")
1315       response = send_uri_no_key(uri)
1316       ICVSB.ldebug("Response returned (id=#{response.id})! Labels: #{response.
1317                     ↪ labels}")
1318       # Update the benchmark key id
1319       response.benchmark_key_id = @current_key.id
1320       ICVSB.ldebug("Updated response (id=#{response.id}) with benchmark key = #{
1321                     ↪ response.benchmark_key_id}...")
1322       # Now check to see if it was valid given that the response was successful
1323       if response.success?
1324         ICVSB.ldebug("Checking if this response (id=#{response.id}) is valid
1325                     ↪ against current key (id=#{key.id})")
1326         response_valid, response_invalid_reasons = @current_key.valid_against?(
1327                     ↪ response)
1328       end
1329       result[:labels] = response.labels
1330       result[:response] = response.hash
1331       result[:service_error] = result[:response][:service_error].to_s unless
1332         ↪ result[:response][:service_error].nil?
1333       response_valid ||= !result[:response][:service_error].nil?
1334       # Increment invalid state count if response error ONLY (i.e., not service
1335                     ↪ error)
1336       unless response_valid
1337         ICVSB.ldebug("Validation of current key (id=#{@current_key.id}) failed
1338                     ↪ against response \"\
1339                     \"(id=#{response.id}). Reasons: #{response_invalid_reasons.join('; ')}\"")
1340         result[:response_errors] = response_invalid_reasons
1341         @invalid_state_count += 1
1342         ICVSB.linfo('Error has occurred in response validation. \
1343                     \"Invalid state count count is now #{@invalid_state_count}.")"
1344       end
1345     end
1346   end
1347
1348   # If benchmark trigger on num failures is set
1349   if @benchmark_config[:trigger_on_failcount].positive? &&
1350     @invalid_state_count > @benchmark_config[:trigger_on_failcount]
1351     ICVSB.linfo("Benchmark has failed #{@benchmark_config[:.
1352                     ↪ trigger_on_failcount]} \"\
1353                     'times... retriggering benchmark...'")
1354     @invalid_state_count = 0
1355     trigger_benchmark
1356   end
1357
1358   # Response behaviour is dependent on the severity encoded within the key
1359   case @current_key.benchmark_severity
1360   when BenchmarkSeverity::EXCEPTION
1361     # Only expose errors if they exist
1362     if (result[:key_errors].nil? || result[:key_errors].empty?) &&
1363       result[:response_errors].nil? &&
1364       result[:service_error].nil?
1365     result
1366     else
1367       {
1368         key_errors: result[:key_errors],
1369         response_errors: result[:response_errors],
1370         service_error: result[:service_error]
1371       }
1372     end
1373   when BenchmarkSeverity::WARNING
1374     # Flag a warning to the warning endpoint about this result if sev is WARN
1375     _flag_warning(result)
1376   result
1377 
```

```

1367   when BenchmarkSeverity::INFO
1368     # Log to info...
1369     unless key_valid
1370       ICVSB.lwarn("Benchmarked request made for #{uri} with invalid key \"\
1371         "(id=#{@current_key.id}) -- error reasons: #{key_invalid_reasons.join \
1372           '<-- ('; ')'}")
1373     end
1374     unless response_valid
1375       ICVSB.lwarn("Benchmarked request made for #{uri} and response violated \
1376         <-- current key \"\
1377           "(id=#{@current_key.id}) -- error reasons: #{response_invalid_reasons. \
1378             '<-- join('; ')'}")
1379     end
1380   result
1381 when BenchmarkSeverity::NONE
1382   # Passthrough...
1383   result
1384 end
1385
# Makes a request to benchmark's the client's current key against the
# client's URIs to benchmark against. Expires the existing current key
# if a new benchmark key is no longer valid against the old benchmark key.
1386 # @return [void]
1387 def trigger_benchmark
1388   @is_benchmarking = true
1389   new_key = _benchmark
1390   old_key = @current_key
1391   expiry_occurred = false
1392   if @current_key.nil?
1393     @current_key = new_key
1394   else
1395     # Check if the key is valid
1396     valid_key, invalid_reasons = @current_key.valid_against?(new_key)
1397     unless valid_key
1398       ICVSB.lerror('BenchmarkedRequestClient no longer has a valid key! ' \
1399         "Reason(s) = '#{invalid_reasons.join('; ')}'" \
1400         "Expiring old key (id=#{@current_key.id}) with new key (id=#{new_key.id \
1401           '<-- })")
1402       @current_key.expire
1403       @current_key = new_key
1404       expiry_occurred = true
1405     end
1406   end
1407   # # Check if the responses are valid against the current key
1408   # new_key.batch_request.responses.each do |res|
1409   #   valid_response, invalid_reasons = @current_key.valid_against?(res)
1410   #   unless valid_response
1411   #     ICVSB.lerror('BenchmarkedRequestClient has a violated response! ' \
1412   #       "Reason(s) = '#{invalid_reasons.join('; ')}'. Falling back to old key (id \
1413   #         '<-- =#{old_key.nil? ? '<NONE>' : old_key.id})...")
1414   #     @current_key.expire
1415   #     @current_key = old_key
1416   #     @current_key.&.unexpire
1417   #     expiry_occurred = true
1418   #   end
1419   # end
1420   @is_benchmarking = false
1421   _flag_benchmarking_complete(new_key, old_key, expiry_occurred)
1422 end
1423
# Locates the last behaviour token key from the given date
# @param [DateTime] Date at which the key should be searched from
1424
1425

```

```

1426      # @param [BenchmarkKey] The benchmark key found, or nil.
1427      def find_key_since(date)
1428        candidate_bks = BenchmarkKey.where(
1429          service_id: @service.id,
1430          benchmark_severity_id: @key_config[:severity].id,
1431          max_labels: @max_labels,
1432          min_confidence: @min_confidence,
1433          delta_labels: @key_config[:delta_labels],
1434          delta_confidence: @key_config[:delta_confidence],
1435          expected_labels: @key_config[:expected_labels].map(&:downcase).join(','),
1436        ).where(Sequel[:created_at] > date).reverse_order(:created_at)
1437        return nil if candidate_bks.nil?
1438
1439        candidate_bks.find do |bk|
1440          (Set[*bk.batch_request.uris] ^ Set[@dataset]).empty?
1441        end
1442      end
1443
1444    private
1445
1446    # Forwards a full result to the benchmarked request client's warning endpoint
1447    # @param [Hash] result See #send_uri_with_key
1448    # @return [void]
1449    def _flag_warning(result)
1450      return if @benchmark_config[:warning_callback_uri].nil? || @key_config[:  

1451        ↪ severity] != BenchmarkSeverity::WARNING
1452
1453      uri = @benchmark_config[:warning_callback_uri]
1454      data = result
1455      Thread.new do
1456        ICSVSB.linfo("POSTing to warning endpoint '#{uri}' data=#{data}")
1457        req = Net::HTTP::Post.new(uri)
1458        req.body = data.to_json
1459        req.content_type = 'application/json; charset=utf8'
1460        res = Net::HTTP.start(uri.hostname, uri.port) do |http|
1461          http.request(req)
1462        end
1463        ICSVSB.linfo("Response from warning endpoint: #{res.code} #{res.message}")
1464        ICSVSB.ldebug("Response body is: #{res.body}") if res.is_a?(Net::  

1465          ↪ HTTPSuccess)
1466      end
1467
1468    # Forwards a new key that has been generated due to benchmark trigger and
1469    # sends the current or old key (depending on expiry_occurred flag.)
1470    # @param [BenchmarkKey] new_key The new key that was generated from the
1471    # benchmark that was triggered.
1472    # @param [BenchmarkKey] old_or_current_key The current key, if expiry did
1473    # not occur, or the old key if expiry did occur.
1474    # @param [Boolean] expiry_occurred Indicates if the current_key was expired
1475    # and replaced with the new_key.
1476    # @return [void]
1477    def _flag_benchmarking_complete(new_key, old_or_current_key, expiry_occurred)
1478      return if @benchmark_config[:benchmark_callback_uri].nil?
1479
1480      uri = @benchmark_config[:benchmark_callback_uri]
1481      old_or_current_key_id = old_or_current_key.nil? ? nil : old_or_current_key.  

1482        ↪ id
1483      data = { new_key: new_key.id, old_key: old_or_current_key_id, expiry_occurred  

1484        ↪ : expiry_occurred }
1485      Thread.new do
1486        ICSVSB.linfo("POSTing to benchmark complete endpoint '#{uri}' data=#{data}"  

1487          ↪ )
1488        req = Net::HTTP::Post.new(uri)

```

```
1485     req.body = data.to_json
1486     req.content_type = 'application/json; charset=utf8'
1487     res = Net::HTTP.start(uri.hostname, uri.port) do |http|
1488       http.request(req)
1489     end
1490     ICVSB.linfo("Response from benchmark complete endpoint: #{res.code} #{res.
1491                   ↪ message}")
1491     ICVSB.ldebug("Response body is: #{res.body}") if res.is_a?(Net::
1492                   ↪ HTTPSuccess)
1493   end
1494
1495 # Benchmarks this client against a set of URIs, returning this client's
1496 # configured key configuration. Internal method...
1497 # @return [BenchmarkKey] A key representing the result of this benchmark.
1498 def _benchmark
1499   @last_benchmark_time = DateTime.now
1500   @benchmark_count += 1
1501   ICVSB.linfo("Benchmarking dataset against dataset of #{@dataset.count} URIs.
1502               ↪ ")
1502   "Times benchmarked=#{benchmark_count}")
1503   br, thr = send_uris_no_key_async(@dataset)
1504   ICVSB.linfo("Benchmarking this dataset using batch request with id=#{br.id}.
1504               ↪ ")
1505   # Wait for all threads to finish...
1506   thr.each(&:join)
1507   ICVSB.linfo("Batch request with id=#{br.id} is now complete!")
1508   bk = BenchmarkKey.create(
1509     service_id: @service.id,
1510     benchmark_severity_id: @key_config[:severity].id,
1511     batch_request_id: br.id,
1512     created_at: DateTime.now,
1513     expired: false,
1514     delta_labels: @key_config[:delta_labels],
1515     delta_confidence: @key_config[:delta_confidence],
1516     expected_labels: @key_config[:expected_labels].map(&:downcase).join(','),
1517     max_labels: @max_labels,
1518     min_confidence: @min_confidence
1519   )
1520   # Ensure every response is updated with this key
1521   br.responses.each do |res|
1522     ICVSB.ldebug("Updating response id=#{res.id} to benchmark key id=#{bk.id}.
1522               ↪ ")
1523     res.benchmark_key_id = bk.id
1524   end
1525   ICVSB.linfo("Benchmarking dataset is complete (benchmark key id=#{bk.id}).")
1526   bk
1527 end
1528 end
1529 end
```

Listing B.2: Implementation of the architecture facade API.

```

1 # frozen_string_literal: true
2
3 # Author:: Alex Cummaudo (mailto:ca@deakin.edu.au)
4 # Copyright:: Copyright (c) 2019 Alex Cummaudo
5 # License:: MIT License
6
7 require 'sinatra'
8 require 'time'
9 require 'json'
10 require 'cgi'
11 require 'require_all'
12 require_all 'lib'
13
14
15 set :root, File.dirname(__FILE__)
16 set :public_folder, File.join(File.dirname(__FILE__), 'static')
17 set :show_exceptions, false
18 set :demo_folder, File.join(File.dirname(__FILE__), 'demo')
19
20 store = {}
21
22 before do
23   if request.body.size.positive?
24     request.body.rewind
25     @params = JSON.parse(request.body.read, symbolize_names: true)
26   end
27 end
28
29 def halt!(code, message)
30   content_type 'text/plain'
31   halt code, message
32 end
33
34 def check_brc_id(id, store)
35   halt! 400, 'Benchmark id must be a positive integer' unless id.integer? && id.
36   ↪ to_i.positive?
37   halt! 400, "No such benchmark request client exists with id=#{id}" unless store
38   ↪ .key?(id)
39 end
40
41 get '/' do
42   File.read(File.expand_path('index.html', settings.public_folder))
43 end
44
45 # Creates a new benchmark request client with given parameters
46 post '/benchmark' do
47   # Extract params
48   service = params[:service] || ''
49   benchmark_dataset = params[:benchmark_dataset] || ''
50   max_labels = params[:max_labels] || ''
51   min_confidence = params[:min_confidence] || ''
52   trigger_on_schedule = params[:trigger_on_schedule] || ''
53   trigger_on_failcount = params[:trigger_on_failcount] || ''
54   benchmark_callback_uri = params[:benchmark_callback_uri] || ''
55   warning_callback_uri = params[:warning_callback_uri] || ''
56   expected_labels = params[:expected_labels] || ''
57   delta_labels = params[:delta_labels] || ''
58   delta_confidence = params[:delta_confidence] || ''
59   severity = params[:severity] || ''
60
61   # Check param types
62   unless max_labels.integer? && max_labels.to_i.positive?

```

```

61     halt! 400, 'max_labels must be a positive integer'
62   end
63   unless min_confidence.float? && min_confidence.to_f.positive?
64     halt! 400, 'min_confidence must be a positive float'
65   end
66   unless delta_labels.integer? && delta_labels.to_i.positive?
67     halt! 400, 'delta_labels must be a positive integer'
68   end
69   unless delta_confidence.float? && delta_confidence.to_f.positive?
70     halt! 400, 'delta_confidence must be a positive float'
71   end
72   unless ICVSB::VALID_SERVICES.include?(service.to_sym)
73     halt! 400, "service must be one of #{ICVSB::VALID_SERVICES.join(', ', '')}"
74   end
75   unless trigger_on_schedule.cronline?
76     halt! 400, 'trigger_on_schedule must be a cron string in * * * * * (see man 5
77     ↪ crontab)'
78   end
79   unless trigger_on_failcount.integer? && trigger_on_failcount.to_i >= -1
80     halt! 400, 'trigger_on_failcount must be zero or positive integer'
81   end
82   if !benchmark_callback_uri.empty? && !benchmark_callback_uri.uri?
83     halt! 400, 'benchmark_callback_uri is not a valid URI'
84   end
85   unless ICVSB::VALID_SEVERITIES.include?(severity.to_sym)
86     halt! 400, "severity must be one of #{ICVSB::VALID_SEVERITIES.join(', ', '')}"
87   end
88   if ICVSB::BenchmarkSeverity[name: severity.to_s] == ICVSB::BenchmarkSeverity::
89     ↪ WARNING && !warning_callback_uri.uri?
90     halt! 400, 'Must provide a valid warning_callback_uri when severity is WARNING
91     ↪ '
92   end
93   halt! 400, 'benchmark_dataset has not been specified' if benchmark_dataset.
94     ↪ empty?
95   benchmark_dataset = benchmark_dataset.lines.map(&:strip)
96   expected_labels = expected_labels.empty? ? [] : expected_labels.split(',').map
97     ↪ (&:strip)
98   benchmark_dataset.each do |uri|
99     unless uri.uri?
100       halt! 400, "benchmark_dataset must be a list of uris separated by a newline
101         ↪ character; #{uri} is not a valid URI"
102     end
103   end
104
105   # Convert params
106   brc = ICVSB::BenchmarkedRequestClient.new(
107     ICVSB::Service[name: service.to_s],
108     benchmark_dataset,
109     max_labels: max_labels.to_i,
110     min_confidence: min_confidence.to_f,
111     opts: {
112       trigger_on_schedule: trigger_on_schedule,
113       trigger_on_failcount: trigger_on_failcount.to_i,
114       benchmark_callback_uri: benchmark_callback_uri,
115       warning_callback_uri: warning_callback_uri,
116       expected_labels: expected_labels,
117       delta_labels: delta_labels.to_i,
118       delta_confidence: delta_confidence.to_f,
119       severity: ICVSB::BenchmarkSeverity[name: severity.to_s],
120       autobenchmark: false
121     }
122   )

```

```

119  # Benchmark on new thread
120  Thread.new do
121    brc.trigger_benchmark
122    store[brc.object_id] = brc
123  end
124
125  store[brc.object_id] = brc
126
127  status 201
128  content_type 'application/json; charset=utf-8'
129  { id: brc.object_id }.to_json
130 end
131
132 # Gets all auxillary information about the benchmark
133 get '/benchmark/:id' do
134   id = params[:id].to_i
135   check_brc_id(id, store)
136   brc = store[id]
137
138   content_type 'application/json; charset=utf-8'
139   {
140     id: id,
141     service: brc.service.name,
142     created_at: brc.created_at,
143     current_key_id: brc.current_key ? brc.current_key.id : nil,
144     is_benchmarking: brc.benchmarking?,
145     last_scheduled_benchmark_time: brc.last_scheduled_benchmark_time,
146     next_scheduled_benchmark_time: brc.next_scheduled_benchmark_time,
147     mean_scheduled_benchmark_duration: brc.mean_scheduled_benchmark_duration,
148     last_scheduled_benchmark_duration: brc.last_scheduled_benchmark_duration,
149     invalid_state_count: brc.invalid_state_count,
150     last_benchmark_time: brc.last_benchmark_time,
151     benchmark_count: brc.benchmark_count,
152     config: {
153       max_labels: brc.max_labels,
154       min_confidence: brc.min_confidence,
155       key: brc.key_config,
156       benchmarking: brc.benchmark_config
157     },
158     benchmark_dataset: brc.dataset
159   }.to_json
160 end
161
162 patch '/benchmark/:id' do
163   # Set is_benchmarking to true to force the benchmark to reevaluate...
164   # Else, endpoint is ignored
165   id = params['id'].to_i
166   check_brc_id(id, store)
167   brc = store[id]
168
169   status 202
170   response = {
171     id: id,
172     service: brc.service.name,
173     current_key_id: brc.current_key ? brc.current_key.id : nil,
174     is_benchmarking: brc.benchmarking?
175   }
176   if brc.service == ICVSB::Service::DEMO && params[:demo_timestamp]
177     brc.demo_timestamp = params[:demo_timestamp] if ['t1', 't2'].include?(params[:  
      ↪ demo_timestamp])
178     response[:timestamp] = brc.demo_timestamp
179   end
180
181   brc.trigger_benchmark if params[:is_benchmarking] && !brc.benchmarking?

```

```

182
183   response.to_json
184 end
185
186 # Gets all auxillary information about this key's benchmark
187 get '/benchmark/:id/key' do
188   id = params[:id].to_i
189   check_brc_id(id, store)
190   brc = store[id]
191
192   halt! 422, 'The requested benchmark client is still benchmarking its first key'
193   ↪ if brc.current_key.nil?
194
195   current_key_id = brc.current_key.id
196   redirect "/key/#{current_key_id}"
197 end
198
199 get '/key/:id' do
200   id = params[:id].to_i
201   bk = BenchmarkKey[id: params[:id]]
202
203   halt! 400, 'id must be an integer' unless id.integer?
204   halt! 400, "No such benchmark key request client exists with id=#{id}" if bk.
205   ↪ nil?
206
207   content_type 'application/json; charset=utf-8'
208   {
209     id: bk.id,
210     service: bk.service.name,
211     created_at: bk.created_at,
212     benchmark_dataset: bk.batch_request.uris,
213     success: bk.success?,
214     expired: bk.expired?,
215     severity: bk.severity.name,
216     responses: bk.batch_request.responses.map(&:hash),
217     config: {
218       expected_labels: bk.expected_labels_set.to_a,
219       delta_labels: bk.delta_labels,
220       delta_confidence: bk.delta_confidence,
221       max_labels: bk.max_labels,
222       min_confidence: bk.min_confidence
223     }
224   }.to_json
225 end
226
227 # Gets the log of the benchmark with the given id
228 get '/benchmark/:id/log' do
229   id = params[:id].to_i
230
231   check_brc_id(id, store)
232
233   content_type 'text/plain'
234   store[id].read_log
235 end
236
237 post '/callbacks/benchmark' do
238   "Acknowledged benchmark completion with params: '#{params}'..."
239 end
240
241 post '/callbacks/warning' do
242   "Acknowledged benchmark warning params: '#{params}'..."
243 end
244
245 # Labels resources against the provided uri. This is a conditional HTTP request.

```

```

244 # Must provide "If-Match" request header field with at least one ETag. Note that
245 # the ETag must ALWAYS been provided in the following format:
246 #
247 # W/"<benchmark-id>[;<behaviour-token>]"
248 #
249 # Note that the ETag is a weak ETag; ``weak ETag values of two representations
250 # of the same resources might be semantically equivalent, but not byte-for-byte
251 # identical.'' (https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/ETag).
252 # That is, as the developer is not directly accessing the service, they are
253 # only getting a semantically equivalent representation of the labels, but not
254 # a byte-for-byte equivalent (the model may have changed slightly, given the
255 # latest benchmark used.)
256 #
257 # The first id, the benchmark-id, is mandatory as the request must know what
258 # benchmark dataset (and service) the requested URI is being made against.
259 #
260 # The following behaviour-token is optional, indicating the tolerances to which
261 # the response will be made, and the behaviour by which the response will change
262 # given if evolution has occurred since the last benchmark was made. (Not that
263 # internally to this project, we refer to the behaviour token as a BenchmarkKey
264 # -- see ICSVSB::BenchmarkKey.)
265 #
266 # One may provide multiple ETags (separated by commas) in the format:
267 #
268 # W/"<benchmark-id1>[;<behaviour-token1>]",W/"<benchmark-id2>[;<behaviour-token2
269 # <-- >]" ...
270 #
271 # Where this is the case, the label requested will attempt to match ANY of the
272 # tags provided. If failure occurs for the first, it will default to the next
273 # ETag, and so on.
274 #
275 # If NO behaviour-token is specified, then then (additionally) one must provide
276 # an "If-Unmodified-Since" request header field, indicating that the resource
277 # (labels) must have been unmodified since the given date. This will attempt to
278 # automatically locate the nearest behaviour token that was generated after the
279 # given date and request the labels against that date.
280 #
281 # The endpoint will return one of the following HTTP responses:
282 #
283 # - 200 OK if this is the first request made to this URI;
284 # - 400 Bad Request if invalid parameters were provided by the client;
285 # - 412 Precondition Failed if the key/unmodified time provided is no longer
286 # valid, and thus the key provided (or time provided) is violating the
287 # valid tolerances embedded within the key (responding further details
288 # reasoning what tolerances were violated as metadata in the response body);
289 # - 428 Precondition Required if no If-Match field is provided in request;
290 # - 422 Unprocessable Entity if a service error has occurred, indicating the
291 # service cannot process the entity or a bad request was made.
292 # - 500 Internal Server Error if a facade error has occurred.
293 #
294 # The endpoint will return the following HTTP response headers:
295 #
296 # - ETag: The ETag that was used to successfully generate a response
297 # - Last-Modified: The last time the benchmark-id was benchmarked against
298 # its dataset
299 # - Expires: The next time the benchmark with the provided id will be
300 # benchmarked against its dataset
301 # - Age: Indicates that the response provided is cached (i.e., no changes
302 # to the service the last time it was benchmarked against the dataset
303 # to not be considered a violation); returns the time elapsed in seconds
304 # since then
305 get '/labels' do
306   image_uri = CGI.unescape(params[:image])

```

```

307 |     if_match = request.env['HTTP_IF_MATCH'] || ''
308 |     if_unmodified_since = request.env['HTTP_IF_UNMODIFIED_SINCE'] || ''
309 |
310 |     halt! 400, 'URI provided to analyse is not a valid URI' unless image_uri.uri?
311 |     halt! 428, 'Missing If-Match in request header' if if_match.nil?
312 |     if !if_unmodified_since.empty? && !if_unmodified_since.httpdate?
313 |       halt! 400, 'If Unmodified Since must be compliant with the RFC 2616 HTTP date
314 |         ↪ format'
315 |     end
316 |     if_unmodified_since_date = if_unmodified_since.empty? ? nil : Time.httpdate(
317 |       ↪ if_unmodified_since)
318 |
319 |     relay_body = nil
320 |     relay_etag = nil
321 |     relay_last_modified = nil
322 |     relay_expires = nil
323 |
324 |     # Scan through each comma-separated ETag
325 |     etags = if_match.scan(%r{W/"(\d+;?\d+)",?})
326 |     if etags.empty?
327 |       halt! 428, 'Malformed ETags provided. Ensure you are using the correct format.
328 |         ↪ '
329 |     end
330 |     etags.each do |etag|
331 |       etag = etag[0]
332 |       benchmark_id, benchmark_key_id = etag.split(';').map(&:to_i)
333 |
334 |       # Check if we have a valid benchmark id
335 |       check_brc_id(benchmark_id, store)
336 |       brc = store[benchmark_id]
337 |       bk = nil
338 |
339 |       # Check if we have a key; if no key we must have a If-Unmodified-Since.
340 |       if benchmark_key_id.nil? && if_unmodified_since.empty?
341 |         halt! 400, "You have provided a benchmark id (id=#{benchmark_id}) \"\
342 |           without a behaviour token. Please provide a behaviour token \
343 |           'or include the If-Unmodified-Since request header with a RFC \
344 |           '2616-compliant HTTP date string.'"
345 |       elsif !benchmark_key_id.nil?
346 |         # Check if valid key
347 |         if ICSVB::BenchmarkKey.where(id: benchmark_key_id).empty?
348 |           halt! 400, "No such key with id #{benchmark_key_id} exists!"
349 |         end
350 |         unless benchmark_key_id.integer? && benchmark_key_id.positive?
351 |           halt! 400, 'Behaviour token must be a positive integer.'
352 |         end
353 |
354 |         bk = ICSVB::BenchmarkKey[id: benchmark_key_id]
355 |       elsif !if_unmodified_since_date.nil?
356 |         bk = brc.find_key_since(if_unmodified_since_date)
357 |         halt! 412, "No compatible behaviour token found unmodified since #{
358 |           ↪ if_unmodified_since_date}." if bk.nil?
359 |
360 |       # Process...
361 |       result = brc.send_uri_with_key(image_uri, bk)
362 |
363 |       # Set HTTP status+body as appropriate if there is no more ETags or if
364 |       # this was a successful response (i.e., no errors so don't keep trying other
365 |       # ETags...)
366 |       error = result.key?(:key_errors) || result.key?(:response_errors) || result.
367 |         ↪ key?(:service_error)

```

```

366  if [etag] == etags.last || !error
367  if result[:key_errors] || result[:response_errors]
368    status 412
369    content_type 'application/json; charset=utf-8'
370
371    key_error_len = result[:key_errors].nil? ? 0 : result[:key_errors].length
372    res_error_len = result[:response_errors].nil? ? 0 : result[::
373      ↪ response_errors].length
374
375    key_error_data = result[:key_errors].nil? ? [] : result[:key_errors].map
376      ↪ (&:to_h)
377    res_error_data = result[:response_errors].nil? ? [] : result[::
378      ↪ response_errors].map(&:to_h)
379
380    relay_body = {
381      num_key_errors: key_error_len,
382      num_response_errors: res_error_len,
383      key_errors: key_error_data,
384      response_errors: res_error_data
385    }.to_json
386
387  elsif result[:service_error]
388    status 422
389    content_type 'text/plain'
390    relay_body = result[:service_error]
391
392  else
393    content_type 'application/json; charset=utf-8'
394    unless result[:cached].nil?
395      age_sec = ((DateTime.now - result[:cached]) * 24 * 60 * 60).to_i.to_s
396      headers 'Age' => age_sec
397    end
398    status 200
399    relay_body = result[:response].to_json
400  end
401
402  relay_etag = etag
403  relay_last_modified = brc.current_key.nil? ? brc.created_at.httpdate : brc.
404    ↪ current_key.created_at.httpdate
405  relay_expires = brc.next_scheduled_benchmark_time.httpdate
406
407  end
408  headers \
409    'ETag' => "W/\"#{relay_etag}\\"", \
410    'Expires' => relay_expires, \
411    'Last-Modified' => relay_last_modified
412
413  body relay_body
414
415  error do |e|
416    halt! 500, e.message
417  end
418
419  #####
420  # DEMONSTRATION RELATED API
421  #####
422  get '/demo/categories.json' do
423    content_type 'application/json; charset=utf-8'
424    send_file(File.join(settings.demo_folder, 'categories.json'))
425
426  get '/demo/random/:type.jpg' do
427    category_data = JSON.parse(
428      File.read(File.join(settings.demo_folder, 'categories.json'))
429    )
430    ok_categories = category_data.keys
431
432  end

```

```
426 |     category = params[:type]
427 |
428 |     halt! 400, 'No category provided' if category.empty?
429 |     unless ok_categories.include?(category)
430 |       halt! 400, "Unknown category '#{category}'. Accepted category types are: '#{
431 |         ↪ ok_categories.join("", "")}'."
432 |
433 |     id = category_data[category].sample
434 |
435 |     redirect "/demo/data/#{id}.jpg"
436 |   end
437 |
438 |   get '/demo/data/:id.*' do |_|
439 |     image_id = params[:id].split('.').first
440 |     time_id = params[:id].split('.').last
441 |
442 |     unless File.exist?(File.join(settings.demo_folder, image_id + '.jpg'))
443 |       halt! 400, "No such image with id '#{image_id}' exists in the demo database."
444 |     end
445 |     unless %w[jpg jpeg json].include?(ext)
446 |       halt! 400, 'Invalid file extension. Suffix with .jp[e]g or .t1.json or .t2.
447 |         ↪ json.'
448 |     end
449 |     ext = 'jpg' if ext == 'jpeg'
450 |
451 |     if ext == 'jpg'
452 |       content_type 'image/jpeg'
453 |     else
454 |       content_type 'application/json; charset=utf-8'
455 |       halt! 400, 'Missing time id (.t1 or .t2).' if time_id.empty? || !%w[t1 t2].
456 |         ↪ include?(time_id)
457 |       image_id += '.' + time_id
458 |     end
459 |   end
460 |   send_file(File.join(settings.demo_folder, image_id + '.' + ext))
```

APPENDIX C

Supplementary Materials to Chapter 8

C.1 Detailed Overview of Our Proposed Taxonomy

An overview of the 5 dimensions and categories (sub-dimensions) within our proposed taxonomy. ILS = In-Literature Score, calculated as a percentage of the number of papers that make the recommendation of all 21 primary sources. IPS = In-Practice Score, calculated as the average compliance to the SUS. Colour scales indicate relevancy weight within ILS or IPS values for comparative purposes, where red = *lowest* and green = *highest*. GCV, AWS, ACV = Presence of category in Google Cloud Vision, Amazon Rekognition, and Azure Cloud Vision documentation. Presence indicated as *fully present* (●), *partially present* (◐), and *not present* (○).

Key	Description	Primary Sources	ILS	IPS	GCV	AWS	ACV
A1	Quick-start guides to rapidly get started using the API in a specific programming language.	S4, S9, S10	0.14	0.88	●	○	●
A2	Low-level reference manual documenting all API components to review fine-grade detail.	S1, S3, S4, S8, S9, S10, S11, S12, S15, S16, S17	0.52	0.56	●	●	●
A3	Explanations of the API's high-level architecture to better understand intent and context.	S1, S2, S4, S11, S14, S16, S19, S20	0.38	0.70	●	●	●
A4	Source code implementation and code comments (where applicable) to understand the API author's mindset.	S1, S4, S7, S12, S13, S17, S20	0.33	0.47	○	○	○
A5	Code snippets (with comments) of no more than 30 LoC to understand a basic component functionality within the API.	S1, S2, S4, S5, S6, S7, S9, S10, S11, S14, S15, S16, S18, S20, S21	0.71	0.89	●	●	●
A6	Step-by-step tutorials, with screenshots to understand how to build a non-trivial piece of functionality with multiple components of the API.	S1, S2, S4, S5, S7, S9, S10, S15, S16, S18, S20, S21	0.57	0.54	○	●	●
A7	Downloadable source code of production-ready applications that use the API to understand implementation in a large-scale solution.	S1, S2, S5, S9, S15	0.24	0.66	○	○	●
A8	Best-practices of implementation to assist with debugging and efficient use of the API.	S1, S2, S4, S5, S7, S8, S9, S14	0.38	0.68	○	●	○
A9	An exhaustive list of all major components that exist within the API.	S4, S16, S19	0.14	0.69	○	●	●
A10	Minimum system requirements and dependencies to use the API.	S4, S7, S13, S17, S19	0.24	0.71	○	○	○
A11	Instructions to install or begin using the API and details on its release cycle and updating it.	S4, S7, S8, S9, S11, S13, S16, S19	0.38	0.86	○	●	○
A12	Error definitions that describe how to address a specific problem.	S1, S2, S4, S5, S9, S11, S13	0.33	0.84	○	○	○

Continued on next page...

Key	Description	Primary Sources	ILS	IPS	GCV	AWS	ACV
B1	A brief description of the purpose or overview of the API as a low barrier to entry.	S1, S2, S4, S5, S6, S8, S10, S11, S15, S16	0.48	0.80	●	●	●
B2	Descriptions of the types of applications the API can develop.	S2, S4, S9, S11, S15, S18	0.29	0.57	○	○	●
B3	Descriptions of the types of users who should use the API.	S4, S9	0.10	0.44	○	○	○○
B4	Descriptions of the types of users who will use the product the API creates.	S4	0.05	0.42	○	○	○○
B5	Success stories about the API used in production.	S4	0.05	0.49	○	●	●
B6	Documentation to compare similar APIs within the context to this API.	S2, S6, S13, S18	0.19	0.47	○	○	●●
B7	Limitations on what the API can and cannot provide.	S4, S5, S8, S9, S14, S16	0.29	0.94	○	●	●●
C1	Descriptions of the relationship between API components and domain concepts.	S3, S10	0.10	0.51	○	○	●
C2	Definitions of domain-terminology and concepts, with synonyms if applicable.	S2, S3, S4, S6, S7, S10, S14, S16	0.38	0.74	○	○	○
C3	Generalised documentation for non-technical audiences regarding the API and its domain.	S4, S8, S16	0.14	0.55	●	●	●
D1	A list of FAQs.	S4, S7	0.10	0.75	●	●	●
D2	Troubleshooting suggestions.	S4, S8	0.10	0.56	○	○	○○
D3	Diagrammatically representing API components using visual architectural representations.	S6, S13, S20	0.14	0.63	○	○	○○
D4	Contact information for technical support.	S4, S8, S19	0.14	0.21	●	●	●
D5	A printed/printable resource for assistance.	S4, S6, S7, S9, S16	0.24	0.56	○	●	●

Continued on next page...

Key	Description	Primary Sources	ILS	IPS	GCV	AWS	ACV
D6	Licensing information.	S7	0.05	0.66	○	○	●
E1	Searchable knowledge base.	S3, S4, S6, S10, S14, S17, S18	0.33	0.81	●	●	●
E2	Context-specific discussion forum.	S4, S10, S11	0.14	0.58	●	●	●
E3	Quick-links to other relevant documentation frequently viewed by developers.	S6, S16, S20	0.14	0.63	○	○	○
E4	Structured navigational style (e.g., breadcrumbs).	S6, S10, S20	0.14	0.58	●	●	●
E5	Visualised map of navigational paths to certain API components in the website.	S6, S14, S20	0.14	0.50	○	○	○
E6	Consistent look and feel of documentation.	S1, S2, S3, S5, S6, S8, S10, S15, S20	0.43	0.70	●	●	●

C.2 Sources of Documentation

Sources of documentation used for the validation of the taxonomy. For clarity, exact webpages are not referenced for each category, but can be found in supplementary materials which can be downloaded from the URL listed in the paper.

Service	Document Sources
Google Cloud Vision	https://cloud.google.com/vision/docs/quickstart-client-libraries https://googleapis.github.io/google-cloud-java/google-cloud-clients/apidocs/index.html https://cloud.google.com/vision/#cloud-vision-use-cases https://cloud.google.com/vision/docs/quickstart-client-libraries#using_the_client_library https://cloud.google.com/vision/docs/tutorials https://cloud.google.com/community/tutorials?q=vision https://cloud.google.com/vision/docs/samples#mobile_platform_examples https://cloud.google.com/docs/enterprise/best-practices-for-enterprise-organizations https://cloud.google.com/functions/docs/bestpractices/tips https://cloud.google.com/vision/#derive-insight-from-images-with-our-powerful-cloud-vision-api https://cloud.google.com/vision/docs/quickstart-client-libraries https://cloud.google.com/vision/docs/release-notes https://cloud.google.com/vision/docs/reference/rpc/google.rpc#google.rpc.Code https://cloud.google.com/vision/#derive-insight-from-your-images-with-our-powerful-----pretrained-api-models-or-easily-train-custom-vision-models-with-automl-----vision-beta https://cloud.google.com/vision/#insight-from-your-images https://developers.google.com/machine-learning/glossary/ https://cloud.google.com/vision/docs/resources https://cloud.google.com/vision/sla https://cloud.google.com/vision/docs/data-usage https://cloud.google.com/vision/docs/support#searchbox https://cloud.google.com/vision/docs/support

Continued on next page...

Service	Document Sources
Amazon Rekognition	<p>https://docs.aws.amazon.com/rekognition/latest/dg/getting-started.html</p> <p>https://docs.aws.amazon.com/AWSJavaSDK/latest/javadoc/index.html</p> <p>https://aws.amazon.com/blogs/machine-learning/using-amazon-rekognition-to-identify-persons-of-interest-for-law-enforcement/</p> <p>https://aws.amazon.com/rekognition/#Rekognition_Image_Use_Cases</p> <p>https://docs.aws.amazon.com/rekognition/latest/dg/labels-detect-labels-image.html</p> <p>https://aws.amazon.com/rekognition/getting-started/#Tutorials</p> <p>https://aws.amazon.com/blogs/machine-learning/category/artificial-intelligence/amazon-rekognition/</p> <p>https://docs.aws.amazon.com/code-samples/latest/catalog/code-catalog-javascript-example_code-rekognition.html</p> <p>https://docs.aws.amazon.com/rekognition/latest/dg/best-practices.html</p> <p>https://docs.aws.amazon.com/rekognition/latest/dg/API_Operations.html</p> <p>https://aws.amazon.com/rekognition/image-features/</p> <p>https://aws.amazon.com/releasenotes/?tag=releasenotes%23keywords%23amazon-rekognition</p> <p>https://docs.aws.amazon.com/rekognition/latest/dg/setting-up.html</p> <p>https://aws.amazon.com/rekognition/</p> <p>https://aws.amazon.com/rekognition/</p> <p>https://docs.aws.amazon.com/rekognition/latest/dg/limits.html</p> <p>https://aws.amazon.com/rekognition/pricing/</p> <p>https://aws.amazon.com/rekognition/sla/</p> <p>https://aws.amazon.com/rekognition/faqs/</p> <p>https://docs.aws.amazon.com/rekognition/latest/dg/video-troubleshooting.html</p> <p>https://docs.aws.amazon.com/rekognition/latest/dg/rekognition-dg.pdf</p> <p>https://github.com/awsdocs/amazon-rekognition-developer-guide/issues</p> <p>https://forums.aws.amazon.com/thread.jspa?threadID=285910</p>

Continued on next page...

Service	Document Sources
Azure Computer Vision	https://docs.microsoft.com/en-au/azure/cognitive-services/computer-vision/quickstarts-sdk/csharp-analyze-sdk https://docs.microsoft.com/en-us/java/api/overview/azure/cognitiveservices/client/computervision?view=azure-java-stable https://docs.microsoft.com/en-us/azure/architecture/example-scenario/ai/intelligent-apps-image-processing https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/tutorials/java-tutorial https://docs.microsoft.com/en-us/azure/cognitive-services/custom-vision-service/logo-detector-mobile https://docs.microsoft.com/en-au/azure/cognitive-services/computer-vision/tutorials/storage-lab-tutorial https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/tutorials/csharptutorial https://docs.microsoft.com/en-us/azure/cognitive-services/custom-vision-service/getting-started-improving-your-classifier https://docs.microsoft.com/en-au/azure/cognitive-services/computer-vision/home#analyze-images-for-insight https://docs.microsoft.com/en-au/azure/cognitive-services/computer-vision/vision-api-how-to-topics/howtocallvisionapi https://docs.microsoft.com/en-us/azure/cognitive-services/custom-vision-service/release-notes https://docs.microsoft.com/en-au/azure/cognitive-services/computer-vision/ https://azure.microsoft.com/en-au/services/cognitive-services/computer-vision/ https://azure.microsoft.com/en-us/pricing/details/cognitive-services/computer-vision/ https://docs.microsoft.com/en-au/azure/cognitive-services/computer-vision/concept-tagging-images https://docs.microsoft.com/en-au/azure/cognitive-services/computer-vision/home https://azure.microsoft.com/en-us/support/legal/sla/cognitive-services/v1_1/ https://docs.microsoft.com/en-au/azure/cognitive-services/computer-vision/faq https://azure.microsoft.com/en-us/support/legal/

C.3 List of Primary Sources

Below lists the primary sources identified in our systematic mapping study. They are listed in order of assignment to the taxonomy described in Appendix C.1.

- [S1] M. P. Robillard, “What makes APIs hard to learn? Answers from developers,” *IEEE Software*, vol. 26, no. 6, pp. 27–34, 2009, DOI 10.1109/MS.2009.193. ISSN 0740-7459
- [S2] M. P. Robillard and R. Deline, “A field study of API learning obstacles,” *Empirical Software Engineering*, vol. 16, no. 6, pp. 703–732, 2011, DOI 10.1007/s10664-010-9150-8. ISSN 1382-3256
- [S3] A. J. Ko and Y. Riche, “The role of conceptual knowledge in API usability,” in *Proceedings of the 2011 IEEE Symposium on Visual Languages and Human Centric Computing*. Pittsburgh, PA, USA: IEEE, September 2011. DOI 10.1109/VLHCC.2011.6070395. ISBN 978-1-45771245-6 pp. 173–176
- [S4] J. Nykaza, R. Messinger, F. Boehme, C. L. Norman, M. Mace, and M. Gordon, “What programmers really want: Results of a needs assessment for SDK documentation,” in *Proceedings of the 20th Annual International Conference on Computer Documentation*. Toronto, ON, Canada: ACM, October 2002. DOI 10.1145/584955.584976, pp. 133–141
- [S5] R. Watson, M. Mark Stamnes, J. Jeannot-Schroeder, and J. H. Spyridakis, “API documentation and software community values: A survey of open-source API documentation,” in *Proceedings of the 31st ACM International Conference on Design of Communication*. Greenville, SC, USA: ACM, September 2013. DOI 10.1145/2507065.2507076, pp. 165–174
- [S6] S. Y. Jeong, Y. Xie, J. Beaton, B. A. Myers, J. Stylos, R. Ehret, J. Karstens, A. Efeoglu, and D. K. Busse, “Improving documentation for eSOA APIs through user studies,” in *Proceedings of the First International Symposium on End User Development*, vol. 5435 LNCS. Siegen, Germany: Springer, March 2009. DOI 10.1007/978-3-642-00427-8_6. ISSN 0302-9743 pp. 86–105
- [S7] E. Aghajani, C. Nagy, O. L. Vega-Marquez, M. Linares-Vasquez, L. Moreno, G. Bavota, and M. Lanza, “Software Documentation Issues Unveiled,” in *Proceedings of the 41st International Conference on Software Engineering*. Montreal, QC, Canada: IEEE, May 2019. DOI 10.1109/ICSE.2019.00122. ISBN 978-1-72-810869-8. ISSN 0270-5257 pp. 1199–1210
- [S8] S. Haselbock, R. Weinreich, G. Buchgeher, and T. Kriechbaum, “Microservice Design Space Analysis and Decision Documentation: A Case Study on API Management,” in *Proceedings of the 11th International Conference on Service-Oriented Computing and Applications, SOCA 2018*, Paris, France, November 2019, DOI 10.1109/SOCA.2018.00008, pp. 1–8
- [S9] S. Inzunza, R. Juárez-Ramírez, and S. Jiménez, “API Documentation,” in *Proceedings of the 6th World Conference on Information Systems and Technologies*. Naples, Italy: Springer, March 2018. DOI 10.1007/978-3-319-77712-2_22, pp. 229–239
- [S10] M. Meng, S. Steinhardt, and A. Schubert, “Application programming interface documentation: What do software developers want?” *Journal of Technical Writing and Communication*, vol. 48, no. 3, pp. 295–330, August 2018, DOI 10.1177/0047281617721853. ISSN 1541-3780
- [S11] R. S. Geiger, N. Varoquaux, C. Mazel-Cabasse, and C. Holdgraf, “The Types, Roles, and Practices of Documentation in Data Analytics Open Source Software Libraries: A Collaborative Ethnography of Documentation Work,” *Computer Supported Cooperative Work: CSCW: An International Journal*, vol. 27, no. 3-6, pp. 767–802, May 2018, DOI 10.1007/s10606-018-9333-1. ISSN 1573-7551
- [S12] A. Head, C. Sadowski, E. Murphy-Hill, and A. Knight, “When not to comment: Questions and tradeoffs with API documentation for C++ projects,” in *Proceedings of the 40th International Conference on Software Engineering*, ser. questions and tradeoffs with API documentation for C++ projects. Gothenburg, Sweden: ACM, May 2018. DOI 10.1145/3180155.3180176. ISSN 0270-5257 pp. 643–653

- [S13] L. Aversano, D. Guardabascio, and M. Tortorella, “Analysis of the Documentation of ERP Software Projects,” *Procedia Computer Science*, vol. 121, pp. 423–430, January 2017, DOI 10.1016/j.procs.2017.11.057. ISSN 1877-0509
- [S14] M. P. Robillard, A. Marcus, C. Treude, G. Bavota, O. Chaparro, N. Ernst, M. A. Gerosall, M. Godfrey, M. Lanza, M. Linares-Vásquez, G. C. Murphy, L. Moreno, D. Shepherd, and E. Wong, “On-demand developer documentation,” in *Proceedings of the 33rd IEEE International Conference on Software Maintenance and Evolution*. Shanghai, China: IEEE, September 2017. DOI 10.1109/ICSME.2017.17, pp. 479–483
- [S15] R. Watson, “Development and application of a heuristic to assess trends in API documentation,” in *Proceedings of the 30th ACM International Conference on Design of Communication*. Seattle, WA, USA: ACM, October 2012. DOI 10.1145/2379057.2379112. ISBN 978-1-45031497-8 pp. 295–302
- [S16] W. Maalej and M. P. Robillard, “Patterns of knowledge in API reference documentation,” *IEEE Transactions on Software Engineering*, 2013, DOI 10.1109/TSE.2013.12. ISSN 0098-5589
- [S17] D. L. Parnas and S. A. Vilkomir, “Precise documentation of critical software,” in *Proceedings of 10th IEEE International Symposium on High Assurance Systems Engineering*. Plano, TX, USA: IEEE, November 2007. DOI 10.1109/HASE.2007.63. ISSN 1530-2059 pp. 237–244
- [S18] C. Bottomley, “What part writer? What part programmer? A survey of practices and knowledge used in programmer writing,” in *Proceedings of the 2005 IEEE International Professional Communication Conference*. Limerick, Ireland: IEEE, July 2005. DOI 10.1109/IPCC.2005.1494255, pp. 802–812
- [S19] A. Taulavuori, E. Niemelä, and P. Kallio, “Component documentation - A key issue in software product lines,” *Information and Software Technology*, vol. 46, no. 8, pp. 535–546, June 2004, DOI 10.1016/j.infsof.2003.10.004. ISSN 0950-5849
- [S20] J. Kotula, “Using patterns to create component documentation,” *IEEE Software*, vol. 15, no. 2, pp. 84–92, 1998, DOI 10.1109/52.663791. ISSN 0740-7459
- [S21] S. G. McLellan, A. W. Roesler, J. T. Tempest, and C. I. Spinuzzi, “Building more usable APIs,” *IEEE Software*, vol. 15, no. 3, pp. 78–86, 1998, DOI 10.1109/52.676963. ISSN 0740-7459

C.4 Survey Questions

This section contains the exact text of the survey described in Section 8.5.1. Our instrument also included questions where answers were not included in the research reported in this article, e.g. questions 1 and 2 regarding consent and ensuring participants have had development experience. Images used within the survey have been removed.

Developer opinions towards the importance of web API documentation recommendations

In this study, we are finding out how important recommendations of web API documentation are to developers. From this, we will improve AI-powered APIs. While there are screenshots of example APIs in the questions, think of an API that you have used based on **your own prior experience** when answering these questions. Thanks for taking the time to answer these questions; it should only take you about **10–20 minutes** to complete.

Attribution Notice

Portions of this questionnaire are reproduced from work created and shared by Google and used according to terms described in the Creative Commons 3.0 Attribution License.

Implementation-specific documentation of web APIs

When answering these questions please answer with respect to **your own experience** in learning web APIs (if applicable). Any examples provided exist solely to help illustrate the statement. For each question, please nominate how much you agree with the following statements: *[Strongly agree, Somewhat agree, Neither agree nor disagree, Somewhat disagree, Strongly disagree]*

- Q3a. I think quick-start guides with code that help me get started with an API's client library are important. e.g., quick-start guides that show how to get started and interact with the API and its responses.
- Q3b. I don't find low-level documentation of all classes and methods particularly helpful. e.g., a generated online reference manual from Javadoc comments.
- Q3c. I would imagine that explanations of the API's high-level architecture, context and rationale would be important to better understand how to consume the API. e.g., a graphic showing how the API could fit into the wider context of an application.
- Q3d. If I want to understand why an API did something that I didn't expect, the source code comments generally don't help me. e.g., an example from the Lodash API that describes why `set.add` isn't directly returned.
- Q3e. I find small code snippets with comments to demonstrate a single component's basic functionality within the API a useful way to learn. e.g., 10-30 lines of code to demonstrating various how-tos of a computer vision API.
- Q3f. I think it's cumbersome to read through step-by-step tutorials that show how to build something non-trivial with multiple components using the API. e.g., a ten-step tutorial documenting how to combine face recognition, face analysis, scene description, and landmark detection API components to generate descriptions of photos.

- Q3g. I think it's useful to download source code of production-ready applications that demonstrate the use of multiple facets of the API. e.g., a downloadable iOS app that demonstrates how to perform image analysis on an iPhone/iPad.
- Q3h. I think official documentation describing the 'best-practices' of how to use the API to assist with debugging and efficiency is not helpful. e.g., an article describing the correct ways of doing things, the best tools to use, and how to write well-performing code.
- Q3i. I believe an exhaustive list of all major components in the API without excessive detail would be useful when learning an API. e.g., a computer vision web API might list object detection, object localisation, facial recognition, and facial comparison as its 4 components.
- Q3j. I believe minimum system requirements and/or dependencies to use the API do not always need to be part of official documentation. e.g., I can find descriptions of how to get started with a Python environment for a cloud platform on community forums instead of the API's website.
- Q3k. I think instructions on how to install or access the API, update it, and the frequency of its release cycle is all useful information to know about. e.g., a list showing the latest releases, what was added and how to update your application to make use of it.
- Q3l. Error codes describing specific problems with an API are not helpful. e.g., a list of canonical HTTP error codes and how to interpret them.
-

Rationale-specific documentation of web APIs

When answering these questions please answer with respect to **your own experience** in learning web APIs (if applicable). Any examples provided exist solely to help illustrate the statement. For each question, please nominate how much you agree with the following statements: [*Strongly agree, Somewhat agree, Neither agree nor disagree, Somewhat disagree, Strongly disagree*]

- Q4a. I think that, as a starting point when beginning to learn about an API, I would like to read about descriptions of the API's purpose and overview.
- Q4b. I don't find descriptions of the types of applications the API can develop helpful.
- Q4c. I believe that descriptions of the types of developers who should and shouldn't use the API is important to know.
- Q4d. I don't think that descriptions of the types of end-users who will use the product built using the API is important to know in advance.
- Q4e. I think that if I read success stories about when the API was previously used in production, I would have a better indicator of how I could use that API.
- Q4f. I think that documentation that compares an API to other, similar APIs confusing and not important.
- Q4g. I believe it is important to know about what the limitations are on what the API can and cannot provide.

Conceptual-specific documentation of web APIs

When answering these questions please answer with respect to **your own experience** in learning web APIs (if applicable). Any examples provided exist solely to help illustrate the

statement. For each question, please nominate how much you agree with the following statements: [*Strongly agree, Somewhat agree, Neither agree nor disagree, Somewhat disagree, Strongly disagree*]

- Q5a. I wouldn't read through theory about the API's domain that relates theoretical concepts to API components and how both work together.
 - Q5b. I think it is important to know the definitions of the API's domain-specific terminology and concepts (with synonyms where needed). e.g., a computer vision API that uses machine learning should list machine learning concepts.
 - Q5c. It's not really important to document information about the API to non-technical audiences, such as managers and other stakeholders. e.g., pricing information, uptime information, QoS metrics/SLAs etc.
-

General-support documentation of web APIs

When answering these questions please answer with respect to **your own experience** in learning web APIs (if applicable). Any examples provided exist solely to help illustrate the statement. For each question, please nominate how much you agree with the following statements: [*Strongly agree, Somewhat agree, Neither agree nor disagree, Somewhat disagree, Strongly disagree*]

- Q6a. I find lists of Frequently Asked Questions (FAQs) helpful.
 - Q6b. When something goes wrong, I don't read through troubleshooting suggestions for specific problems straight away as I like to solve it myself.
 - Q6c. I like to see diagrammatic representations of an API's components using visual architectural visualisations. e.g., UML class diagram, sequence diagram.
 - Q6d. I wouldn't look for email addresses and/or phone number for technical support in an API's documentation.
 - Q6e. I generally refer to a programmer's reference guide or textbook about the API when I need to.
 - Q6f. I don't think it's important to read about the licensing information about the API.
-

The effect of structure and tooling on web API documentation

When answering these questions please answer with respect to **your own experience** in learning web APIs (if applicable). Any examples provided exist solely to help illustrate the statement. For each question, please nominate how much you agree with the following statements: [*Strongly agree, Somewhat agree, Neither agree nor disagree, Somewhat disagree, Strongly disagree*]

- Q7a. I would like to use a searchable knowledge base to find information.
- Q7b. I think a context-specific discussion forum between developers isn't very helpful as it just introduces noise. e.g., issue trackers, Slack group.
- Q7c. I think links to other similar documentation frequently viewed by other developers would be useful. e.g., 'people who viewed this also viewed...'
- Q7d. If I get lost within the API's documentation, a 'breadcrumbs'-style of navigation isn't very useful to me.

- Q7e. A visualised map of navigational paths to common API components in the website would be useful to have. e.g., a large and complex API for Enterprise Service-Oriented Architecture where I could click into various boxes to read about components and arrows to read about how they are related.
- Q7f. I believe ensuring consistent look and feel of all documentation isn't necessary to a good API documentation.
-

Demographics

- Q8a. Are you, or do you aspire to be, a professional programmer? Or would you consider programming a hobby?
[Professional, Hobbyist]
- Q8b. How many years have you been programming?
[1–5 years, 6–10 years, 11–15 years, 16–20 years, 21–30 years, 31–40 years, 41+ years]
- Q8c. In what type of role would you say your current job falls into?
[Back-end developer, Data or business analyst, Data scientist or machine learning specialist, Database administrator, Designer, Desktop or enterprise applications developer, DevOps specialist, Educator or academic researcher, Embedded applications or devices developer, Engineering manager, Front-end developer, Full-stack developer, Game or graphics developer, Marketing or sales professional, Mobile developer, Product manager, QA or test developer, Student, System administration]
- Q8d. What level of seniority would you say this role falls into?
[Intern Role, Graduate Role, Junior Role, Mid-Tier Role, Senior Role, Lead Role, Principal Role, Management, N/A (e.g., I am a student), Other]
- Q8e. What industry would you say you work in?
[Cloud-based solutions or services, Consulting, Data and analytics, Financial technology or services, Healthcare technology or services, Information technology, Media, advertising, publishing, or entertainment, Other software development, Retail or eCommerce, Software as a service (SaaS) development, Web development or design, N/A (e.g., I am a student), Other industry not listed here]
-

*** End of Survey ***

APPENDIX D

Authorship Statements

Deakin University Authorship Procedure

Schedule A: Authorship Statement

1. Details of the publication and executive author

Title of publication	Losing Confidence in Quality: Unspoken Evolution of Computer Vision Services
Publication details	Presented at the 35th IEEE International Conference on Software Maintenance and Evolution, Cleveland, USA, 2019
Name of executive author	Alex Cummaudo
School/Institute/Division if at Deakin Organisation and address if non-Deakin	Applied Artificial Intelligence Institute
Email or phone	ca@deakin.edu.au

2. Inclusion of publication in a thesis

Is it intended to include this publication in a higher degree by research (HDR) thesis?
*If Yes, please complete Section 3
 If No, go straight to Section 4.*

3. HDR thesis author's declaration

Name of HDR thesis author if different from above. <i>(If the same, write "as above")</i>	As above
School/Institute/Division if at Deakin	Applied Artificial Intelligence Institute
Thesis title	Taming the Evolving Black Box: Towards Improved Integration and Documentation of Intelligent Web Services
If there are multiple authors, give a full description of HDR thesis author's contribution to the publication.	See page 2

I declare that the above is an accurate description of my contribution to this paper, and the contributions of other authors are as described below.

Signed:



Dated: 22 July 2019

4. Description of all author contributions

Name and affiliation of author 1	Alex Cummaudo Applied Artificial Intelligence Institute Deakin University
Contribution of author 1	Alex Cummaudo initiated the conception of the project. Additionally, he designed a detailed methodology, conducted all data collection via a data-collection instrument he designed and implemented and performed a majority of data analysis. He drafted the full manuscript and made further revisions, modifications and prepared the camera ready version for publication in the conference proceedings.
Name and affiliation of author 2	Rajesh Vasa Applied Artificial Intelligence Institute Deakin University
Contribution of author 2	Rajesh Vasa contributed to the initial conception of this project by providing high-level guidance over overview of what the project and its experiments should comprise of. Rajesh also contributed to detailed revisions of the initial manuscripts, and assisted in advising Alex Cummaudo on improved analytical insight into the collected results. Rajesh Vasa also assisted in shaping the paper to specifically target the conference audience. Rajesh Vasa is the primary supervisor of Alex Cummaudo.
Name and affiliation of author 3	John Grundy Faculty of Information Technology Monash University
Contribution of author 3	John Grundy provided high-level oversight of the project. He contributed to detailed reviews of the methodology and manuscript. John Grundy is the external supervisor of Alex Cummaudo.
Name and affiliation of author 4	Mohamed Abdelrazek School of Information Technology Deakin University
Contribution of author 4	Mohamed Abdelrazek made final edits and suggestions to the final draft of the manuscript before submitting for peer review. Mohamed Abdelrazek is an associate supervisor of Alex Cummaudo.
Name and affiliation of author 5	Andrew Cain School of Information Technology Deakin University
Contribution of author 5	Andrew Cain made edits and suggestions to the abstract and introduction paragraphs of the manuscript. Andrew Cain is an associate supervisor of Alex Cummaudo.

5. Author declarations

I agree to be named as one of the authors of this work, and confirm:

- i. that I have met the authorship criteria set out in the Deakin University Research Conduct Policy,
- ii. that there are no other authors according to these criteria,
- iii. that the description in Section 4 of my contribution(s) to this publication is accurate,
- iv. that the data on which these findings are based are stored as set out in Section 7 below.

If this work is to form part of an HDR thesis as described in Sections 2 and 3, I further

- v. consent to the incorporation of the publication into the candidate's HDR thesis submitted to Deakin University and, if the higher degree is awarded, the subsequent publication of the thesis by the university (subject to relevant Copyright provisions).

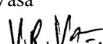
Author 1

Alex Cummaudo

Signed: 
Dated: 22 July 2019

Author 2

Rajesh Vasa

Signed: 
Dated: 22 July 2019

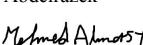
Author 3

John Grundy

Signed: 
Dated: 22 July 2019

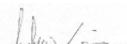
Author 4

Mohamed Abdelrazek

Signed: 
Dated: 22 July 2019

Author 5

Andrew Cain

Signed: 
Dated: 22 July 2019

6. Other contributor declarations

There are no other contributors for this publication to declare.

7. Data storage

The original data for this project are stored in the following locations. (The locations must be within an appropriate institutional setting. If the executive author is a Deakin staff member and data are stored outside Deakin University, permission for this must be given by the Head of Academic Unit within which the executive author is based.)

Data format	Comma separated values (CSV), iPython Notebook
Storage location	Deakin University Research Data Store (RDS) Location: RDS29448-Alex-Cummaudo-PhD/results/icsme19

8. Additional notices

This form must be retained by the executive author, within the school or institute in which they are based.

If the publication is to be included as part of an HDR thesis, a copy of this form must be included in the thesis with the publication.

Deakin University Authorship Procedure

Schedule A: Authorship Statement

1. Details of the publication and executive author

Title of publication	What should I document? A preliminary systematic mapping study into API documentation knowledge
Publication details	Presented at the 13th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), Porto de Galinhas, Brazil, 2019
Name of executive author	Alex Cummaudo
School/Institute/Division if at Deakin	Applied Artificial Intelligence Institute
Organisation and address if non-Deakin	
Email or phone	ca@deakin.edu.au

2. Inclusion of publication in a thesis

Is it intended to include this publication in a higher degree by research (HDR) thesis? Yes
*If Yes, please complete Section 3
If No, go straight to Section 4.*

3. HDR thesis author's declaration

Name of HDR thesis author if different from above. <i>(If the same, write "as above")</i>	As above
School/Institute/Division if at Deakin	Applied Artificial Intelligence Institute
Thesis title	Taming the Evolving Black Box: Towards Improved Integration and Documentation of Intelligent Web Services
If there are multiple authors, give a full description of HDR thesis author's contribution to the publication.	See page 2

I declare that the above is an accurate description of my contribution to this paper, and the contributions of other authors are as described below.

Signed:



Dated: 22 July 2019

4. Description of all author contributions

Name and affiliation of author 1	Alex Cummaudo Applied Artificial Intelligence Institute Deakin University
Contribution of author 1	Alex Cummaudo devised the conception of this project and the intended objectives and hypotheses. Additionally, he designed a detailed methodology, conducted data collection with a custom tool he wrote himself and performed analysis. He drafted the manuscript and made further revisions, modifications and prepared the camera ready version for publication in the conference proceedings.
Name and affiliation of author 2	Rajesh Vasa Applied Artificial Intelligence Institute Deakin University
Contribution of author 2	Rajesh Vasa contributed to the initial conception of this project by providing high-level guidance over overview of what the project and its experiments should comprise of. Rajesh also contributed to detailed revisions of the initial manuscripts, and assisted in advising Alex Cummaudo on improved analytical insight into the collected results. Rajesh Vasa also assisted in shaping the paper to specifically target the conference audience. Rajesh Vasa is the primary supervisor of Alex Cummaudo.
Name and affiliation of author 3	John Grundy Faculty of Information Technology Monash University
Contribution of author 3	John Grundy provided high-level oversight of the project. He contributed to detailed reviews of the methodology and manuscript. John Grundy is the external supervisor of Alex Cummaudo.

5. Author declarations

I agree to be named as one of the authors of this work, and confirm:

- i. that I have met the authorship criteria set out in the Deakin University Research Conduct Policy,
- ii. that there are no other authors according to these criteria,
- iii. that the description in Section 4 of my contribution(s) to this publication is accurate,
- iv. that the data on which these findings are based are stored as set out in Section 7 below.

If this work is to form part of an HDR thesis as described in Sections 2 and 3, I further

- v. consent to the incorporation of the publication into the candidate's HDR thesis submitted to Deakin University and, if the higher degree is awarded, the subsequent publication of the thesis by the university (subject to relevant Copyright provisions).

Author 1

Alex Cummaudo

Signed: 
Dated: 22 July 2019

Author 2

Rajesh Vasa

Signed: 
Dated: 22 July 2019

Author 3

John Grundy

Signed: 
Dated: 22 July 2019

6. Other contributor declarations

There are no other contributors for this publication to declare.

7. Data storage

The original data for this project are stored in the following locations. (The locations must be within an appropriate institutional setting. If the executive author is a Deakin staff member and data are stored outside Deakin University, permission for this must be given by the Head of Academic Unit within which the executive author is based.)

Data format	Comma separated values (CSV), Portable Document Format (PDF)
Storage location	Deakin University Research Data Store (RDS) Location: RDS29448-Alex-Cummaudo-PhD/results/esem19

8. Additional notices

This form must be retained by the executive author, within the school or institute in which they are based.

If the publication is to be included as part of an HDR thesis, a copy of this form must be included in the thesis with the publication.

Deakin University Authorship Procedure

Schedule A: Authorship Statement

1. Details of the publication and executive author

Title of publication	Interpreting Cloud Computer Vision Pain-Points: A Mining Study of Stack Overflow
Publication details	Presented at the 42nd International Conference on Software Engineering, Seoul, South Korea, 2020
Name of executive author	Alex Cummaudo
School/Institute/Division if at Deakin	Applied Artificial Intelligence Institute
Organisation and address if non-Deakin	
Email or phone	ca@deakin.edu.au

2. Inclusion of publication in a thesis

Is it intended to include this publication in a higher degree by research (HDR) thesis? Yes
*If Yes, please complete Section 3
If No, go straight to Section 4.*

3. HDR thesis author's declaration

Name of HDR thesis author if different from above. <i>(If the same, write "as above")</i>	As above
School/Institute/Division if at Deakin	Applied Artificial Intelligence Institute
Thesis title	Taming the Evolving Black Box: Towards Improved Integration and Documentation of Intelligent Web Services
If there are multiple authors, give a full description of HDR thesis author's contribution to the publication.	See page 2

I declare that the above is an accurate description of my contribution to this paper, and the contributions of other authors are as described below.

Signed:



Dated: 27 August 2019

4. Description of all author contributions

Name and affiliation of author 1

Alex Cummaudo
Applied Artificial Intelligence Institute
Deakin University

Contribution of author 1

Alex Cummaudo initiated the conception of the project. Additionally, he designed a detailed methodology, conducted the experiment and mined data against the methodology devised, performed a majority of data analysis and categorised 525 Stack Overflow posts. He drafted the full manuscript and made further revisions, modifications and prepared the camera ready version for publication in the conference proceedings.

Name and affiliation of author 2

Rajesh Vasa
Applied Artificial Intelligence Institute
Deakin University

Contribution of author 2

Rajesh Vasa contributed to the initial conception of this project by providing high-level guidance over overview of what the project and its experiments should comprise of. Rajesh also contributed to detailed revisions of the initial manuscripts, and assisted in advising Alex Cummaudo on improved analytical insight into the collected results. Rajesh Vasa is the primary supervisor of Alex Cummaudo.

Name and affiliation of author 3

Scott Barnett
Applied Artificial Intelligence Institute
Deakin University

Contribution of author 3

Scott Barnett conducted a statistical distribution analysis for this experiment. He contributed to detailed reviews of the methodology and manuscript. He also contributed a major section of the work regarding Technical Domain Models.

Name and affiliation of author 4

John Grundy
Faculty of Information Technology
Monash University

Contribution of author 4

John Grundy provided high-level oversight of the project. He contributed to detailed reviews of the methodology and manuscript. John Grundy is the external supervisor of Alex Cummaudo.

Name and affiliation of author 5

Mohamed Abdelrazek
School of Information Technology
Deakin University

Contribution of author 5

Mohamed Abdelrazek made final edits and suggestions to the final draft of the manuscript before submitting for peer review. Mohamed Abdelrazek is an associate supervisor of Alex Cummaudo.

5. Author declarations

I agree to be named as one of the authors of this work, and confirm:

- i. that I have met the authorship criteria set out in the Deakin University Research Conduct Policy,
- ii. that there are no other authors according to these criteria,
- iii. that the description in Section 4 of my contribution(s) to this publication is accurate,
- iv. that the data on which these findings are based are stored as set out in Section 7 below.

If this work is to form part of an HDR thesis as described in Sections 2 and 3, I further

- v. consent to the incorporation of the publication into the candidate's HDR thesis submitted to Deakin University and, if the higher degree is awarded, the subsequent publication of the thesis by the university (subject to relevant Copyright provisions).

Author 1

Alex Cummaudo

Signed: 
Dated: 27 August 2019

Author 2

Rajesh Vasa

Signed: 
Dated: 27 August 2019

Author 3

Scott Barnett

Signed: 
Dated: 27 August 2019

Author 4

John Grundy

Signed: 
Dated: 27 August 2019

Author 5

Mohamed Abdelrazek

Signed: 
Dated: 27 August 2019

6. Other contributor declarations

There are no other contributors for this publication to declare.

7. Data storage

The original data for this project are stored in the following locations. (The locations must be within an appropriate institutional setting. If the executive author is a Deakin staff member and data are stored outside Deakin University, permission for this must be given by the Head of Academic Unit within which the executive author is based.)

Data format	Comma separated values (CSV), Excel Spreadsheet
Storage location	Deakin University Research Data Store (RDS) Location: RDS29448-Alex-Cummaudo-PhD/results/icse20

8. Additional notices

This form must be retained by the executive author, within the school or institute in which they are based.

If the publication is to be included as part of an HDR thesis, a copy of this form must be included in the thesis with the publication.

Deakin University Authorship Procedure

Schedule A: Authorship Statement

1. Details of the publication and executive author

Title of publication	Threshy: Supporting safe usage of intelligent web services
Publication details	Presented at the 28th Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (Demonstrations Track)
Name of executive author	Alex Cummaudo
School/Institute/Division if at Deakin	Applied Artificial Intelligence Institute
Organisation and address if non-Deakin	
Email or phone	ca@deakin.edu.au

2. Inclusion of publication in a thesis

Is it intended to include this publication in a higher degree by research (HDR) thesis? Yes
*If Yes, please complete Section 3
If No, go straight to Section 4.*

3. HDR thesis author's declaration

Name of HDR thesis author if different from above. <i>(If the same, write "as above")</i>	As above
School/Institute/Division if at Deakin	Applied Artificial Intelligence Institute
Thesis title	Taming the Evolving Black Box: Towards Improved Integration and Documentation of Intelligent Web Services
If there are multiple authors, give a full description of HDR thesis author's contribution to the publication.	See page 2

I declare that the above is an accurate description of my contribution to this paper, and the contributions of other authors are as described below.

Signed:



Dated: 14 January 2020

4. Description of all author contributions

Name and affiliation of author 1	Alex Cummaudo Applied Artificial Intelligence Institute Deakin University
Contribution of author 1	Alex Cummaudo drafted the manuscript for this work, prepared visualisations within the paper, made further revisions and changes per reviewer feedback and (will) prepare the camera ready version for publication in the conference proceedings. Alex also created the required demonstration video required for this publication (https://bit.ly/2YKeYhE), drafting the voiceover script, recording the voiceover itself, producing animations within the video, and recording a video of the tool in use.
Name and affiliation of author 2	Scott Barnett Applied Artificial Intelligence Institute Deakin University
Contribution of author 2	Scott Barnett contributed to the initial conception of this project by providing high-level guidance on the conceptual workflow and associated tooling. He also assisted in implementing the tool. Scott contributed to detailed reviews of the methodology and manuscript and provided feedback for the required video demonstration. Scott also provided a detailed revision of the manuscript and provided contribution to specific portions of the paper.
Name and affiliation of author 3	Rajesh Vasa Applied Artificial Intelligence Institute Deakin University
Contribution of author 3	Rajesh Vasa contributed guidance to the conceptual workflow and associated tooling presented in this paper. Rajesh also contributed to detailed revisions of the initial manuscripts and provided feedback on the tool and its associated demonstration video. Rajesh Vasa is the primary supervisor of Alex Cummaudo.
Name and affiliation of author 4	John Grundy Faculty of Information Technology Monash University
Contribution of author 4	John Grundy provided high-level oversight of the project. He contributed to detailed reviews of the manuscript and associated demonstration video. John Grundy is the external supervisor of Alex Cummaudo.

5. Author declarations

I agree to be named as one of the authors of this work, and confirm:

- i. that I have met the authorship criteria set out in the Deakin University Research Conduct Policy,
- ii. that there are no other authors according to these criteria,
- iii. that the description in Section 4 of my contribution(s) to this publication is accurate,
- iv. that the data on which these findings are based are stored as set out in Section 7 below.

If this work is to form part of an HDR thesis as described in Sections 2 and 3, I further

- v. consent to the incorporation of the publication into the candidate's HDR thesis submitted to Deakin University and, if the higher degree is awarded, the subsequent publication of the thesis by the university (subject to relevant Copyright provisions).

Author 1

Alex Cummaudo


Signed:
Dated: 14 January 2020

Author 2

Scott Barnett


Signed:
Dated: 14 January 2020

Author 3

Rajesh Vasa


Signed:
Dated: 14 January 2020

Author 4

John Grundy


Signed:
Dated: 14 January 2020

6. Other contributor declarations

There are no other contributors for this publication to declare.

7. Data storage

The original data for this project are stored in the following locations. (The locations must be within an appropriate institutional setting. If the executive author is a Deakin staff member and data are stored outside Deakin University, permission for this must be given by the Head of Academic Unit within which the executive author is based.)

Data format	JavaScript, Python, HTML, Keynote File, iMovie File
Storage location	Deakin University Research Data Store (RDS) Location: RDS29448-Alex-Cummaudo-PhD/results/icse(d)20

8. Additional notices

This form must be retained by the executive author, within the school or institute in which they are based.

If the publication is to be included as part of an HDR thesis, a copy of this form must be included in the thesis with the publication.

Deakin University Authorship Procedure

Schedule A: Authorship Statement

1. Details of the publication and executive author

Title of publication	Assessing API documentation knowledge for computer vision services
Publication details	Submitted to the IEEE Transactions on Software Engineering
Name of executive author	Alex Cummaudo
School/Institute/Division if at Deakin	Applied Artificial Intelligence Institute
Organisation and address if non-Deakin	
Email or phone	ca@deakin.edu.au

2. Inclusion of publication in a thesis

Is it intended to include this publication in a higher degree by research (HDR) thesis? Yes
*If Yes, please complete Section 3
If No, go straight to Section 4.*

3. HDR thesis author's declaration

Name of HDR thesis author if different from above. <i>(If the same, write "as above")</i>	As above
School/Institute/Division if at Deakin	Applied Artificial Intelligence Institute
Thesis title	Taming the Evolving Black Box: Towards Improved Integration and Documentation of Intelligent Web Services
If there are multiple authors, give a full description of HDR thesis author's contribution to the publication.	See page 2

I declare that the above is an accurate description of my contribution to this paper, and the contributions of other authors are as described below.

Signed: 
Dated: 10 March 2020

4. Description of all author contributions

Name and affiliation of author 1	Alex Cummaudo Applied Artificial Intelligence Institute Deakin University
Contribution of author 1	Alex Cummaudo devised the conception of this project and the intended objectives and hypotheses. Additionally, he designed a detailed methodology, conducted data collection with a custom tool he wrote himself and performed analysis. He also designed and conducted the survey instrument listed within this publication. He drafted the full manuscript and made further revisions, modifications. He made detailed revisions to all graphs and figures within this paper.
Name and affiliation of author 2	Rajesh Vasa Applied Artificial Intelligence Institute Deakin University
Contribution of author 2	Rajesh Vasa contributed to the initial conception of this project by providing high-level guidance over overview of what the project and its experiments should comprise of. Rajesh also contributed to detailed revisions of the initial manuscript, and assisted in advising Alex Cummaudo on improved analytical insight into the collected results. Rajesh Vasa is the primary supervisor of Alex Cummaudo.
Name and affiliation of author 3	John Grundy Faculty of Information Technology Monash University
Contribution of author 3	John Grundy provided high-level oversight of the project. He contributed to detailed reviews of the methodology and manuscript. John Grundy is the external supervisor of Alex Cummaudo.
Name and affiliation of author 4	Mohamed Abdelrazek School of Information Technology Deakin University
Contribution of author 4	Mohamed Abdelrazek made final edits and suggestions to the final draft of the manuscript before submitting for peer review. Mohamed Abdelrazek is an associate supervisor of Alex Cummaudo.

5. Author declarations

I agree to be named as one of the authors of this work, and confirm:

- i. that I have met the authorship criteria set out in the Deakin University Research Conduct Policy,
- ii. that there are no other authors according to these criteria,
- iii. that the description in Section 4 of my contribution(s) to this publication is accurate,
- iv. that the data on which these findings are based are stored as set out in Section 7 below.

If this work is to form part of an HDR thesis as described in Sections 2 and 3, I further

- v. consent to the incorporation of the publication into the candidate's HDR thesis submitted to Deakin University and, if the higher degree is awarded, the subsequent publication of the thesis by the university (subject to relevant Copyright provisions).

Author 1

Alex Cummaudo


Signed:
Dated: 10 March 2020

Author 2

Rajesh Vasa


Signed:
Dated: 10 March 2020

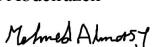
Author 3

John Grundy


Signed:
Dated: 10 March 2020

Author 4

Mohamed Abdelrazek


Signed:
Dated: 10 March 2020

6. Other contributor declarations

There are no other contributors for this publication to declare.

7. Data storage

The original data for this project are stored in the following locations. (The locations must be within an appropriate institutional setting. If the executive author is a Deakin staff member and data are stored outside Deakin University, permission for this must be given by the Head of Academic Unit within which the executive author is based.)

Data format	Comma separated values (CSV), Portable Document Format (PDF)
Storage location	Deakin University Research Data Store (RDS) Location: RDS29448-Alex-Cummaudo-PhD/results/tse2020

8. Additional notices

This form must be retained by the executive author, within the school or institute in which they are based.

If the publication is to be included as part of an HDR thesis, a copy of this form must be included in the thesis with the publication.

Deakin University Authorship Procedure

Schedule A: Authorship Statement

1. Details of the publication and executive author

Title of publication	Beware the evolving ‘intelligent’ web service! An integration architecture tactic to guard AI-first components
Publication details	Presented at the 28th Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering
Name of executive author	Alex Cummaudo
School/Institute/Division if at Deakin	Applied Artificial Intelligence Institute
Organisation and address if non-Deakin	
Email or phone	ca@deakin.edu.au

2. Inclusion of publication in a thesis

Is it intended to include this publication in a higher degree by research (HDR) thesis? Yes
*If Yes, please complete Section 3
 If No, go straight to Section 4.*

3. HDR thesis author’s declaration

Name of HDR thesis author if different from above. <i>(If the same, write “as above”)</i>	As above
School/Institute/Division if at Deakin	Applied Artificial Intelligence Institute
Thesis title	Taming the Evolving Black Box: Towards Improved Integration and Documentation of Intelligent Web Services
If there are multiple authors, give a full description of HDR thesis author’s contribution to the publication.	See page 2

I declare that the above is an accurate description of my contribution to this paper, and the contributions of other authors are as described below.

Signed:



Dated: 10 March 2020

4. Description of all author contributions

Name and affiliation of author 1	Alex Cummaudo Applied Artificial Intelligence Institute Deakin University
Contribution of author 1	Alex Cummaudo initiated the conception of the project, designed the architecture that is described in this paper and implemented its codebase. He designed the architectural designs appearing in the paper and many drafts of this design. Additionally, he designed a detailed methodology, conducted the experiment, performed data collection, and performed a majority of data analysis. He drafted the full manuscript and made further revisions, modifications and (will) prepare the camera ready version for publication in the conference proceedings.
Name and affiliation of author 2	Scott Barnett Applied Artificial Intelligence Institute Deakin University
Contribution of author 2	Scott Barnett contributed to the initial concept of this project by providing feedback of the architecture designed. Scott also provided feedback to the architectural designs and figures/graphs appearing in this paper. Scott provided detailed reviews and edits of the introduction, approach and evaluation sections of the manuscript, and contributed to the limitations section.
Name and affiliation of author 3	Rajesh Vasa Applied Artificial Intelligence Institute Deakin University
Contribution of author 3	Rajesh Vasa contributed to the initial conception of this project by providing high-level guidance over overview of what the project and its experiments should comprise of. Rajesh also contributed to detailed revisions of the initial manuscripts, and assisted in advising Alex Cummaudo on improved analytical insight into the collected results. Rajesh Vasa is the primary supervisor of Alex Cummaudo.
Name and affiliation of author 4	John Grundy Faculty of Information Technology Monash University
Contribution of author 4	John Grundy provided high-level oversight of the project. He contributed to detailed reviews of the methodology and manuscript. John Grundy is the external supervisor of Alex Cummaudo.
Name and affiliation of author 5	Mohamed Abdelrazek School of Information Technology Deakin University
Contribution of author 5	Mohamed Abdelrazek made final edits and suggestions to the final draft of the manuscript before submitting for peer review. Mohamed Abdelrazek is an associate supervisor of Alex Cummaudo.

5. Author declarations

I agree to be named as one of the authors of this work, and confirm:

- i. that I have met the authorship criteria set out in the Deakin University Research Conduct Policy,
- ii. that there are no other authors according to these criteria,
- iii. that the description in Section 4 of my contribution(s) to this publication is accurate,
- iv. that the data on which these findings are based are stored as set out in Section 7 below.

If this work is to form part of an HDR thesis as described in Sections 2 and 3, I further

- v. consent to the incorporation of the publication into the candidate's HDR thesis submitted to Deakin University and, if the higher degree is awarded, the subsequent publication of the thesis by the university (subject to relevant Copyright provisions).

Author 1

Alex Cummaudo


Signed:
Dated: 10 March 2020

Author 2

Scott Barnett


Signed:
Dated: 10 March 2020

Author 3

Rajesh Vasa


Signed:
Dated: 10 March 2020

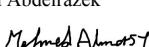
Author 4

John Grundy


Signed:
Dated: 10 March 2020

Author 5

Mohamed Abdelrazek


Signed:
Dated: 10 March 2020

6. Other contributor declarations

There are no other contributors for this publication to declare.

7. Data storage

The original data for this project are stored in the following locations. (The locations must be within an appropriate institutional setting. If the executive author is a Deakin staff member and data are stored outside Deakin University, permission for this must be given by the Head of Academic Unit within which the executive author is based.)

Data format	Comma separated values (CSV), Excel Spreadsheet, Ruby Code
Storage location	Deakin University Research Data Store (RDS) Location: RDS29448-Alex-Cummaudo-PhD/results/fse2020

8. Additional notices

This form must be retained by the executive author, within the school or institute in which they are based.

If the publication is to be included as part of an HDR thesis, a copy of this form must be included in the thesis with the publication.

Deakin University Authorship Procedure

Schedule A: Authorship Statement

1. Details of the publication and executive author

Title of publication	Merging Intelligent API Responses Using a Proportional Representation Approach
Publication details	Presented at the 19th International Conference on Web Engineering (ICWE), Daejeon, South Korea, 2019
Name of executive author	Tomohiro Otake
School/Institute/Division if at Deakin Organisation and address if non-Deakin	Faculty of Science, Engineering and Built Environment
Email or phone	tomohiro.otake@deakin.edu.au

2. Inclusion of publication in a thesis

Is it intended to include this publication in a higher degree by research (HDR) thesis?
*If Yes, please complete Section 3
 If No, go straight to Section 4.*

3. HDR thesis author's declaration

Name of HDR thesis author if different from above. (If the same, write "as above")	Alex Cummaudo
School/Institute/Division if at Deakin	Applied Artificial Intelligence Institute
Thesis title	Taming the Evolving Black Box: Towards Improved Integration and Documentation of Intelligent Web Services
If there are multiple authors, give a full description of HDR thesis author's contribution to the publication.	See page 2

I declare that the above is an accurate description of my contribution to this paper, and the contributions of other authors are as described below.

Signed:



Dated: 2 August 2019

4. Description of all author contributions

Name and affiliation of author 1 Tomohiro Otake
Faculty of Science, Engineering and Built Environment
Deakin University

Contribution of author 1 Tomohiro Otake designed a detailed methodology for data collection in the primary experiment of this work. He conducted all data collection via a data-collection instrument he designed and implemented and performed a majority of data analysis. He drafted the full manuscript and made further revisions, modifications and prepared the camera ready version for publication in the conference proceedings.

Name and affiliation of author 2 Alex Cummaudo
Applied Artificial Intelligence Institute
Deakin University

Contribution of author 2 Alex Cummaudo's primary contribution to this work was the conception and writing up of the motivating sections in the manuscript. He additionally contributed to detailed editing of the manuscripting to make further revisions and modifications and implemented reviewer feedback.

Name and affiliation of author 3 Mohamed Abdelrazek
Faculty of Science, Engineering and Built Environment
Deakin University

Contribution of author 3 Mohamed Abdelrazek contributed to the initial conception of this project by providing high-level guidance over overview of what the project and its experiments should comprise of. Mohamed also contributed to detailed revisions of the initial manuscripts, and assisted in advising Tomohiro Otake on improved analytical insight into the collected results, and implementing reviewer feedback.

Name and affiliation of author 4 Rajesh Vasa
Faculty of Science, Engineering and Built Environment
Deakin University

Contribution of author 4 Rajesh Vasa provided high-level oversight of the project. He contributed to detailed reviews of the methodology and manuscript.

Name and affiliation of author 5 John Grundy
Faculty of Information Technology
Monash University

Contribution of author 5 John Grundy provided high-level oversight of the project. He contributed to detailed reviews of the methodology and manuscript.

5. Author declarations

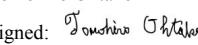
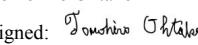
I agree to be named as one of the authors of this work, and confirm:

- i. that I have met the authorship criteria set out in the Deakin University Research Conduct Policy,
- ii. that there are no other authors according to these criteria,
- iii. that the description in Section 4 of my contribution(s) to this publication is accurate,
- iv. that the data on which these findings are based are stored as set out in Section 7 below.

If this work is to form part of an HDR thesis as described in Sections 2 and 3, I further

- v. consent to the incorporation of the publication into the candidate's HDR thesis submitted to Deakin University and, if the higher degree is awarded, the subsequent publication of the thesis by the university (subject to relevant Copyright provisions).

Author 1

Tomohiro Ohtake

 Signed: 
 Dated: 2 August 2019

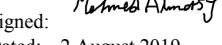
Author 2

Alex Cummaudo

 Signed: 
 Dated: 2 August 2019

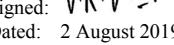
Author 3

Mohamed Abdelrazeck

 Signed: 
 Dated: 2 August 2019

Author 4

Rajesh Vasa

 Signed: 
 Dated: 2 August 2019

Author 5

John Grundy

 Signed: 
 Dated: 2 August 2019

6. Other contributor declarations

There are no other contributors for this publication to declare.

7. Data storage

The original data for this project are stored in the following locations. (The locations must be within an appropriate institutional setting. If the executive author is a Deakin staff member and data are stored outside Deakin University, permission for this must be given by the Head of Academic Unit within which the executive author is based.)

Data format	Comma separated values (CSV)
Storage location	Deakin University Research Data Store (RDS) Location: RDS29448-Alex-Cummaudo-PhD/results/icwe19

8. Additional notices

This form must be retained by the executive author, within the school or institute in which they are based.

If the publication is to be included as part of an HDR thesis, a copy of this form must be included in the thesis with the publication.

Deakin University Authorship Procedure

Schedule A: Authorship Statement

1. Details of the publication and executive author

Title of publication	Ranking Computer Vision Service Issues using Emotion
Publication details	Presented at the 5th International Workshop on Emotion Awareness in Software Engineering, Seoul, South Korea, 2020
Name of executive author	Maheswaree K Curumsing
School/Institute/Division if at Deakin	Applied Artificial Intelligence Institute
Organisation and address if non-Deakin	
Email or phone	m.curumsing@deakin.edu.au

2. Inclusion of publication in a thesis

Is it intended to include this publication in a higher degree by research (HDR) thesis?
*If Yes, please complete Section 3
 If No, go straight to Section 4.*

3. HDR thesis author's declaration

Name of HDR thesis author if different from above. <i>(If the same, write "as above")</i>	Alex Cummaudo
School/Institute/Division if at Deakin	Applied Artificial Intelligence Institute
Thesis title	Taming the Evolving Black Box: Towards Improved Integration and Documentation of Intelligent Web Services
If there are multiple authors, give a full description of HDR thesis author's contribution to the publication.	See page 2

I declare that the above is an accurate description of my contribution to this paper, and the contributions of other authors are as described below.

Signed:



Dated: 31 January 2020

4. Description of all author contributions

Name and affiliation of author 1	Maheswaree K Curumsing Applied Artificial Intelligence Institute Deakin University
Contribution of author 1	Maheswaree Curumsing contributed to the fleshing out of the project concept and coordinating the work.. Maheswaree's expertise in emotion classification was leveraged in the paper, particularly around the background sections and in deciding the correct frameworks to classify posts. She conducted extensive literature reviews for this paper. Maheswaree drafted the introduction, background, part of the methodology and discussion. She was involved in classifying emotions within Stack Overflow posts for inter-rater reliability. She made further revisions to the manuscript and provided modifications where needed.
Name and affiliation of author 2	Alex Cummaudo Applied Artificial Intelligence Institute Deakin University
Contribution of author 2	Alex Cummaudo produced the data set of Stack Overflow posts used for analysis within this paper. He drafted the methodology section that details how this data set was produced. Additionally, he drafted the threats to validity section. He reviewed the entire paper and made contributions to the findings and discussion sections. He set up and conducted inter-rater reliability with two additional raters (Maheswaree and Ulrike Maria). He performed inter-rater reliability statistics against the three raters and against the automatic classifications made from EmoTxt. He prepared the graphs and tables for review, prepared the paper for submission, and ensured the paper was formatted to the guidelines and page limit. Alex made most of the contribution to the paper in terms of content.
Name and affiliation of author 3	Ulrike Maria Graestch Applied Artificial Intelligence Institute Deakin University
Contribution of author 3	Ulrike Maria's contributed to the initial conception of the project and performed the automatic EmoTxt classifier classifications on our Stack Overflow data set, which involved downloading and installing EmoTxt and adapting our data set to be compatible with EmoTxt. She drafted the findings and discussion sections based on the output from the EmoTxt classifier, including constructing the graphs and tables in the paper. Ulrike Maria also conducted a literature review into automatic emotion classifiers into Stack Overflow posts. She extracted the quotes from posts as presented in Table 3.
Name and affiliation of author 4	Scott Barnett Applied Artificial Intelligence Institute Deakin University
Contribution of author 4	Scott Barnett's contribution involved drafting the abstract, conclusion and reviewing the entire manuscript for proofreading.

Scott also contributed in the initial conception of the project by outlining techniques used to run the experiment.

Name and affiliation of author 5

Rajesh Vasa
Applied Artificial Intelligence Institute
Deakin University

Contribution of author 5

Rajesh Vasa contributed to the initial conception of this project by providing high-level guidance over overview of what the project and its experiments should comprise of. Rajesh also contributed to detailed revisions of the initial manuscripts, and assisted in advising Alex Cummaudo on improved analytical insight into the collected results. Rajesh Vasa is the primary supervisor of Alex Cummaudo.

5. Author declarations

I agree to be named as one of the authors of this work, and confirm:

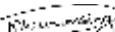
- i. that I have met the authorship criteria set out in the Deakin University Research Conduct Policy,
- ii. that there are no other authors according to these criteria,
- iii. that the description in Section 4 of my contribution(s) to this publication is accurate,
- iv. that the data on which these findings are based are stored as set out in Section 7 below.

If this work is to form part of an HDR thesis as described in Sections 2 and 3, I further

- v. consent to the incorporation of the publication into the candidate's HDR thesis submitted to Deakin University and, if the higher degree is awarded, the subsequent publication of the thesis by the university (subject to relevant Copyright provisions).

Author 1

Maheswaree K Curumsing

Signed: 

Dated: 31 January 2020

Author 2

Alex Cummaudo

Signed: 

Dated: 31 January 2020

Author 3

Ulrike Maria Graestch

Signed: 

Dated: 31 January 2020

Author 4

Scott Barnett

Signed: 
[
Dated: 31 January 2020

Author 5

Rajesh Vasa

Signed: 
Dated: 31 January 2020

6. Other contributor declarations

There are no other contributors for this publication to declare.

7. Data storage

The original data for this project are stored in the following locations. (The locations must be within an appropriate institutional setting. If the executive author is a Deakin staff member and data are stored outside Deakin University, permission for this must be given by the Head of Academic Unit within which the executive author is based.)

Data format	Comma separated values (CSV), Excel Spreadsheet
Storage location	Deakin University Research Data Store (RDS) Location: RDS29448-Alex-Cummaudo-PhD/results/semotion20

8. Additional notices

This form must be retained by the executive author, within the school or institute in which they are based.

If the publication is to be included as part of an HDR thesis, a copy of this form must be included in the thesis with the publication.

APPENDIX E

Ethics Clearance



Rajesh Vasa and Alex Cummaudo
Applied Artificial Intelligence Institute (A²I²)
C.c Mohamed Abdelrazek, Andrew Cain

2 May 2019

Dear Rajesh and Alex

STEC-11-2019-CUMMAUDO titled "*Developer opinions towards the importance of web API documentation recommendations*"

Thank you for submitting the above project for consideration by the Faculty Human Ethics Advisory Group (HEAG). The HEAG recognised that the project complies with the National Statement on Ethical Conduct in Human Research (2007) and has approved it. You may commence the project upon receipt of this communication.

The approval period is for three years until **02/05/22**. It is your responsibility to contact the Faculty HEAG immediately should any of the following occur:

- Serious or unexpected adverse effects on the participants
- Any proposed changes in the protocol, including extensions of time
- Any changes to the research team or changes to contact details
- Any events which might affect the continuing ethical acceptability of the project
- The project is discontinued before the expected date of completion.

You will be required to submit an annual report giving details of the progress of your research. Please forward your first annual report on **02/05/20**. Failure to do so may result in the termination of the project. Once the project is completed, you will be required to submit a final report informing the HEAG of its completion.

Please ensure that the Deakin logo is on the Plain Language Statement and Consent Forms. You should also ensure that the project ID is inserted in the complaints clause on the Plain Language Statement, and be reminded that the project number must always be quoted in any communication with the HEAG to avoid delays. All communication should be directed to sciethic@deakin.edu.au

The Faculty HEAG and/or Deakin University Human Research Ethics Committee (HREC) may need to audit this project as part of the requirements for monitoring set out in the National Statement on Ethical Conduct in Human Research (2007).

If you have any queries in the future, please do not hesitate to contact me.

We wish you well with your research.

Kind regards

A handwritten signature in blue ink that reads "Teresa Treffry".

Teresa Treffry
Secretary, Human Ethics Advisory Group (HEAG)
Faculty of Science Engineering & Built Environment



Rajesh Vasa, Mohamed Abdelrazeq, Andrew Cain, Scott Barnett, Alex Cummaudo
Applied Artificial Intelligence Institute (A²I²) (G)

23rd July 2019

Dear Rajesh and research team

STEC-39-2019-CUMMAUDO titled "*Factors that impact the learnability, interpretability and adoption of intelligent services*".

Thank you for submitting the above project for consideration by the Faculty Human Ethics Advisory Group (HEAG). The HEAG recognised that the project complies with the National Statement on Ethical Conduct in Human Research (2007) and has approved it. You may commence the project upon receipt of this communication.

The approval period is for three years until 23/07/22. It is your responsibility to contact the Faculty HEAG immediately should any of the following occur:

- Serious or unexpected adverse effects on the participants
- Any proposed changes in the protocol, including extensions of time
- Any changes to the research team or changes to contact details
- Any events which might affect the continuing ethical acceptability of the project
- The project is discontinued before the expected date of completion.

You will be required to submit an annual report giving details of the progress of your research. Please forward your first annual report on 23/07/20. Failure to do so may result in the termination of the project. Once the project is completed, you will be required to submit a final report informing the HEAG of its completion.

Please ensure that the project number must always be quoted in any communication with the HEAG to avoid delays. All communication should be directed to sciethic@deakin.edu.au.

The Faculty HEAG and/or Deakin University Human Research Ethics Committee (HREC) may need to audit this project as part of the requirements for monitoring set out in the National Statement on Ethical Conduct in Human Research (2007).

If you have any queries in the future, please do not hesitate to contact me.

We wish you well with your research.

Kind regards

Rickie Morey

Rickie Morey
Senior Research Administration Officer
Representing the Human Ethics Advisory Group (HEAG)
Faculty of Science Engineering & Built Environment